

Reliable Execution of MPI Applications

Andrew Lumsdaine

Brian Barrett

Jeff Squyres

Computer Science Department

Indiana University

Bloomington, IN

Overview

- LAM/MPI background
- Wherefore fault tolerance / reliability?
- Fault tolerance currently provided by LAM/MPI
- Behavior of MPI in presence of failure
- Present and future work
- Conclusions

LAM/MPI background

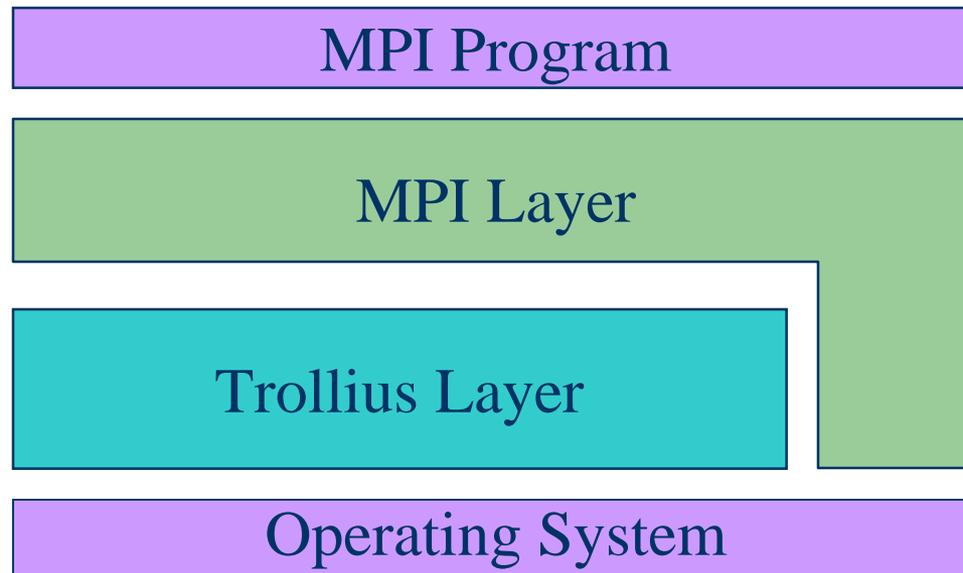
- Author history:
 - Ohio Supercomputing Center
 - University of Notre Dame
 - Indiana University
- <http://www.lam-mpi.org/>

LAM/MPI Background

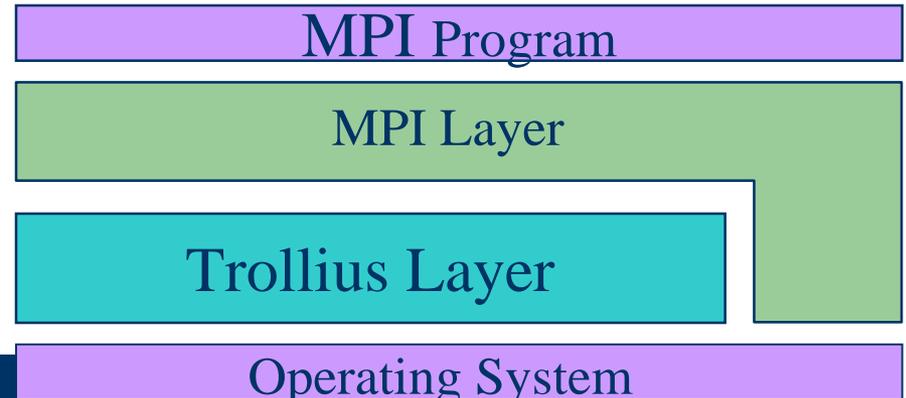
- Open source implementation of MPI
- Contains all of MPI-1 and most of MPI-2
 - Dynamic processes
 - C++ bindings
 - I/O
 - One-sided communication
- Runs on just about any POSIX system

LAM/MPI Architecture

- Layered on Trollius
 - Parallel RTE developed before the MPI standard

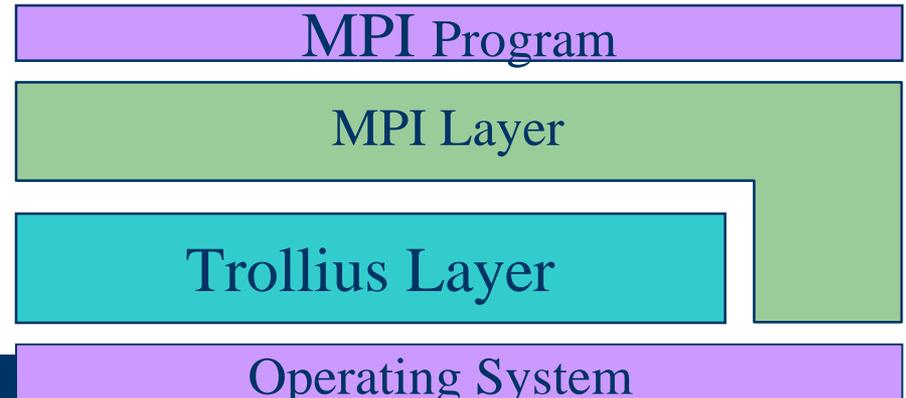


Trollius and MPI



- Trollius provides a high level of infrastructure
 - Uses a small user-level daemon running on each node: the lamd
 - Reliable point-to-point communication layer (UDP)
 - Process control environment
 - Remote I/O with Unix-like semantics
 - Minimal fault detection
 - Communication tracing system

Trollius and MPI



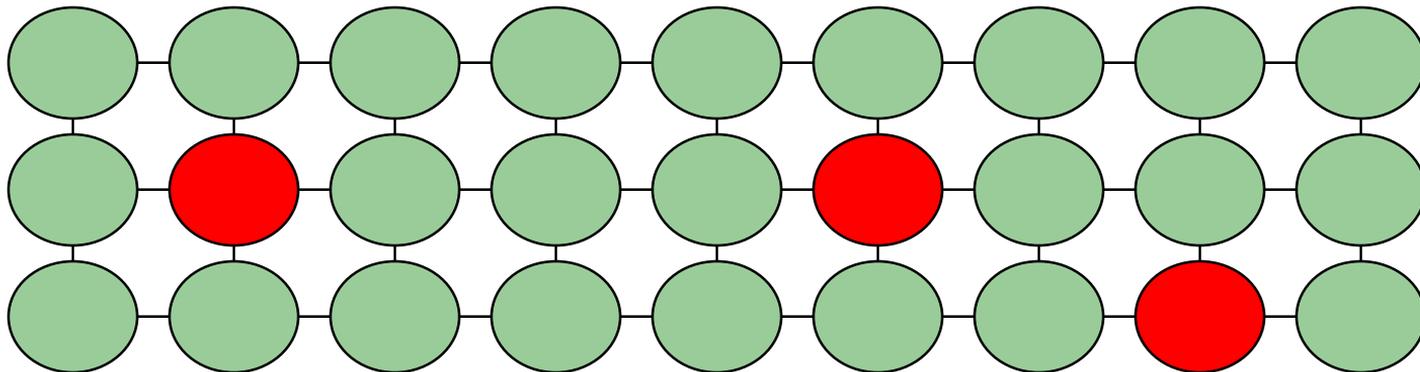
- MPI layer uses many of the services provided by Trollius
 - Uses Trollius' out-of-band communication layer for meta data
 - Process control to launch and and reliably take down jobs
 - Job status information maintained by daemons
- Optionally uses Trollius layer for MPI communication (lamd RPI)

Why Fault Tolerance?

- COTS clusters are clearly the “Big Thing”
- Failures becoming common as code moves from “Big Iron” to large clusters
 - Hardware failure
 - OS failure
 - Kicking out the power cord
- Clusters growing: 1000’s of nodes desirable
- Losing entire job because one node failed costly in time and CPU cycles

What Do We Want?

- Ability to “keep going” after a failure
- Failures we do want to recover from:
 - Unresponsive nodes
 - Network link failures

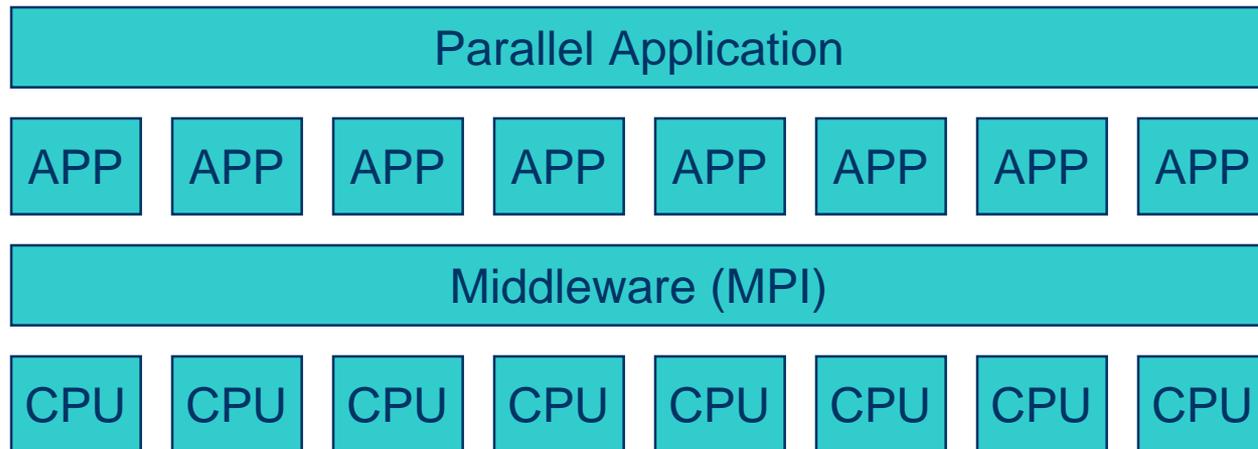


What Do We *Not* Want?

- Failures we do *not* want to recover from:
 - Data integrity (memory/transmission errors, etc.)
 - Program errors (seg faults, divide by 0, etc.)
 - Byzantine errors
- These are considered to be user problems

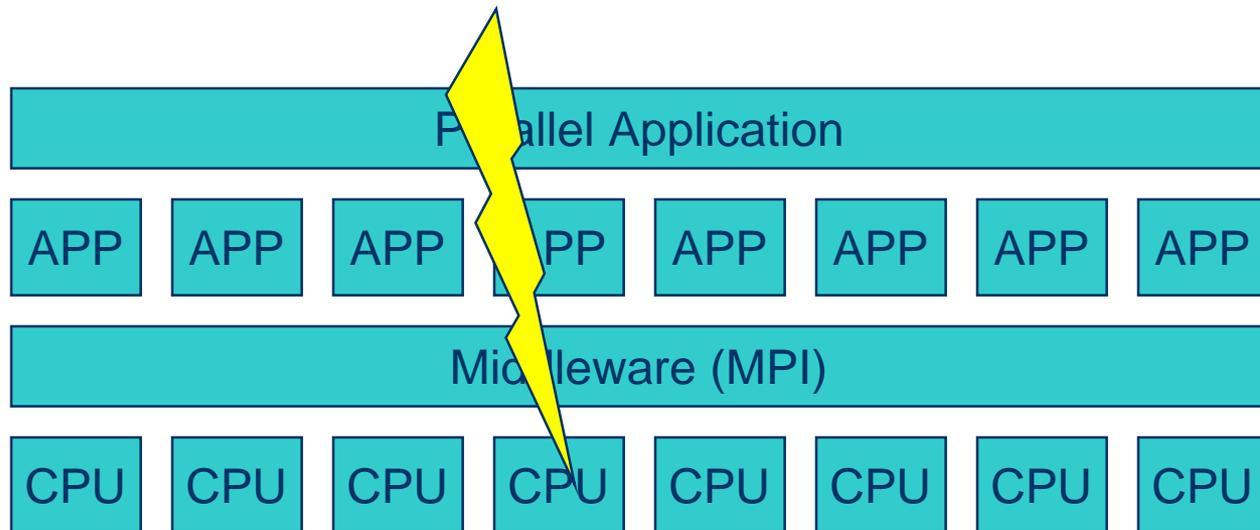
Approaches to Reliability

- Both application and middleware are involved



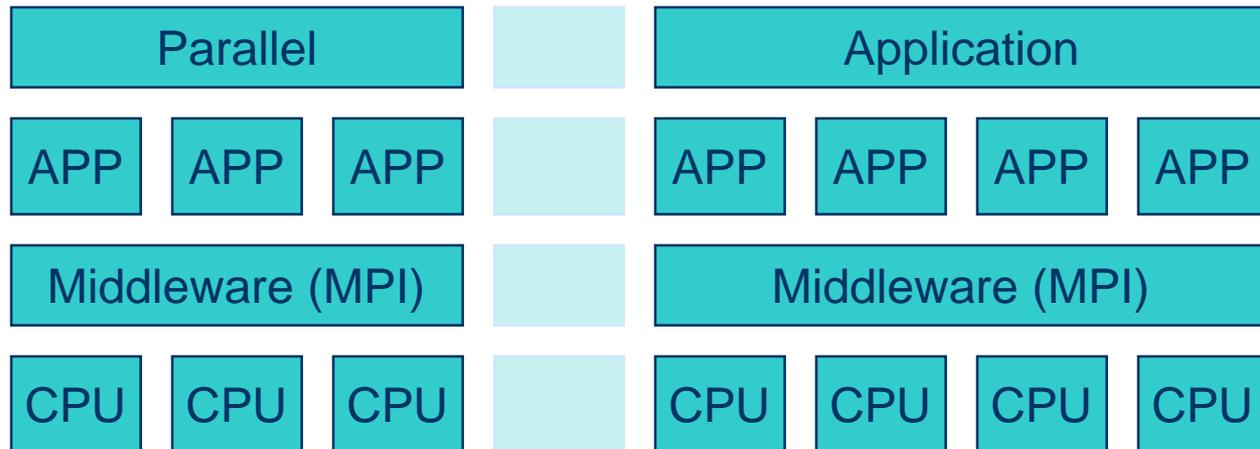
Approaches to Reliability

- On failure, state may be lost



Approaches to Reliability

- On failure, state may be lost



Approaches to Reliability

- Use a stateless programming model
 - Assist application in reconstructing state
 - Periodically save state (checkpoint)
 - Replicate state
-
- These approaches imply requirements on the application and on the middleware

Goals for LAM/MPI Reliability

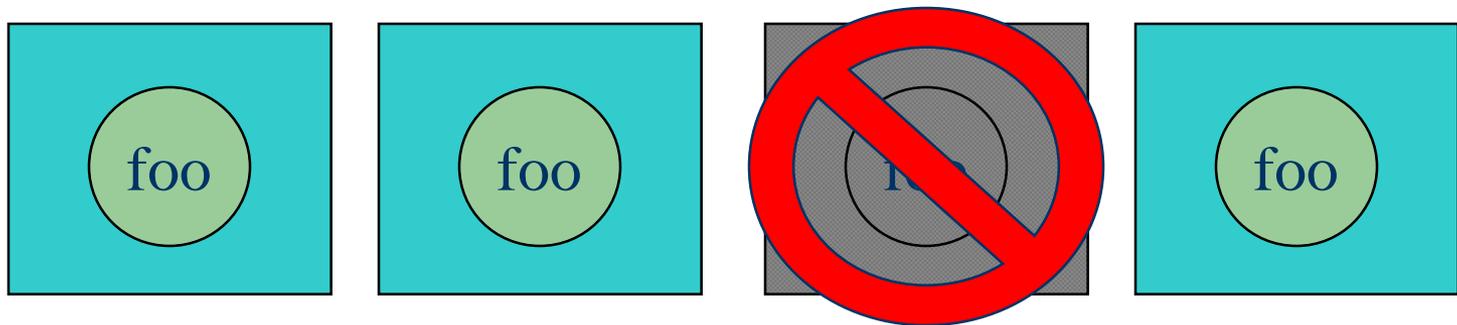
- Use a stateless programming model: today
 - Assist application in reconstructing state: RSN
 - Periodically save state (checkpoint): fall
 - Replicate state
-
- Need to be cognizant of what approach the application is able / willing to use

Goals for LAM/MPI Reliability

- Ensure stability of MPI layer and LAM RTE in the presence of failures
- Detect failures and notify MPI application
- Provide status information to MPI programs
- Recover from transient failures
- User codes remain portable
 - Can compile/link to other MPI implementations*
 - ...but will not utilize reliability extensions
- Provide library for reliable MPI operations

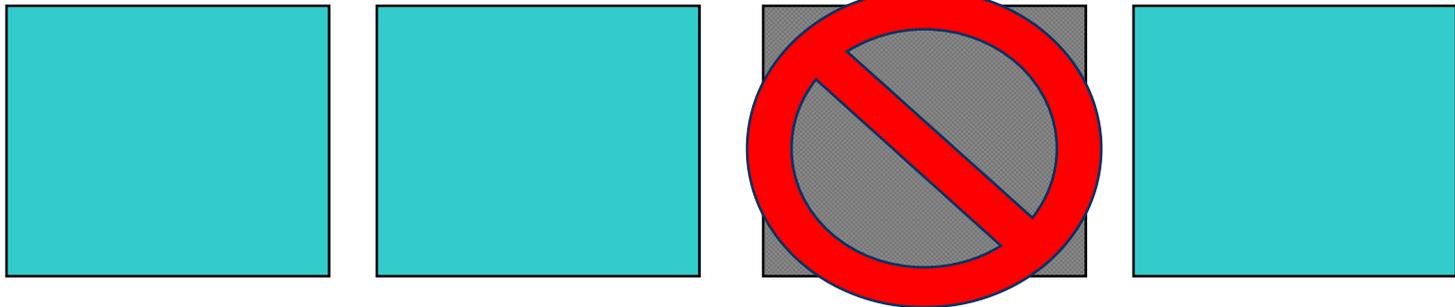
Existing Fault Tolerance in LAM

- Trollius layer can detect failure of a remote node
- Minimal recovery ability from node failure
 - Node is removed from LAM universe
 - Running MPI application is terminated



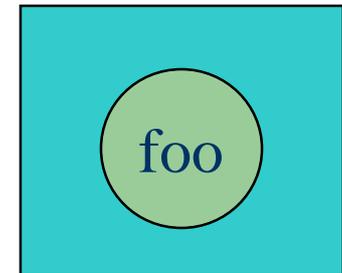
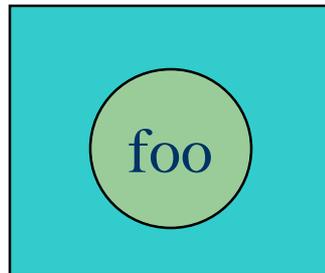
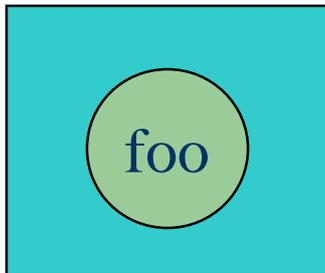
Existing Fault Tolerance in LAM

- Trollius layer can detect failure of a remote node
- Minimal recovery ability from node failure
 - Node is removed from LAM universe
 - Running MPI application is terminated



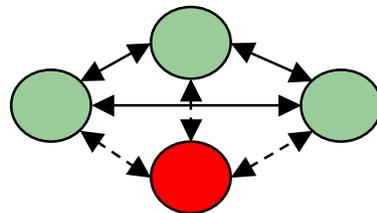
Existing Fault Tolerance in LAM

- Trollius layer can detect failure of a remote node
- Minimal recovery ability from node failure
 - Node is removed from LAM universe
 - Running MPI application is terminated
 - Next mpirun recognizes smaller universe



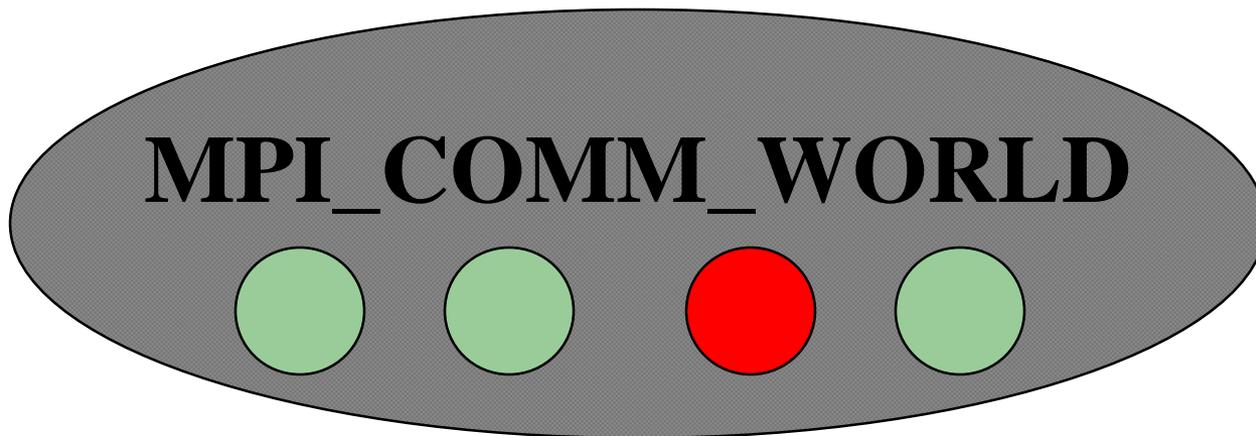
Infrastructure Failure Detection

- Detects failures during normal communication
- Uses “ping” messages to test data links that have not been used recently
- If out-of-band channel is broken between two nodes, remote node considered dead
- Daemons are fully connected (UDP); each individually notice when one goes down



MPI Communicators and Faults

- MPI communicators are static environments
 - A fixed set of MPI processes
 - If a process in a communicator dies, many MPI operations become undefined (e.g., collectives)



Surviving a Process Failure

- Possible implement a program that can survive failures during a MPI job
 - Use MPI_COMM_WORLD size of one
 - Use MPI_COMM_SPAWN to launch worker jobs
 - Pair-wise communicators
 - Spawned processes can die and rest will continue
- Limited solution – must be carefully written with MPI-2 dynamic process control
- Example included with LAM/MPI distribution

Process Failure Example

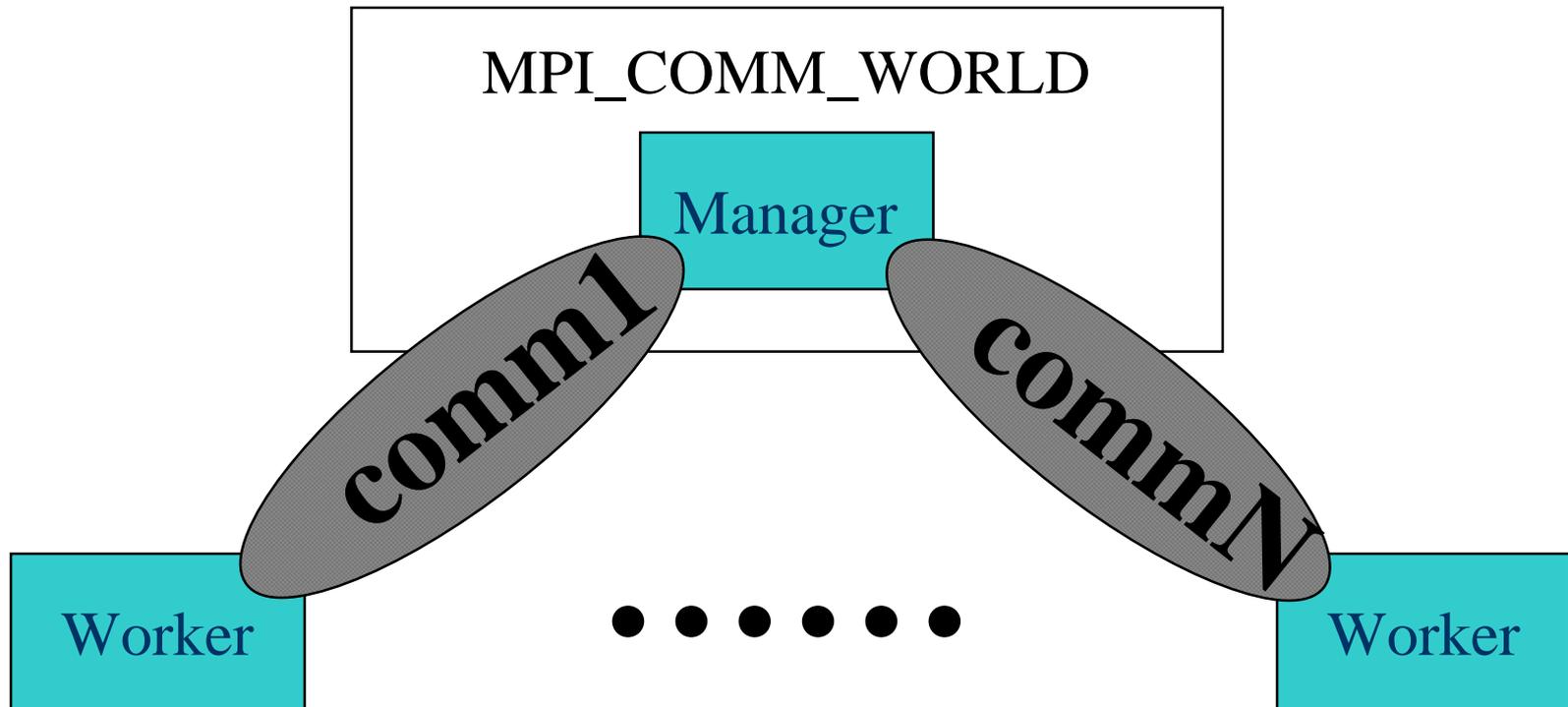
- % ./manager

MPI_COMM_WORLD

Manager

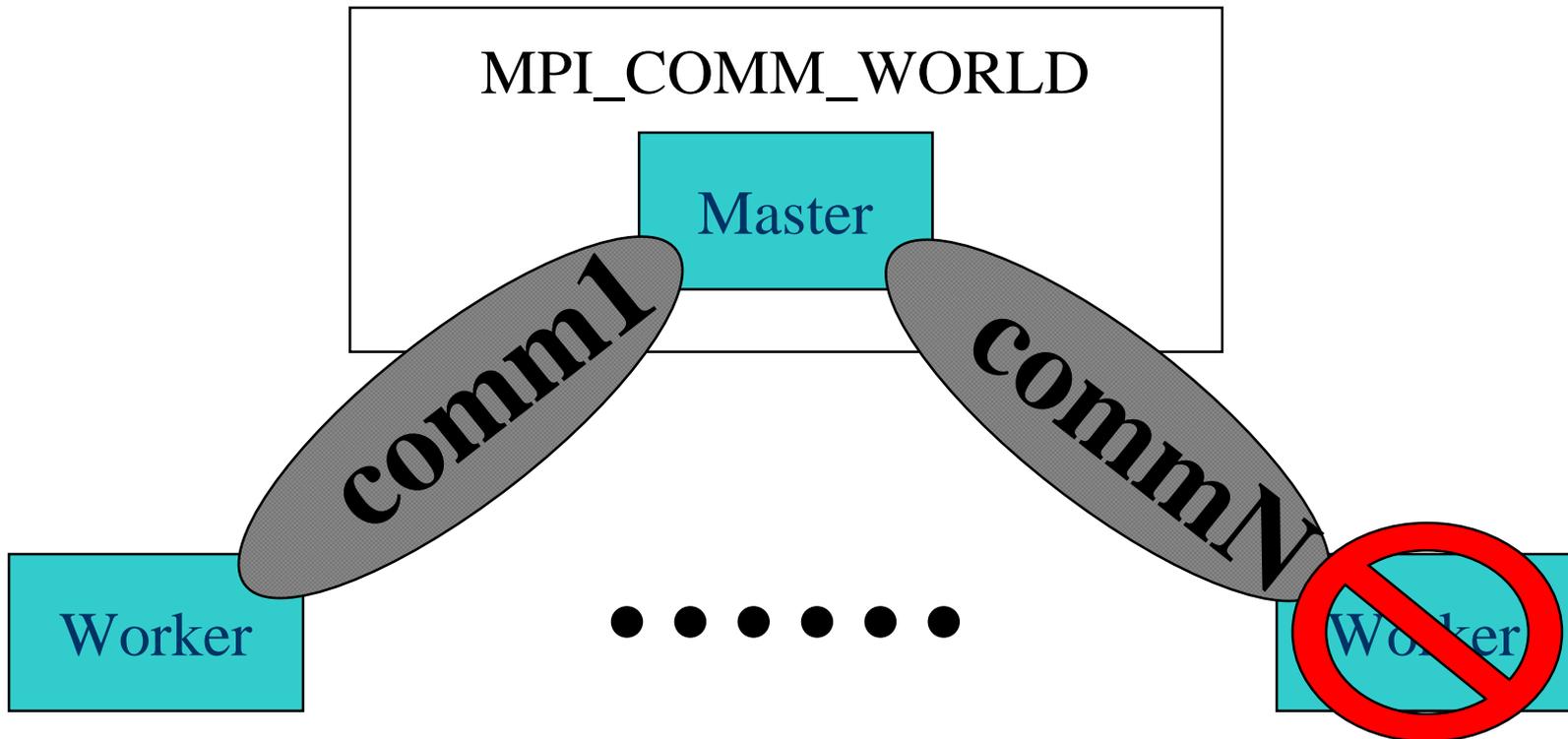
Process Failure Example

- % ./manager



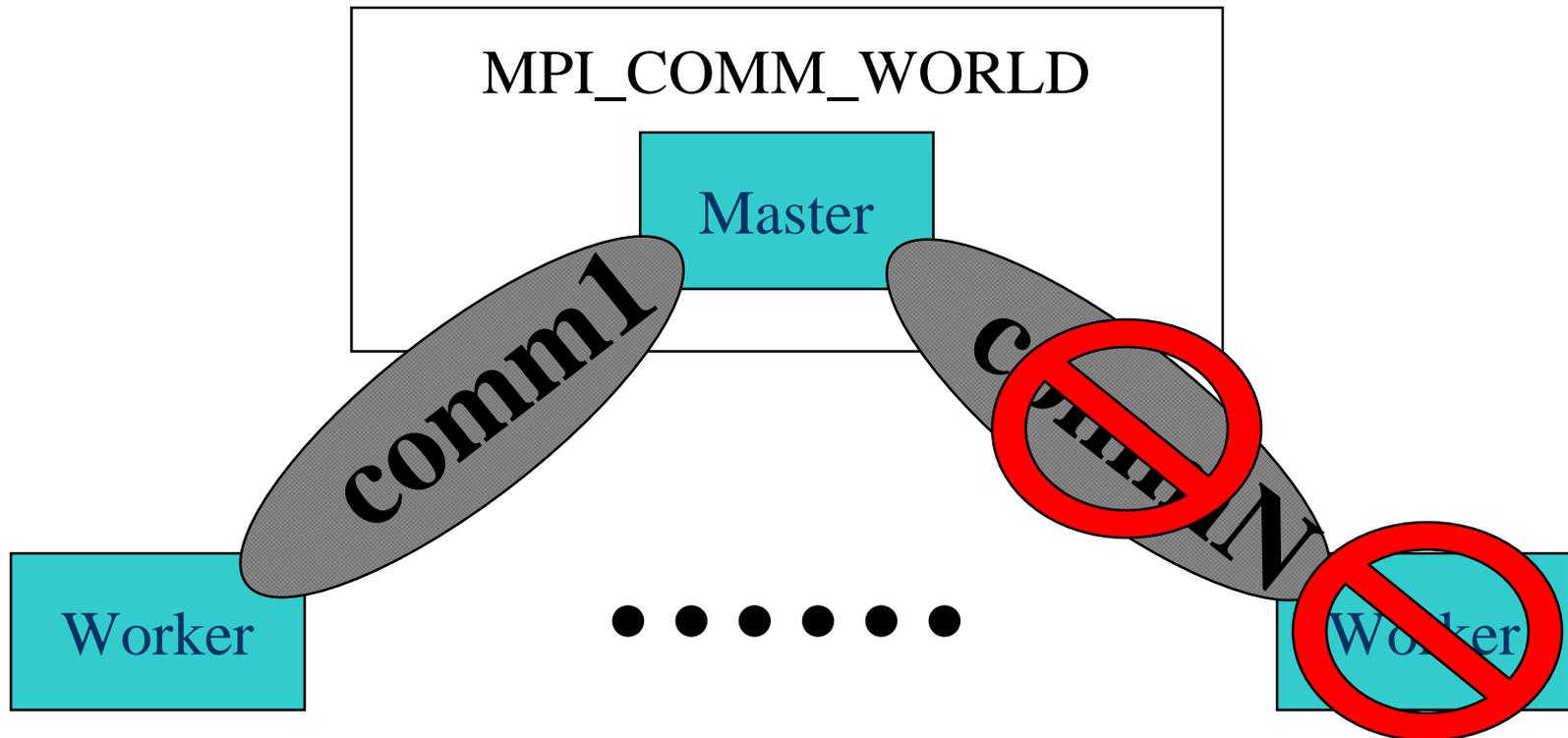
Process Failure Example

- % ./manager



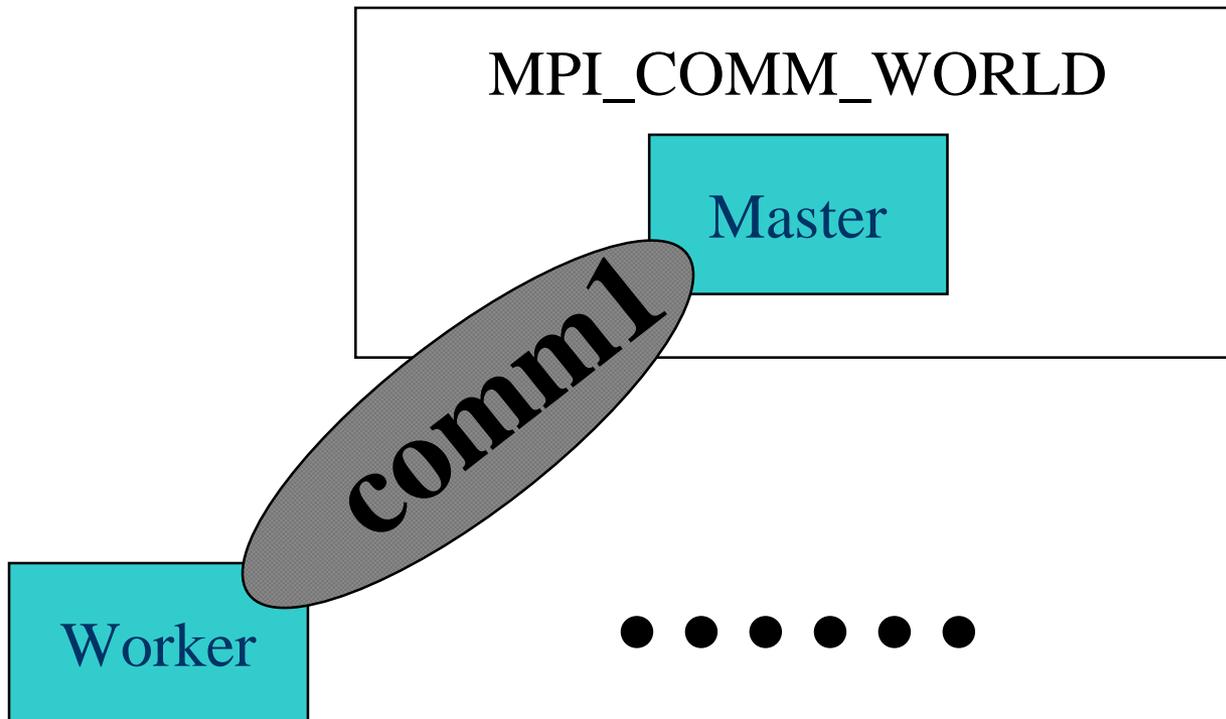
Process Failure Example

- % ./manager



Process Failure Example

- % ./manager



Current Work

- Improve infrastructure for fault detection
 - More reliable detection of faults
 - LAM RTE behaves correctly in presence of faults
- Implement scheme for node to be notified of another node's failure
- Audit code for proper behavior of LAM (especially MPI layer) in presence of faults
- Reliability library

Current Work

- Asynchronous MPI notification of process failure
- Define behavior of MPI layer in presence of faults
- Complete code audit and testing of LAM
- Consider potential recovery from transient failure
- Model of integration into an application

Infrastructure Fault Detection Improvements

- Improve current “least recently used” algorithm for fault detection by introducing topology-based testing
- Allow nodes to notify neighbor nodes of a detected fault, reducing time spent waiting for an NACK from a dead node
- Improve handling of data structures once a node is dead (in order to allow recovery later)

Infrastructure Fault Detection Improvements (cont.)

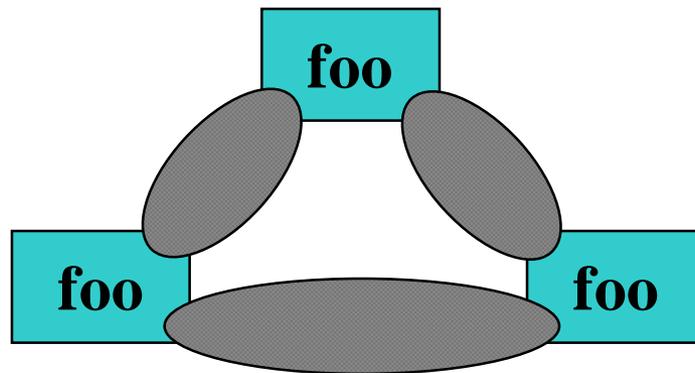
- mpirun needs updating to perform consistently in presence of faults
 - Currently, mpirun may hang if fault occurs before local lamd notices
 - mpirun's behavior depends on when failure occurs
 - Two separate conditions: before and after all processes finish MPI_INIT
 - Allow user to specify whether to continue or abort
- Other LAM utilities must be tested as well

Interacting with the Application

- Detecting a fault useless unless we notify the MPI programs about the fault
- Asynchronous notification ideal – something similar to a signal handler
- Options:
 - Use MPI attribute on `MPI_COMM_WORLD` to register a callback function for faults
 - Add a LAM-specific function to register a callback function

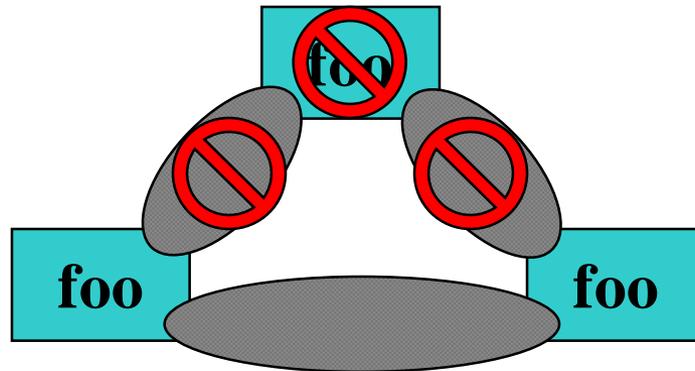
Application Programming Model

- Pair-wise communicators
 - When one process dies, can discard all communicators that it is in
 - All other processes still have healthy communicators



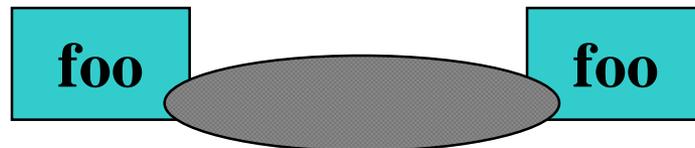
Application Programming Model

- Pair-wise communicators
 - When one process dies, can discard all communicators that it is in
 - All other processes still have healthy communicators



Application Programming Model

- Pair-wise communicators
 - When one process dies, can discard all communicators that it is in
 - All other processes still have healthy communicators



Application State

- Application can maintain N-way communicators
- On failure, contents of wounded communicator are used to make new one
- Requires `COMM_FREE`, `COMM_SPLIT` to work with wounded communicator

Is Anybody Out There?

- In order to properly recover from a fault, a process must know who is still alive
- Possible ways to implement data retrieval:
 - Special attribute on MPI_COMM_WORLD that will return list of processes who are still alive
 - Special attribute on MPI_COMM_WORLD that will return list of processes who died
 - Additional, non-portable, function calls to obtain the lists of information

Is Anybody Out There? (cont.)

- Using attributes:
 - MPI-portable
 - A well-written program can still compile and run under other MPI implementations
- Using LAM-specific functions
 - Could be portable with `#if` statements, and therefore equivalent to attributes
 - Compile-time decision vs. run-time decision

MPI and Faults

- Nothing said about fault tolerance in the MPI standard
 - Deliberate choice – hard to define
 - Behavior of all MPI functions and objects must be specified in order to ensure programs work “as expected” in presence of faults
- Goal: Specification for fault tolerance in LAM will still comply with MPI standard
- Exact semantics and functionality still being researched

MPI Point-to-Point Operations

- Point-to-point operations involving a “down” process must fail
 - LAM will be able to continue correctly
 - Will gracefully fail and return an error
- Point-to-point operations to healthy processes in a “wounded” communicator will succeed

Collective Operations

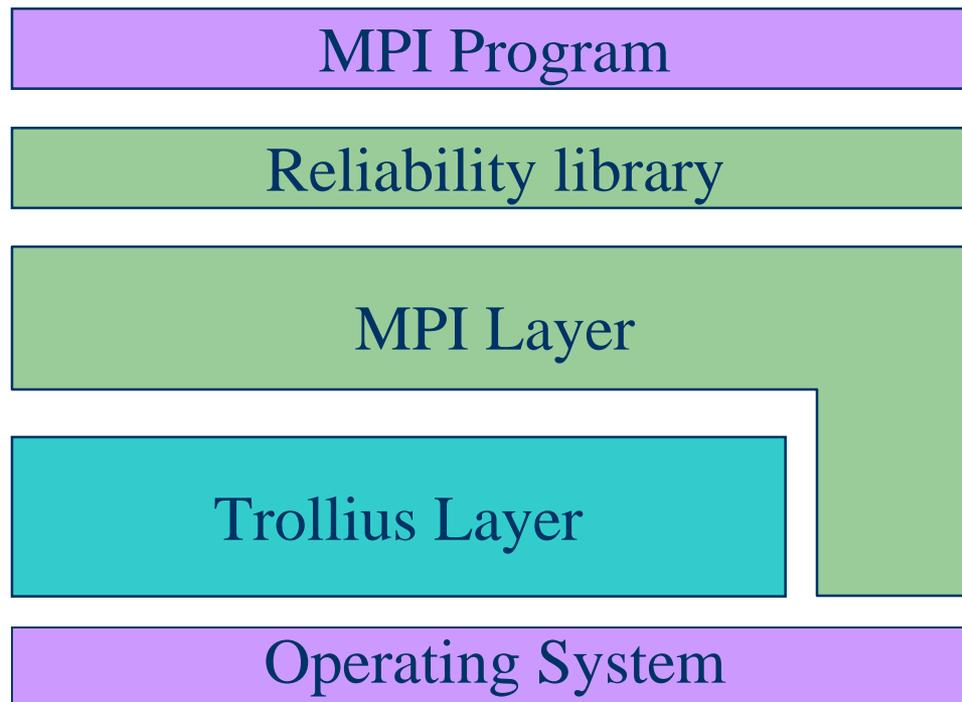
- Collectives on a communicator involving a “dead” process cannot succeed
 - Fail the entire collective
 - Possible for unexpected success with a failure
 - If a process completes its part in a collective operation and then fails, it is possible that not everyone will have finished the collective yet
 - The collective will still finish correctly
- Lots of bookkeeping when using a pair-wise communicator model

Reliability Library

- Absorb the burden of programming model
- Construct and maintain the pair-wise communicators
 - Give the illusion of communicators with many members
 - Perform the bookkeeping necessary for collective operations
 - Re-form wounded communicators when a process fails

Reliability Library

- Sits between user program and MPI library



Checkpoint/Restart

- Joint project with LBNL
- LBNL implementing kernel-level checkpoint restart for single processes
- Integrating with LAM/MPI to checkpoint LAM/MPI jobs
- Transparent to user
- Prototype planned for end of summer

Conclusions

- Several approaches to reliability (state recovery)
- Application and middleware interact accordingly
- LAM/MPI
 - Supports some reliability modes
 - Infrastructure improvements in progress
 - MPI layer improvements being studied
 - Checkpoint/restart under development
 - Reliability library under development
- Applications welcome!