MPI Tool Interfaces
A role model for other standards !?

Martin Schulz

Lawrence Livermore National Laboratory

LLNL-PRES-738989

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

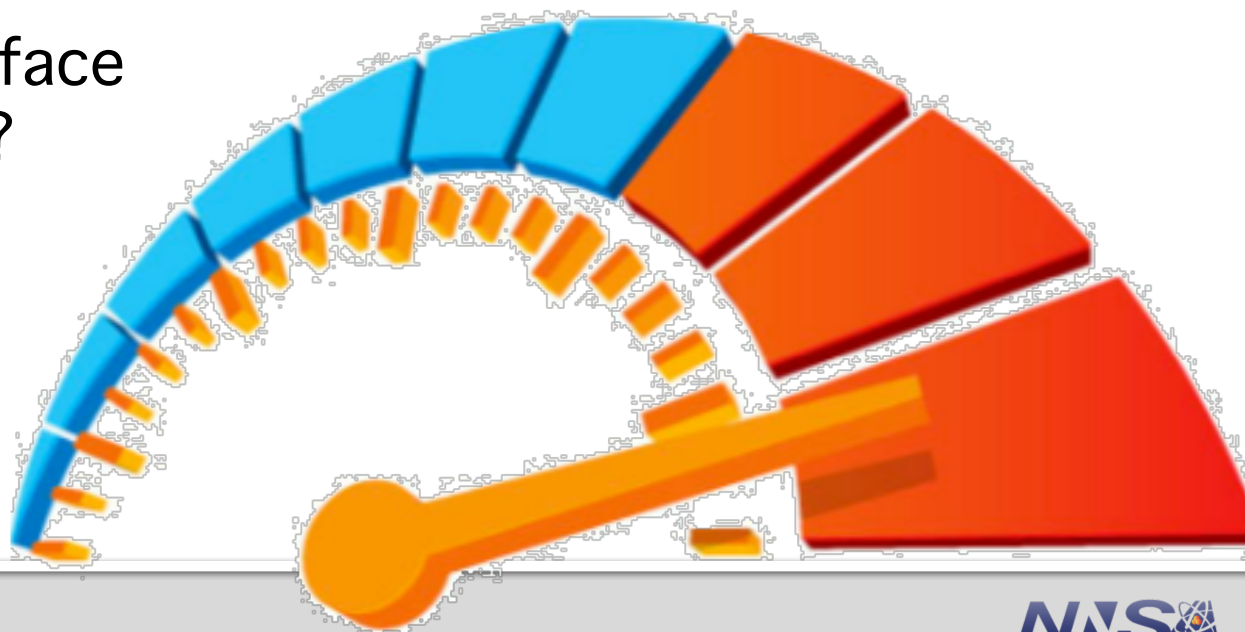
The MPI 1.0 Team Had a Lot of Foresight

People using MPI might care about performance
- After all, it's called High Performance Computing

Hence, people may want to measure performance
- Communication & synchronization is wasted time for computation
- Want to measure how much we waste

Why not add an interface to MPI to enable this?
- Sounds trivial, right?

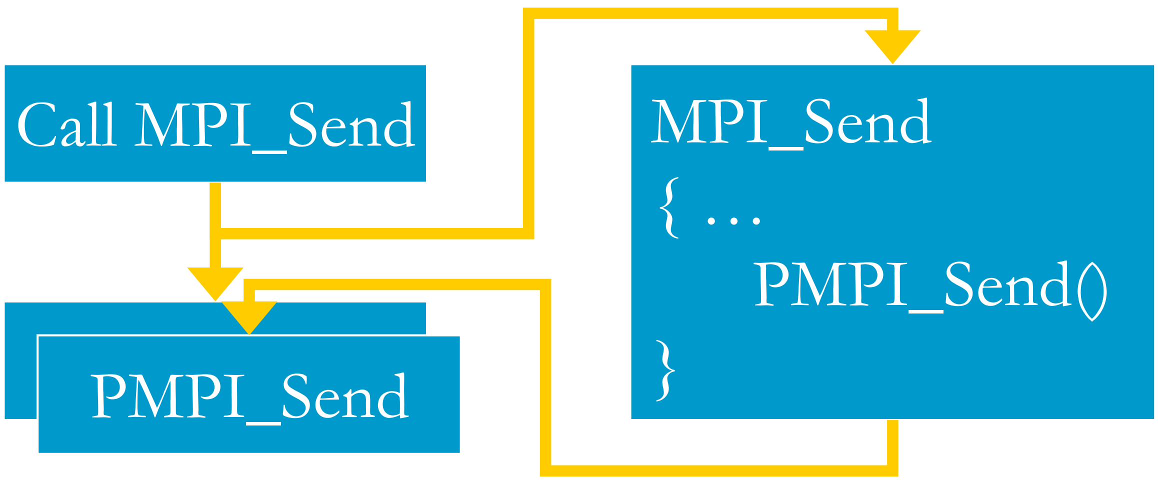
Still today very uncommon!



The MPI Profiling Interface

Simple support for interception of all MPI calls

– Enforced throughout the whole standard
– Coupled with name shifted interface



Easy to implement profiling tools

– Start timer on entry of MPI routine
– Stop timer on exit of MPI routine

The mpiP tool: Example of the Intended Effect

Intercepts all MPI API calls using PMPI

- Records number of invocations
- Measures time spent during MPI function execution
- Gathers data on communication volume
- Aggregates statistics over time

Several analysis options

- Multiple aggregations options/granularity
  - By function name or type
  - By source code location (call stack)
  - By process rank
- Adjustment of reporting volume
- Adjustment of call stack depth that is considered

Provides easy to use reports

http://mpip.sourceforge.net/

The mpiP tool: Example of the Intended Effect

```
bash-3.2$ srun –n4 smg2000
mpiP:
mpiP:
mpiP: mpiP V3.1.2 (Build Dec 16 2008/17:31:26)
mpiP: Direct questions and errors to mpip-
help@lists.sourceforge.net
mpiP:
Running with these driver parameters:
  (nx, ny, nz)    = (60, 60, 60)
  (Px, Py, Pz)    = (4, 1, 1)
  (bx, by, bz)    = (1, 1, 1)
  (cx, cy, cz)    = (1.000000, 1.000000, 1.000000)
  (n_pre, n_post) = (1, 1)
  dim             = 3
  solver ID       = 0
=============================================
Struct Interface:
=============================================
Struct Interface:
  wall clock time = 0.075800 seconds
  cpu clock time  = 0.080000 seconds
```

Header

```
=============================================
Setup phase times:
=============================================
SMG Setup:
  wall clock time = 1.473074 seconds
  cpu clock time  = 1.470000 seconds
=============================================
Solve phase times:
=============================================
SMG Solve:
  wall clock time = 8.176930 seconds
  cpu clock time  = 8.180000 seconds

Iterations = 7
Final Relative Residual Norm = 1.459319e-07

mpiP:
mpiP: Storing mpiP output in [./smg2000-p.4.11612.1.mpiP].
mpiP:
bash-3.2$
```

Output File

mpiP 101 / Output – Metadata

```
@ mpiP
@ Command : ./smg2000-p -n 60 60 60
@ Version                         : 3.1.2
@ MPIP Build date                 : Dec 16 2008, 17:31:26
@ Start time                      : 2009 09 19 20:38:50
@ Stop time                       : 2009 09 19 20:39:00
@ Timer Used                      : gettimeofday
@ MPIP env var                    : [null]
@ Collector Rank                  : 0
@ Collector PID                   : 11612
@ Final Output Dir                : .
@ Report generation               : Collective
@ MPI Task Assignment             : 0 hera27
@ MPI Task Assignment             : 1 hera27
@ MPI Task Assignment             : 2 hera31
@ MPI Task Assignment             : 3 hera31
```

mpiP 101 / Output – Overview

```
-------------------------------------------------------------
@--- MPI Time (seconds) -------------------------------------
-------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0       9.78       1.97    20.12
   1        9.8       1.95    19.93
   2        9.8       1.87    19.12
   3       9.77       2.15    21.99
   *       39.1       7.94    20.29
------------------------------------------------------------
```

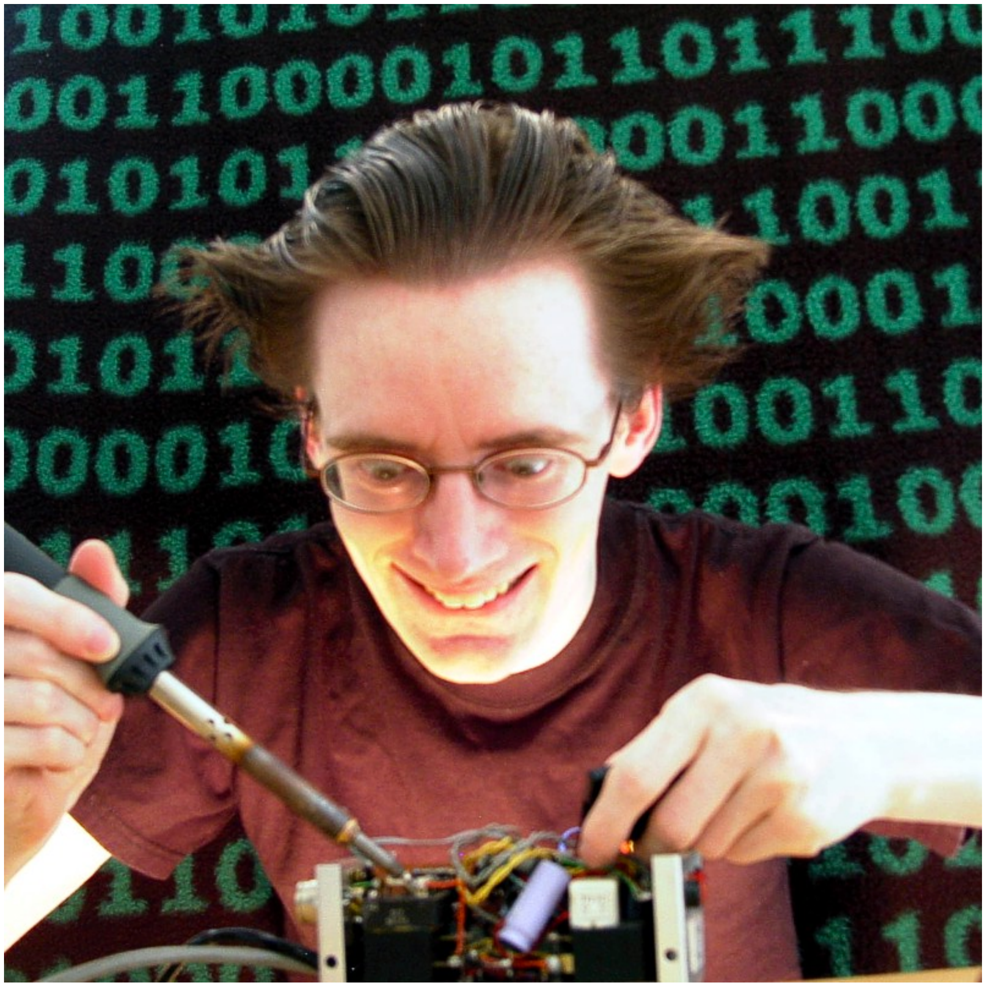
mpiP 101 / Output – Callsites

```
---------------------------------------------------------------------------
@--- Callsites: 23 --------------------------------------------------------
---------------------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct                  MPI_Call
  1   0 communication.c     1405 hypre_CommPkgUnCommit         Type_free
  2   0 timing.c             419 hypre_PrintTiming             Allreduce
  3   0 communication.c      492 hypre_InitializeCommunication Isend
  4   0 struct_innerprod.c   107 hypre_StructInnerProd         Allreduce
  5   0 timing.c             421 hypre_PrintTiming             Allreduce
  6   0 coarsen.c            542 hypre_StructCoarsen           Waitall
  7   0 coarsen.c            534 hypre_StructCoarsen           Isend
  8   0 communication.c     1552 hypre_CommTypeEntryBuildMPI   Type_free
  9   0 communication.c     1491 hypre_CommTypeBuildMPI        Type_free
 10   0 communication.c      667 hypre_FinalizeCommunication   Waitall
 11   0 smg2000.c            231 main                          Barrier
 12   0 coarsen.c            491 hypre_StructCoarsen           Waitall
 13   0 coarsen.c            551 hypre_StructCoarsen           Waitall
 14   0 coarsen.c            509 hypre_StructCoarsen           Irecv
 15   0 communication.c     1561 hypre_CommTypeEntryBuildMPI   Type_free
 16   0 struct_grid.c        366 hypre_GatherAllBoxes          Allgather
 17   0 communication.c     1487 hypre_CommTypeBuildMPI        Type_commit
 18   0 coarsen.c            497 hypre_StructCoarsen           Waitall
 19   0 coarsen.c            469 hypre_StructCoarsen           Irecv
 20   0 communication.c     1413 hypre_CommPkgUnCommit         Type_free
 21   0 coarsen.c            483 hypre_StructCoarsen           Isend
 22   0 struct_grid.c        395 hypre_GatherAllBoxes          Allgatherv
 23   0 communication.c      485 hypre_InitializeCommunication Irecv
---------------------------------------------------------------------------
```

mpiP 101 / Output – per Function Timing

```
---------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ---
---------------------------------------------------------------
Call                 Site       Time    App%    MPI%     COV
Waitall                10   4.4e+03   11.24   55.40     0.32
Isend                   3  1.69e+03    4.31   21.24     0.34
Irecv                  23       980    2.50   12.34     0.36
Waitall                12       137    0.35    1.72     0.71
Type_commit            17       103    0.26    1.29     0.36
Type_free               9      99.4    0.25    1.25     0.36
Waitall                 6      81.7    0.21    1.03     0.70
Type_free              15      79.3    0.20    1.00     0.36
Type_free               1      67.9    0.17    0.85     0.35
Type_free              20      63.8    0.16    0.80     0.35
Isend                  21        57    0.15    0.72     0.20
Isend                   7      48.6    0.12    0.61     0.37
Type_free               8      29.3    0.07    0.37     0.37
Irecv                  19      27.8    0.07    0.35     0.32
Irecv                  14      25.8    0.07    0.32     0.34
...
```

But then something happened …



Tool developers
got very creative!

The Profiling Interface can do so much more!

Record each invocation of an MPI routine
- Lead to broad range of trace tools (e.g., Jumpshot and Vampir)

Inspect message meta-data
- Lead to MPI correctness checkers (e.g., Marmot, Umpire, MUST)

Inspect message contents
- Transparent checksums for message transfers

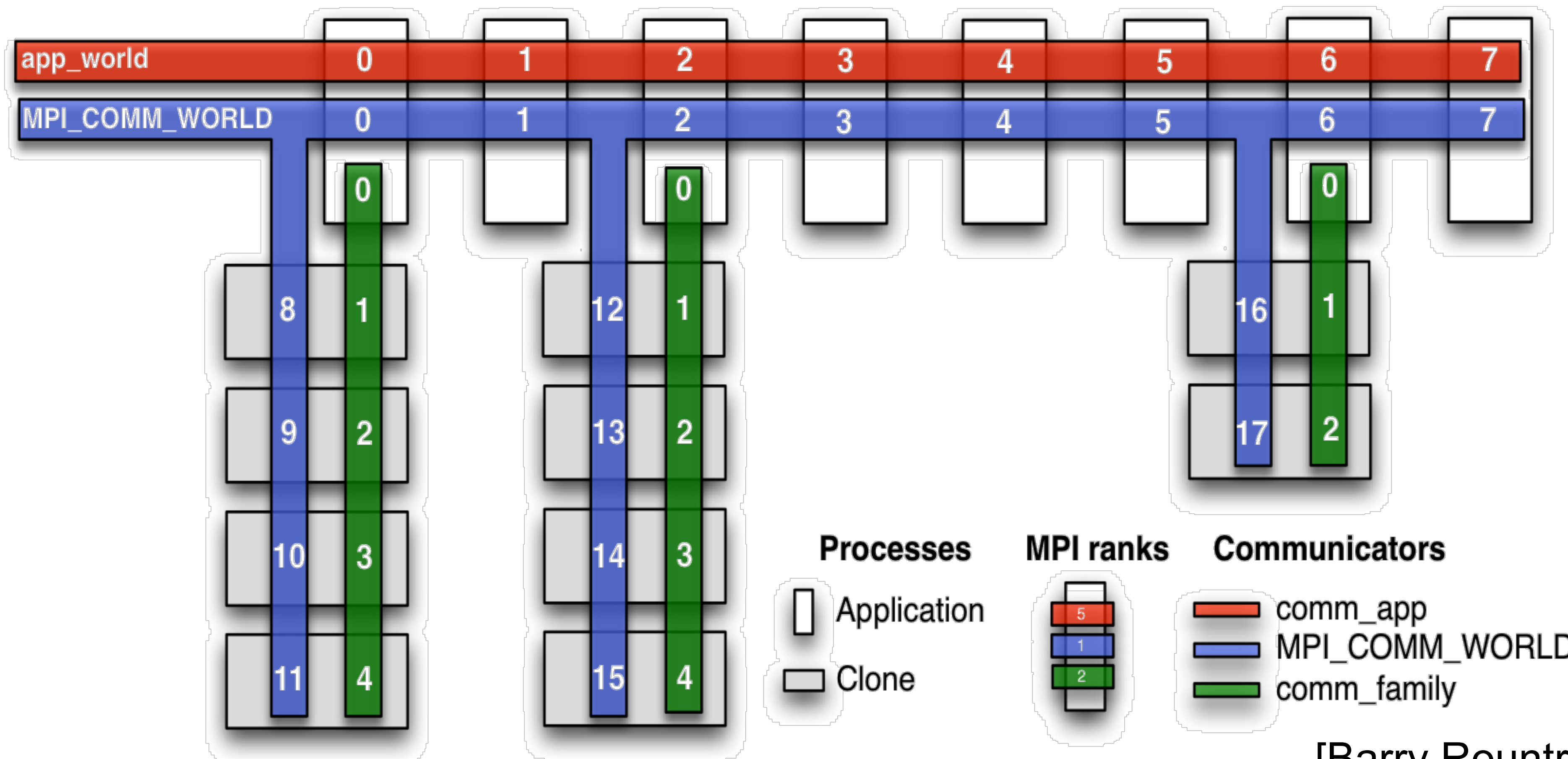
Run applications on reduced MPI_COMM_WORLD
- Reserve nodes for support purposes (e.g., load balancers)

Replace data types to add piggybacking information
- Useful to track critical path information

Replace MPI operations
- Ability to modify/re-implement parts of MPI itself

Extreme example: MPIecho

Transparent cloning of MPI processes

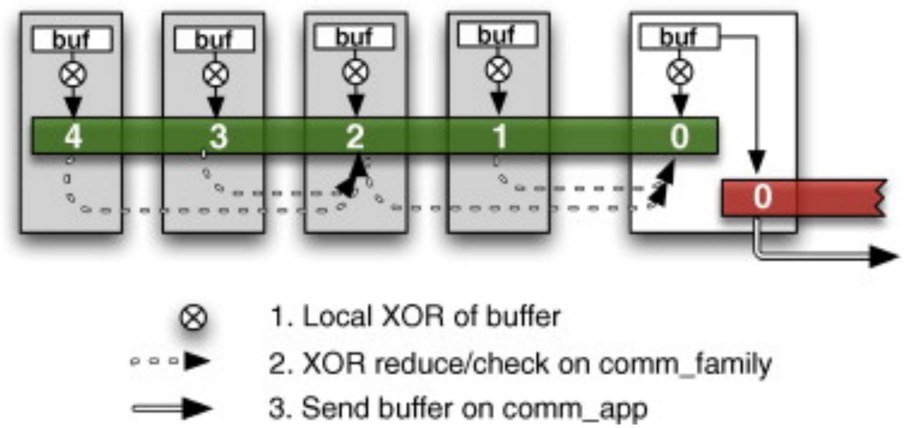


[Barry Rountree]

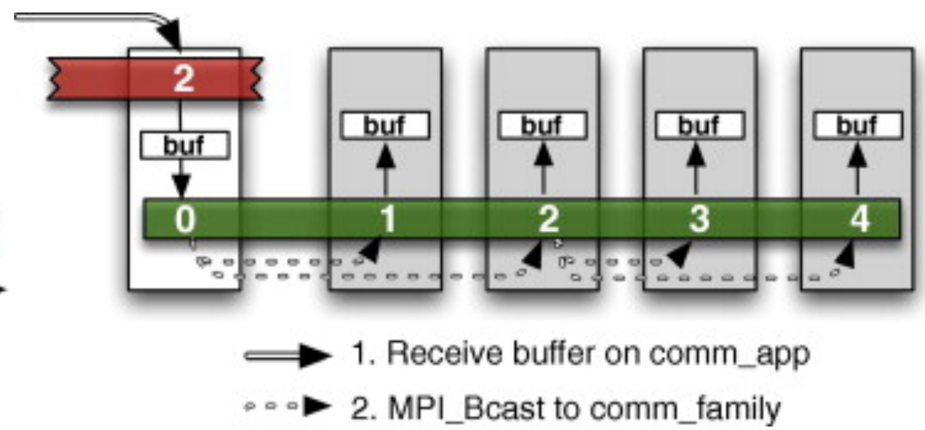
Extreme Example: MPIecho

Implemented through PMPI wrappers

– Send -> No-Op + 1 Send



– Receives -> Bcast

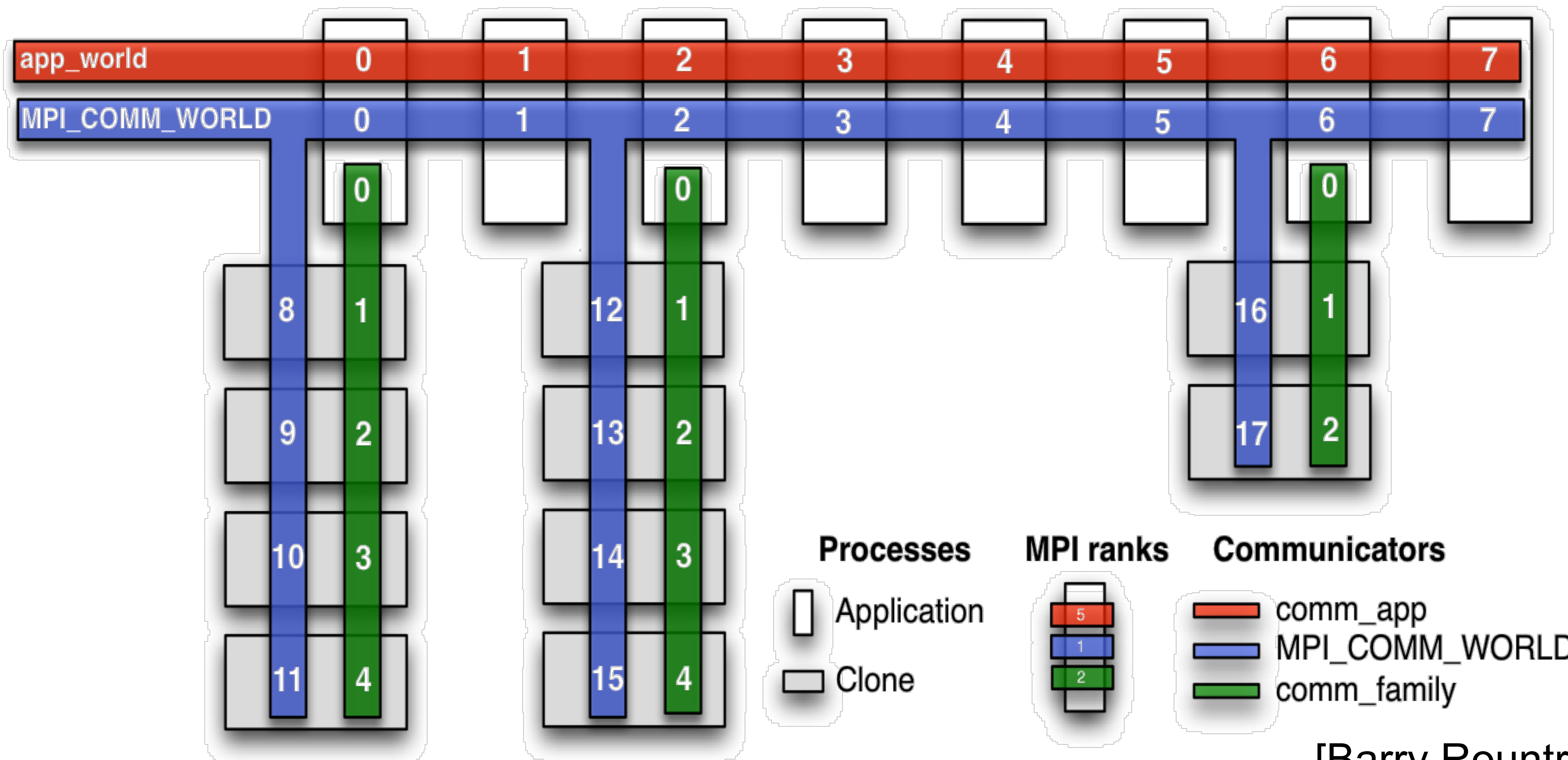


Enables parallelization of tools

– Fault injections
– Memory checking

Extreme example: MPIecho

Transparent cloning of MPI processes



[Barry Rountree]

The State of MPI Tools

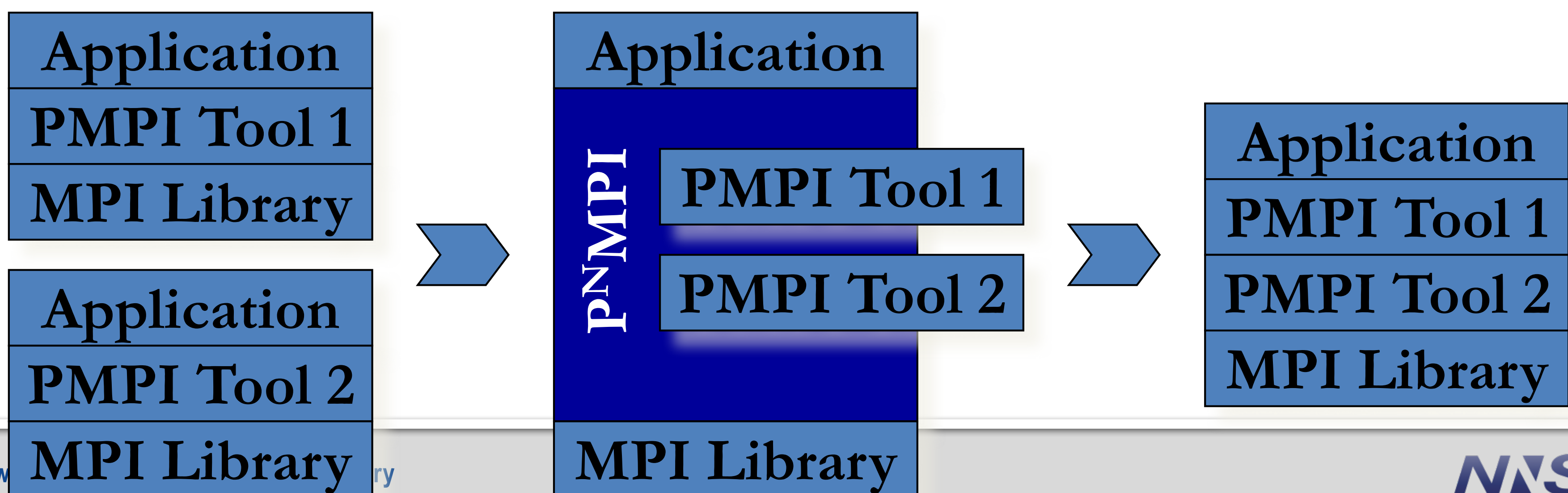
PMPI has led to robust and extensive MPI tool ecosystem

- Wide variety of portable tools
  - Performance, correctness and debugging tools
- Use for application support

PMPI, however, also has problems

- Implementation with weak symbols is often fragile
- Allows only a single tool
- Forces tools to be monolithic

This led to the development of P^nMPI & the QMPI efforts



The Impact on the MPI Standard

The PMPI definition impacts the whole standard
- Even where one doesn't expect it
  - Maximal name length
  - Fortran bindings
  - Threading
- Needs attention to be maintained

PMPI only allows to track application visible information
- Does provide access to internal information
- MPI_T was added to MPI 3.0 to solve this problem
  - After previous failed attempts (like PERUSE)
- MPI can offer internal state for performance and configuration
  - But MPI can decide what to provide and under what name

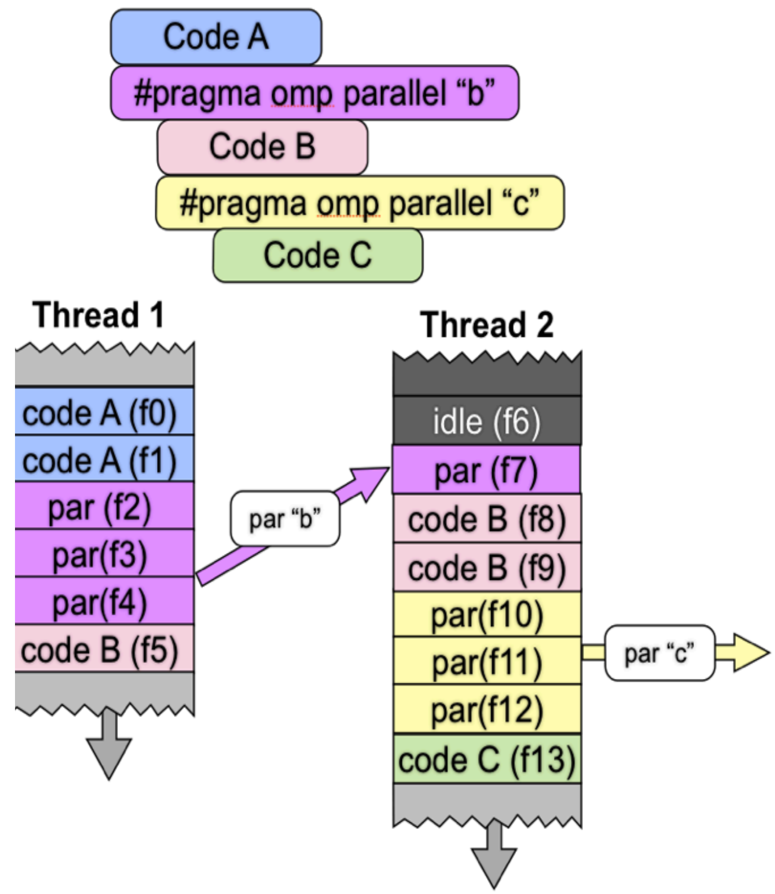
New proposal on MPI_T events in the works
- Callbacks in certain events
- Provides better support for tracing tools
- Again leaves freedom to MPI implementations
- Targeted for MPI 4.0

Other standards are picking up

Other standards are picking up: e.g., OMPT

Goal: enable tools to gather information and associate costs with application source and runtime system

- Hooks for tracing and sampling
- Minimal overhead
- Low implementation complexity
- Mandatory vs. optional parts



Call-stack stitching

- Create user-level view
- Hide runtime impl. details

Status:

- Active API design with outside partners in OpenMP committees
- Included in OpenMP 5.0 draft

But are they overtaking MPI?

The wide-spread use of PMPI is still very unique

- Combined with MPI_T interface(s) provide unprecedented options
- Still exploring the opportunities

But:

MPI does not provide an ABI

- Requires re-compilation of tools for MPI
- Reduces portability and maintainability of tools
- Other standards are specifying all types fully

New MPI interfaces are non committal

- MPI can decide what to offer, if anything
- Names not standardized
- Other standards are allowing more concrete specifications

Summary

MPI provides a strong tool ecosystem

- PMPI is the cornerstone since MPI 1.0
- Developers found creative way to exploit it
- MPI_T interface(s) augment it

Wide range of tools have bee developed

- Performance analysis with Profilers and tracers
- Correctness tools (in combination with debuggers)
- Application support tools

MPI always has been a role model for tool interfaces

- Early adoption in MPI 1.0
- Generally broad support in the MPI Forum
- Strong engagement from tool and MPI developers

But other standards are catching up and MPI could learn something from these efforts as well

- ABIs would make tool maintenance and deployment easier
- More concrete requirements on tool support would be helpful