The Multi-Processor Computing Framework (MPC)

Multi - Processor Computing

MPI Forum
September 24-25, Bordeaux

Jean-Baptiste BESNARD
jbbesnard@paratools.fr

DE LA RECHERCHE À L'INDUSTRIE

cea

ParaTools

Introduction

MPC is a thread-based MPI + OpenMP runtime.

- **Originally developed by CEA**
  - Initiated as a PhD work[1]
  - Started more than ten years ago as part of a complex 3D AMR code[2]
  - Rapidly evolved to become an independent MPI
  - ParaTools collaborates with CEA to develop it
- **Thread-based since the origin**
  - Full MPI_THREAD_MULTIPLE support (no overhead)
  - Provides its own user-level MxN scheduler
- **Built as a framework**

1. Marc Pérache. **Contribution à l'élaboration d'environnements de programmation dédiés au calcul scientifique hautes performances**. Thèse de Doctorat, spécialité informatique, CEA/DAM Île de France, Université de Bordeaux 1, Domaine Universitaire, 351 Cours de la libération, 33405 Talence Cedex, October 2006. Note: 141 pages.

2. Jourdren, H. (2005). **HERA: a hydrodynamic AMR platform for multi-physics simulations**. In Adaptive Mesh Refinement-Theory and Applications (pp. 283-294). Springer Berlin Heidelberg.

ParaTools

MPC Framework (1/4)

MPC is in practice a modular set of components:

- **Message Passing Layer**
  - Inter-thread, TCP, SHM, IB, Portals (in progress)
  - Highly modular and configurable
  - Very compact (easily extensible) and thread-safe
  - Usable as a stand-alone component (library mode)
  - Supports RDMA (to be released), P2P and regular collectives
- **MPI Layer (Fully 1.3 but practically close to 3.0)**
  - Built on top of the Message Passing Layer
  - Full thread-based MPI with MPI_THREAD_MULTIPLE support
  - MPI 3.0 support within one year
- **User-level MxN thread scheduler**
  - Avoids busy-waiting (replaced by context-switching)
  - Full support for POSIX Thread API (e.g. TBB porting)
  - Its policy can be configured
  - Can schedule a (very) large number of threads (oversubscribing)
  - Global view of all supported runtimes (MPI, OpenMP...)

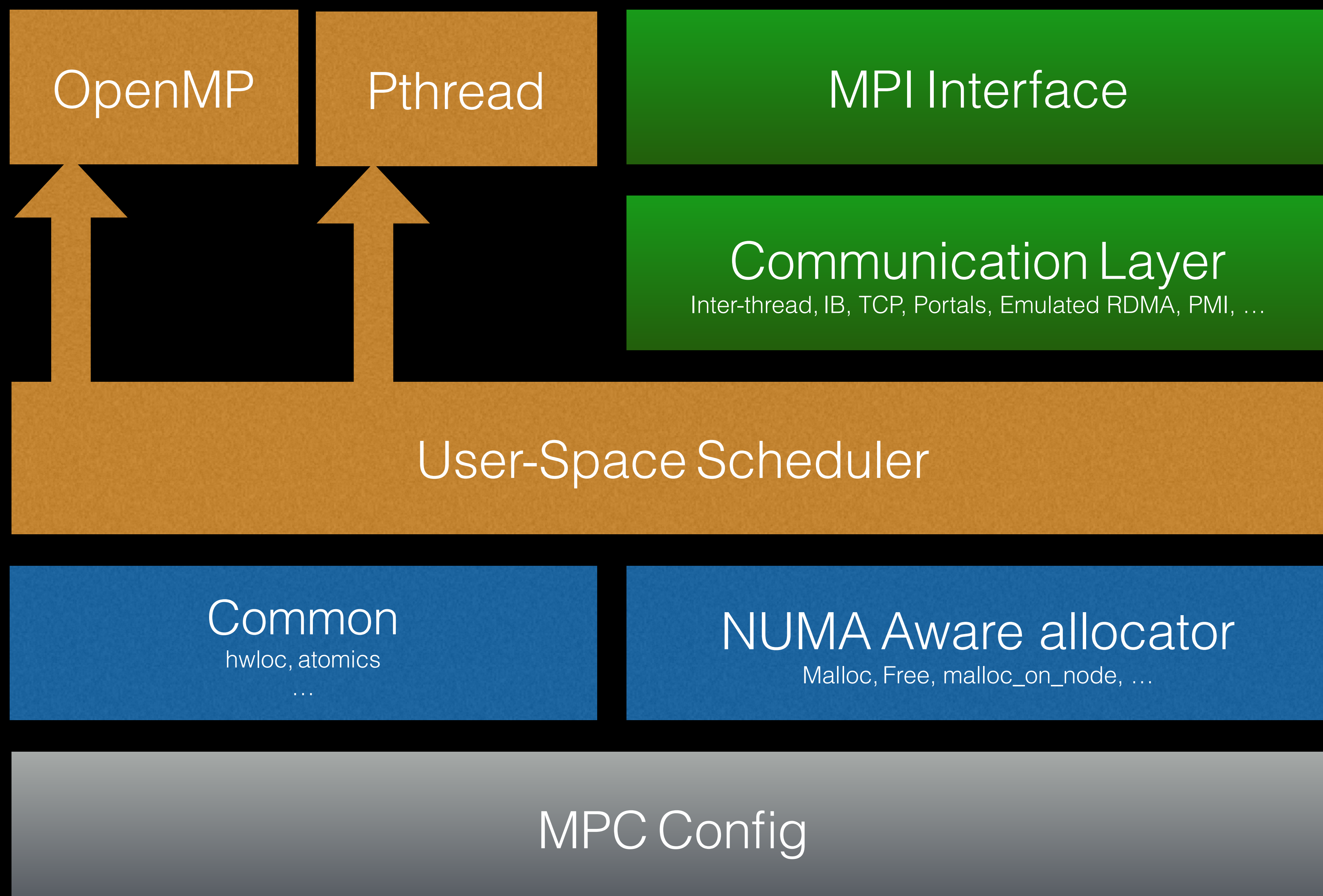
ParaTools

MPC Framework (2/4)

- **OpenMP Runtime**
  - Support for OpenMP 3.1
  - Compatible with GNU and Intel compilers (C, C++ and Fortran)
  - Optimized for hybrid OpenMP + MPI payloads
- **NUMA-Aware Allocator**
  - Per-thread heap to improve data locality and avoid contention
  - Lock-free hierarchical heaps
  - Built to limit system calls (memory recycling)
- **Patched Compiler Chain**
  - Automatic privatization of global variables (GNU and Intel compilers)
  - Hierarchical Local Storage (HLS) for GCC
- **Generic configuration Module**
  - XML-based configuration
  - Easy to extend
  - Designed for maximum modularity

ParaTools

MPC Framework (3/4)



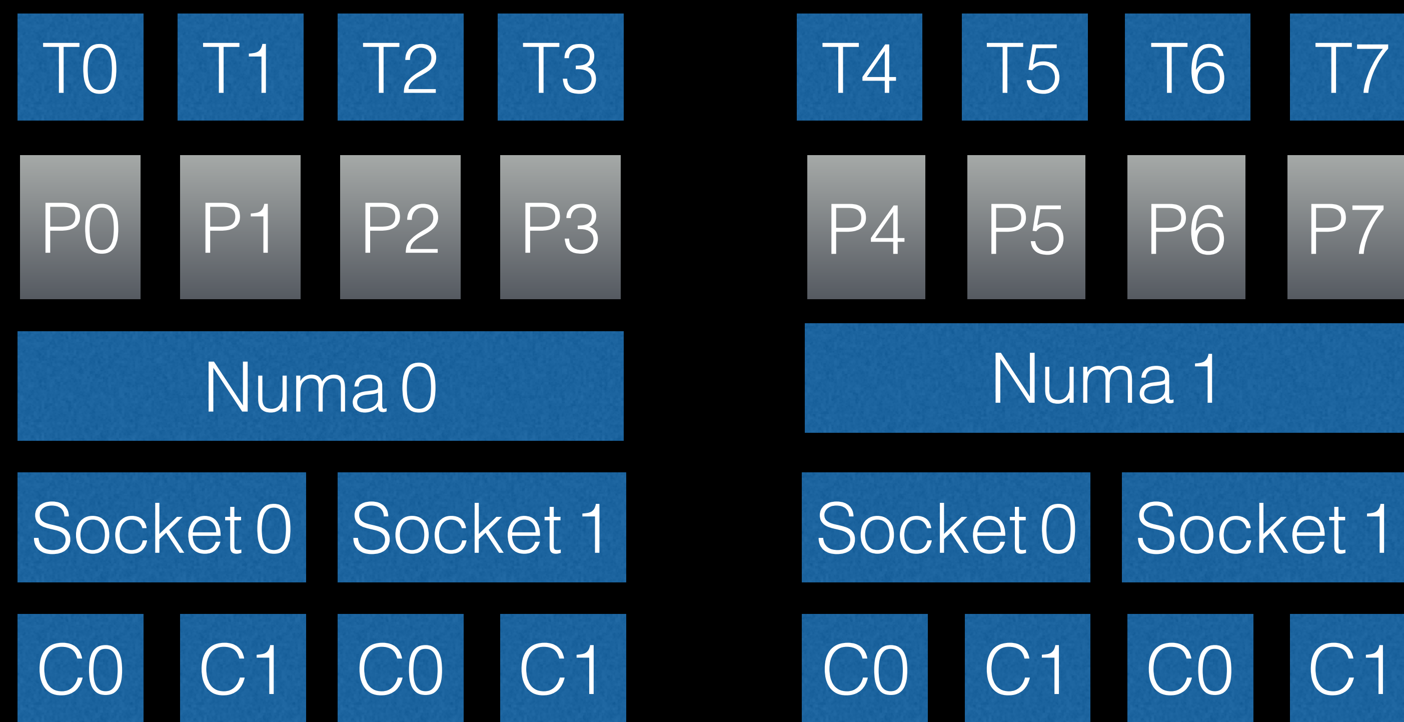
MPC Framework (4/4)

- **Thread-based approach**
  - Reduced launch time (spawns only one process per node)
  - Memory savings (network buffers, runtime internals, process contexts)
  - MPI + X fair mixing over the unified scheduler
  - Fast inter-thread communication (one message => one copy)
- **Lightweight and extensible MPI + X**
  - Relatively small codebase
  - Supports a steadily increasing part of the standards (MPI, OpenMP)
  - Generic network support (complete rail system)
  - Unified XML configuration system
- **Compact RDMA-capable communication library**
  - Multirail support for IB, TCP, Portals*, SHM*
  - Portable RDMA interface*
  - Compile MPC as an auxiliary communication library*
- **MPC is supported by profiling and debugging tools**
  - Patched GDB (user-level thread support)
  - Support for OMPT*, MPIT is a short-term milestone
  - Compatible with Allinea's DDT
  - Compatible with TAU

** to be released in a near future*

ParaTools

Execution Configurations (1/3)



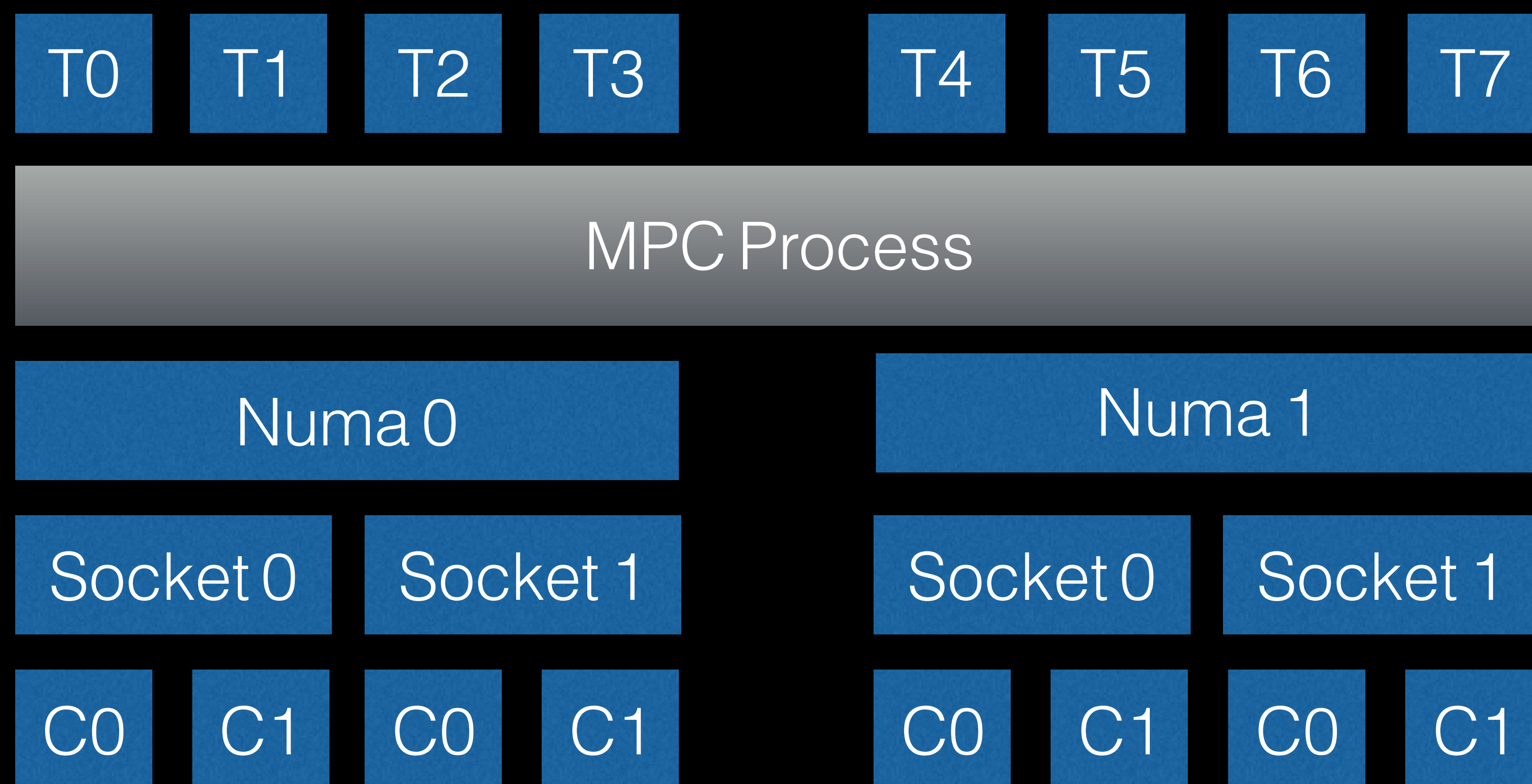
Process-Based MPI

mpcrun -N=1 -p=8 -n=8 ./a.out

Launch is described with four parameters (replacing -np):

- N: Number of nodes
- p: Number of system processes
- n: Number of MPI tasks
- c: Number of cores per process

Execution Configurations (2/3)

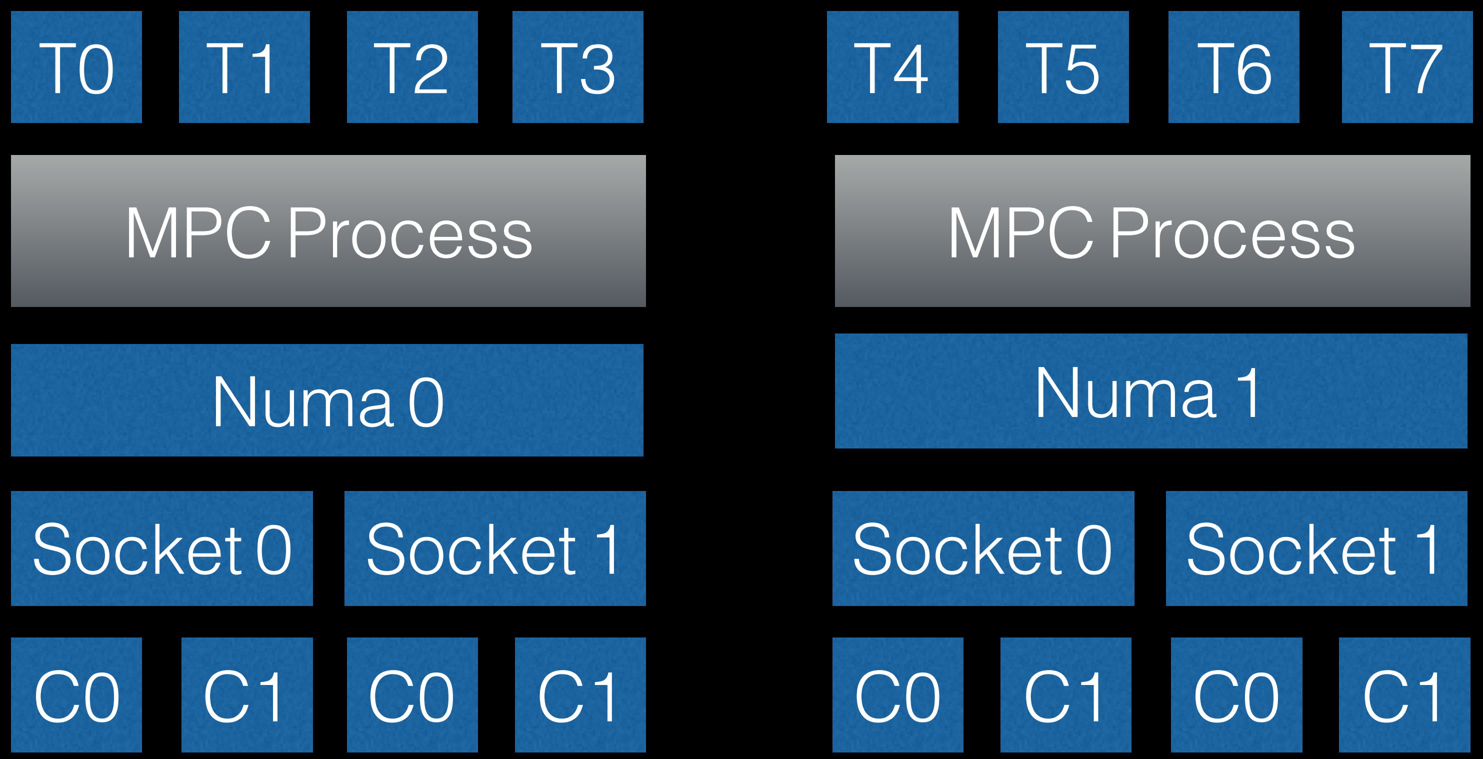


Thread-Based MPI

mpcrun -N=1 -p=1 -n=8 ./a.out

ParaTools

Execution Configurations (3/3)



Mixed Approach

mpcrun -N=1 -p=2 -n=8 -c=4 ./a.out

ParaTools

Privatization Mechanism (1/6)

Motivating example:

```
#include <mpi.h>
#include <stdio.h>

int rank = -1;

int main( int argc, char ** argv )
{
    MPI_Init( &argc, &argv );

    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

     printf("My rank %d", rank );

    MPI_Finalize();

    return 0;
}
```

Privatization Mechanism (2/6)

```
$ mpcrun -net=tcp -p=4 -n=4 ./a.out
Running MPC with HYDRA job manager
[   0/   0/   0/  -1/  -1/   0] WARNING "No RDMA capable rail found (using emulated calls)"
MPC version 3.0.0_devel C/C++ (4 tasks 4 processes 1 cpus (3.59GHz) ethread_mxn) MPC allocator
Rail(0) [TCP_O_IB (ring) (default)]
Initialization time: 0.2s - Memory used: 8MB
My rank 2
My rank 3
My rank 0
My rank 1
```

It's working with n == p ⟶ Process-based MPI

```
$ mpcrun -n=4 ./a.out
Running MPC with HYDRA job manager
MPC version 3.0.0_devel C/C++ (4 tasks 1 processes 4 cpus (3.59GHz) ethread_mxn) MPC allocator
Initialization time: 0.1s - Memory used: 11MB
My rank 1
My rank 1
My rank 1
My rank 1
```

Overwritten global variable ⟶ Thread-based MPI

ParaTools

Privatization Mechanism (3/6)

Global variables should be duplicated (or **privatized**)

One idea is to rely on TLS:

```
__thread int rank = -1;
```

```
$ mpcrun -net=tcp -p=1 -n=8 ./a.out
Running MPC with HYDRA job manager
MPC version 3.0.0_devel C/C++ (8 tasks 1 processes 4 cpus (3.59GHz) ethread_mxn) MPC allocator
Initialization time: 0.1s - Memory used: 9MB
My rank 1
My rank 1
My rank 7
My rank 7
My rank 5
My rank 5
My rank 3
My rank 3
```

TLS solution not working in oversubscribing context (more MPI tasks than cores)

Conclusion: need to design a mechanism able to duplicate global variables and to deal with the context switching logic

ParaTools

Privatization Mechanism (4/6)

MPC solution: a compile-time approach for automatic global variable privatization

- Simplifies porting of existing codes and libraries
- Allows MPI applications to be run in shared-memory

The loader was also patched to optimize context-switching using the GS register on x64 targets.

```
$ mpc_cc t.c
(Front-end C) Automatic privatization to MPC task (variable rank in file t.c line 6)
```

- Promotes global variables to TLS level
- Adds the context-swiching logic to the generated code
- Implementation in GNU compilers (C/C++ and Fortran)
  - Uses the -fmpc-privatize flag (enabled by default)

ParaTools

Privatization Mechanism (5/6)

CEA collaborated with Intel to have this same option implemented in the Intel C/C++ and Fortran compilers.

Excerpt from the Intel ICC compiler manual (starting with 15.0):

-fmpc-privatize (L*X only)

-fno-mpc-privatize (L*X only)

Enables or disables privatization of all static data for the MultiProcessor Computing environment (MPC) unified parallel runtime.

Architecture Restrictions: Only available on Intel(R) 64 architecture

Arguments:

None

Default:

-fno-mpc-privatize
The privatization of all static data for the MPC unified parallel runtime is disabled.

Description:

This option enables or disables privatization of all static data for the MultiProcessor Computing environment (MPC) unified parallel runtime.

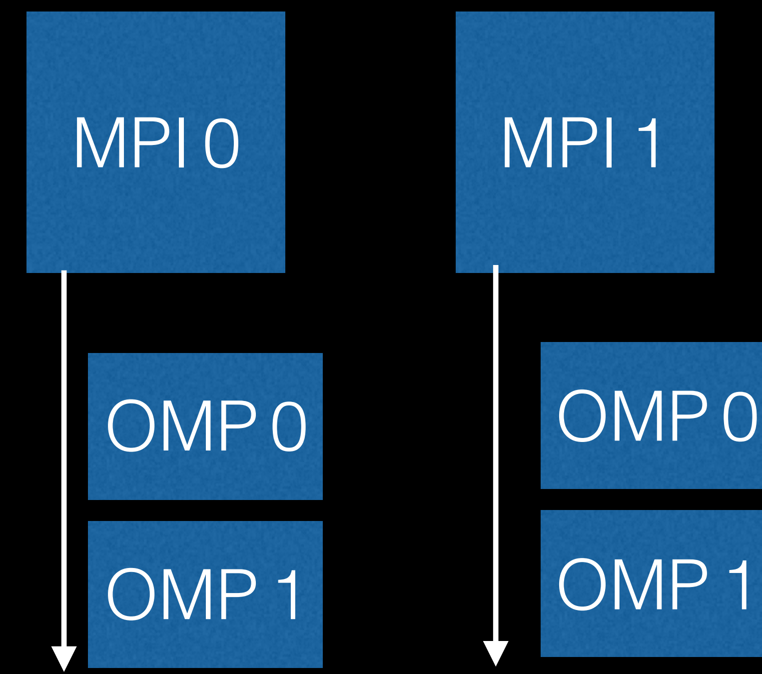
Option -fmpc-privatize causes calls to extended thread-local-storage (TLS) resolution, run-time routines that are not supported on standard Linux* OS distributions.

This option requires installation of another product. For more information, see Feature Requirements

ParaTools

Privatization in Hybrid Context (6/6)

```c
#include <mpi.h>

int rank = -1;

int omprank = -1;
#pragma omp threadprivate(omprank)

int main( int argc, char ** argv )
{
   MPI_Init( &argc, &argv );

   MPI_Comm_rank( MPI_COMM_WORLD, &rank );

   #pragma omp parallel
   {
       omprank = omp_get_thread_num();
       printf("MPI %d OMP %d\n", rank, omprank );
   }

   MPI_Finalize();

   return 0;
}
```



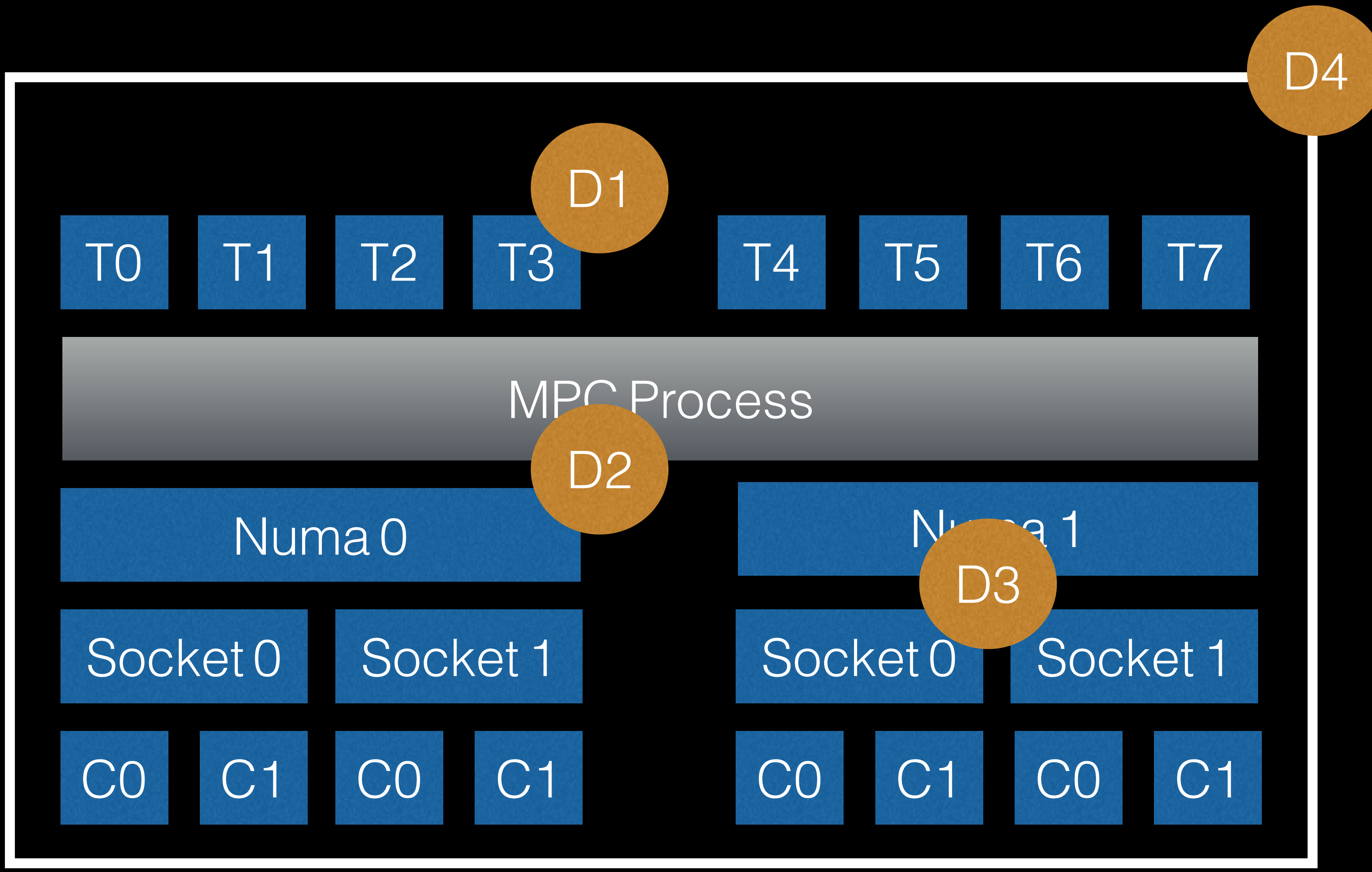

```
MPI 1 OMP 0
MPI 1 OMP 1
MPI 0 OMP 0
MPI 0 OMP 1
```

2 Processes on 4 cores

Hybrid MPI+OpenMP mode requires a specific handling of thread-private OpenMP variables provided by the Extended TLS mechanism

Carribault, P., Pérache, M., & Jourdren, H. (2011). **Thread-local storage extension to support thread-based MPI/openMP applications**. In *OpenMP in the Petascale Era* (pp. 80-93). Springer Berlin Heidelberg.

Hierarchical Local Storage (HLS) (1/2)



Extends the TLS mechanism to store "global" arrays (e.g., huge material data tables) at a given level of the memory hierarchy (Node, NUMA, Cache, Core).

Hierarchical Local Storage (HLS) (2/2)

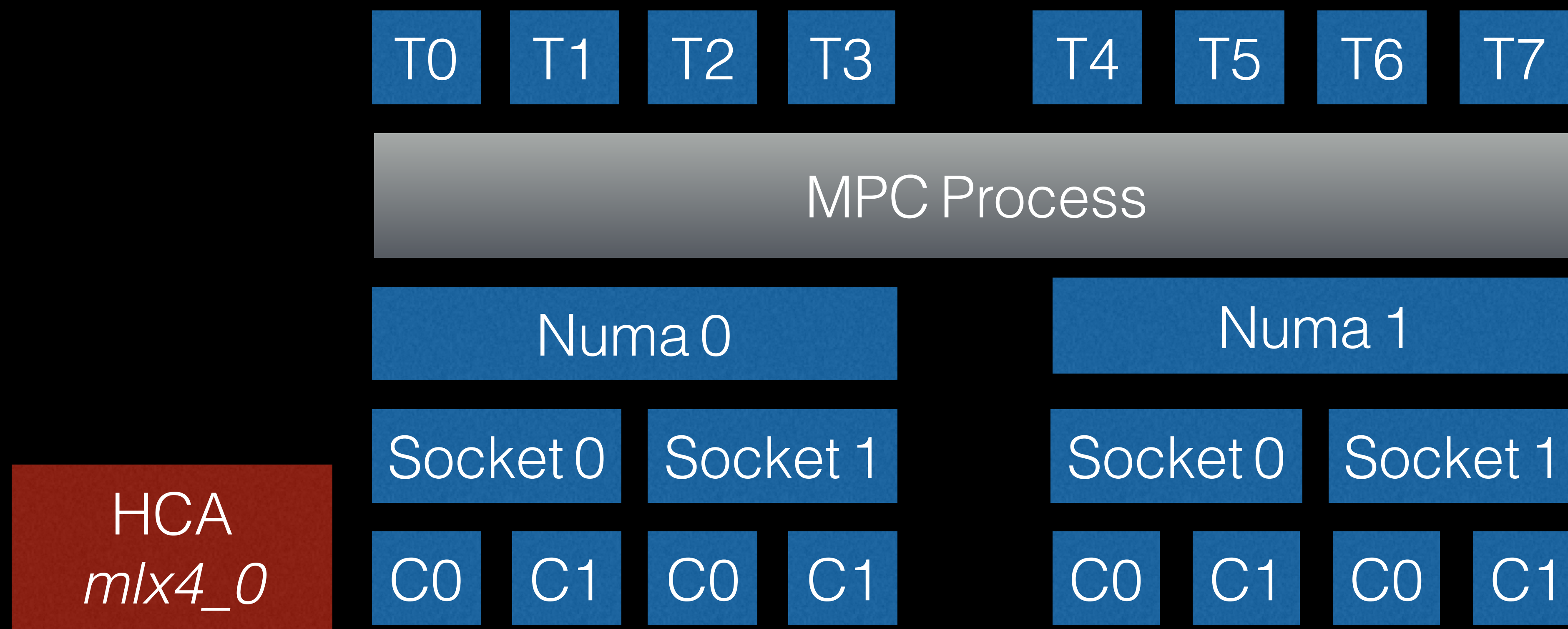
```
int a,b ;
#pragma hls node(a)
#pragma hls numa(b)

void f()
{
 ...
    #pragma hls single(a)
    {
        // executed by one instruction flow per node
        a=4;
    }

    #pragma hls single(b)
    {
        // executed by one instruction flow per NUMA node
        b = 2 ;
    }
}
```

Compiler-enabled (GCC) data sharing between MPI tasks with OpenMP-like directives.

Tchiboukdjian, M., Carribault, P., & Pérache, M. (2012, May). **Hierarchical local storage: Exploiting flexible user-data sharing between MPI tasks**. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International* (pp. 366-377). IEEE.

ParaTools

Collaborative Polling



The HCA being a shared ressource, waiting tasks can help others to progress by polling the network or copying buffers (also true for inter-thread comm).

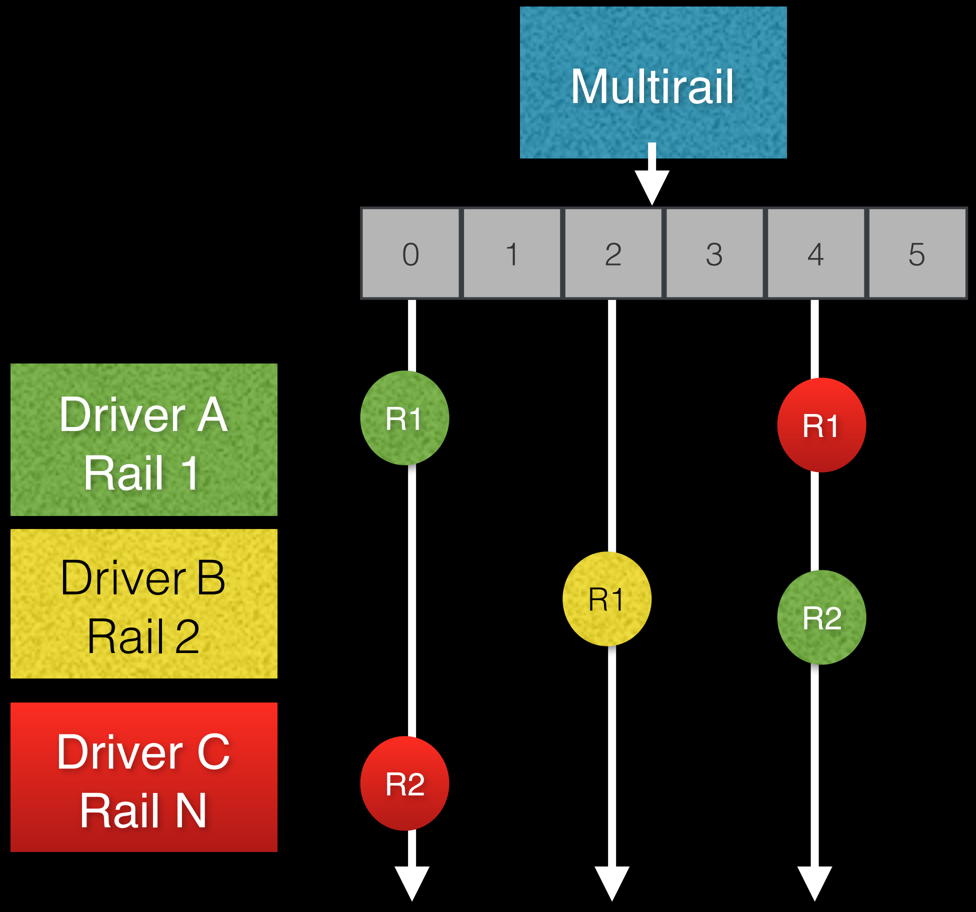
Didelot, S., Carribault, P., Pérache, M., & Jalby, W. (2014). **Improving MPI communication overlap with collaborative polling.** *Computing*, *96*(4), 263-278.

ParaTools

Network Rail Support and Configuration (1/4)

```
<config>
   <name>tcp_config_mpi</name>
   <driver>
           <tcp/>
   </driver>
</config>

<rail>
    <name>tcp_mpi</name>
    <priority>1</priority>
    <topology>ring</topology>
    <config>tcp_config_mpi</config>
</rail>

<rail>
    <name>tcp_large</name>
    <priority>10</priority>
    <topology>none</topology>
    <config>tcp_config_mpi</config>
    <gates>
        <gate>
            <minsize>
                <value>32KB</value>
            </minsize>
        </gate>
    </gates>
</rail>

<cli_option>
    <name>multirail_tcp</name>
    <rails>
        <rail>tcp_large</rail>
        <rail>tcp_mpi</rail>
    </rails>
</cli_option>
```

Endpoint-based Rail-election through priorities and gates Multi-Driver, Multi-Device



Launch command:

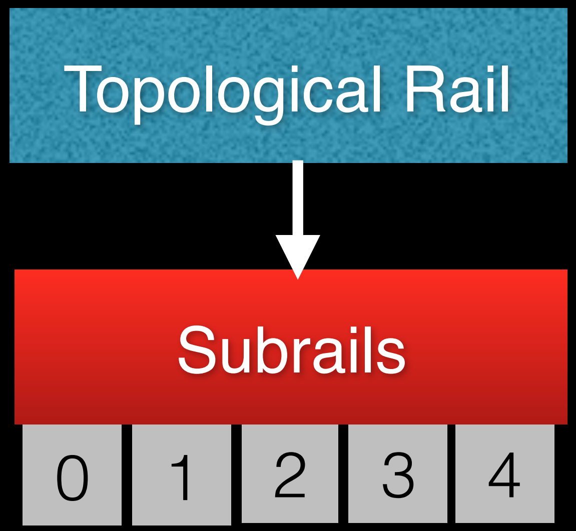
```
mpcrun -net=multirail_tcp -p=8 -n=16 ./a.out
```

ParaTools

Network Rail Support and Configuration (2/4)

Topology-based rail selection between tasks using rail stacking and a regular expression to match devices.

```
<rail>
   <name>ib_mpi_all_devices<name>
   <device>!mlx4_*</device>
   <config>ib_config_mpi</config>
   ...
<rail>
```



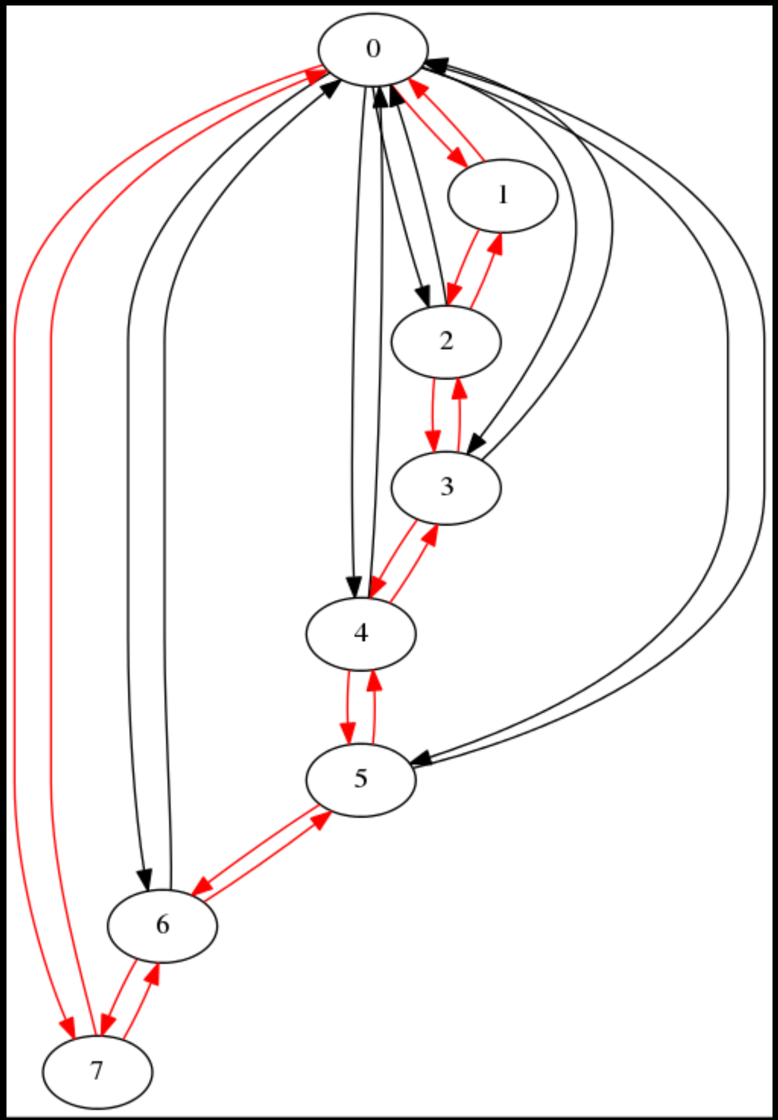

```
Rail(0) [Topological RAIL (ring) (!mlx4_[0,3])]
    Sub-Rail(1) [IB-MT (v2.0) MPI      4/4:mlx4_0 - 4x QDR (32 Gb/s) - 0] (ring) (mlx4_0)]
    Sub-Rail(2) [IB-MT (v2.0) MPI      1/4:mlx4_3 - 4x QDR (32 Gb/s)

Rail(0) [Topological RAIL (ring) (!mlx4_[0-1])]
    Sub-Rail(1) [IB-MT (v2.0) MPI      4/4:mlx4_0 - 4x QDR (32 Gb/s) - 0] (ring) (mlx4_0)]
    Sub-Rail(2) [IB-MT (v2.0) MPI      3/4:mlx4_1 - 4x QDR (32 Gb/s)

Rail(0) [Topological RAIL (ring) (!mlx4_*)]
    Sub-Rail(1) [IB-MT (v2.0) MPI      4/4:mlx4_0 - 4x QDR (32 Gb/s) - 0] (ring) (mlx4_0)]
    Sub-Rail(2) [IB-MT (v2.0) MPI      3/4:mlx4_1 - 4x QDR (32 Gb/s) - 0] (ring) (mlx4_1)]
    Sub-Rail(3) [IB-MT (v2.0) MPI      2/4:mlx4_2 - 4x QDR (32 Gb/s) - 0] (ring) (mlx4_2)]
    Sub-Rail(4) [IB-MT (v2.0) MPI      1/4:mlx4_3 - 4x QDR (32 Gb/s) - 0] (ring) (mlx4_3)]
```

Bull BCS nodes (32 x 4 = 128 cores)

Network Rail Support and Configuration (3/4)

The bootstrapping process relies on topological promotion:

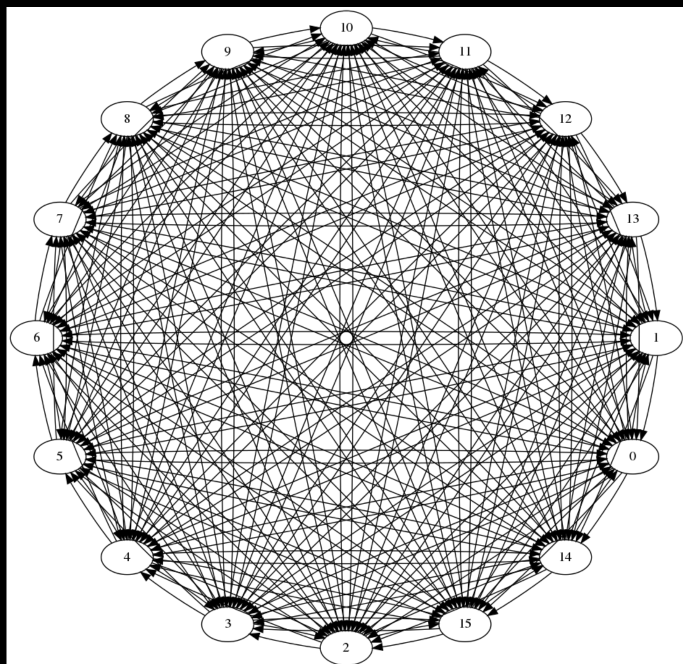
- First a ring is connected using the PMI (can be on a single rail)
- Then a topology is built using this initial connectivity
- On-demand connections are routed (distance) on this topology



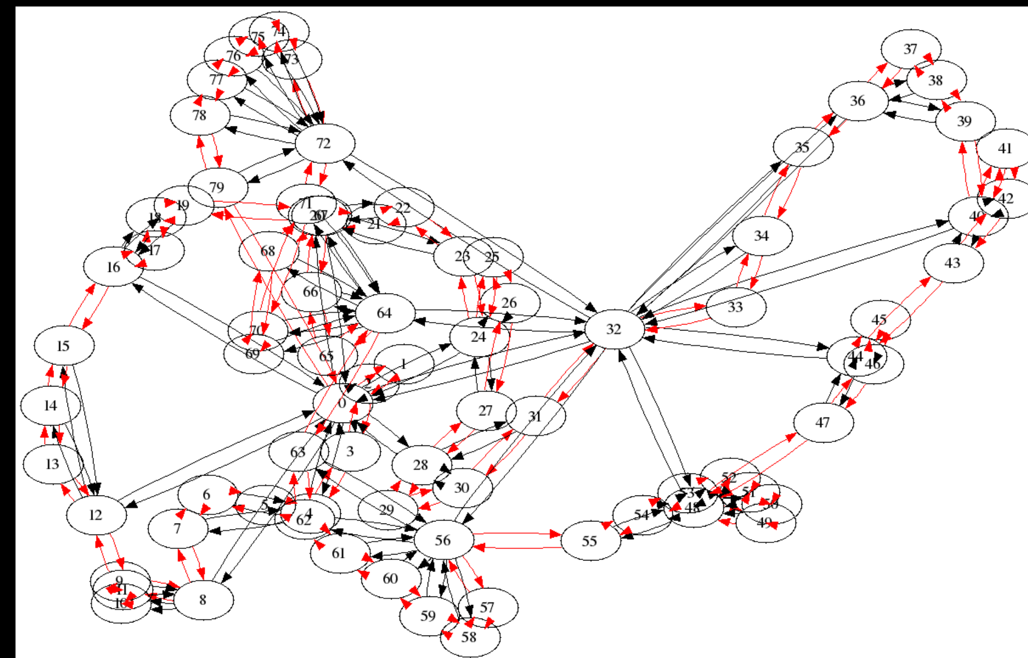
Connectivity after a barrier (8-tree) static routes are in RED, on-demand routes are in black.

The RED links ensure that any connection can be initiated thanks to routing.

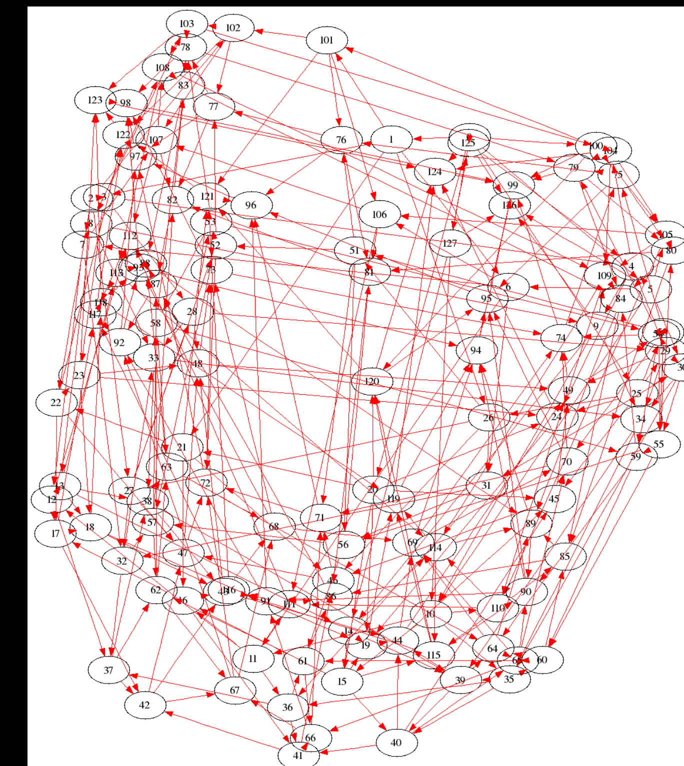
Network Rail Support and Configuration (4/4)



Fully Connected



Ring + Init Barrier



Torus

Promoting the routing topology reduces the network diameter, speeding up **control messages** (internal RPCs). Il also opens the way for "out-of-band" messages (loosely synchronized, one-sided).

- Used internally for emulated RMA, MPI Windows, On-Demand
- User-defined RPCs (remote control, in-situ visualization) — "Server Mode"
- IDEA: out-of-band messages (load-balancing, profiling)
- IDEA: contention management (route instead of connecting N-1 avoidance) on-demand connection hiding (for small payloads)

MPC Development Tracks (1/4)

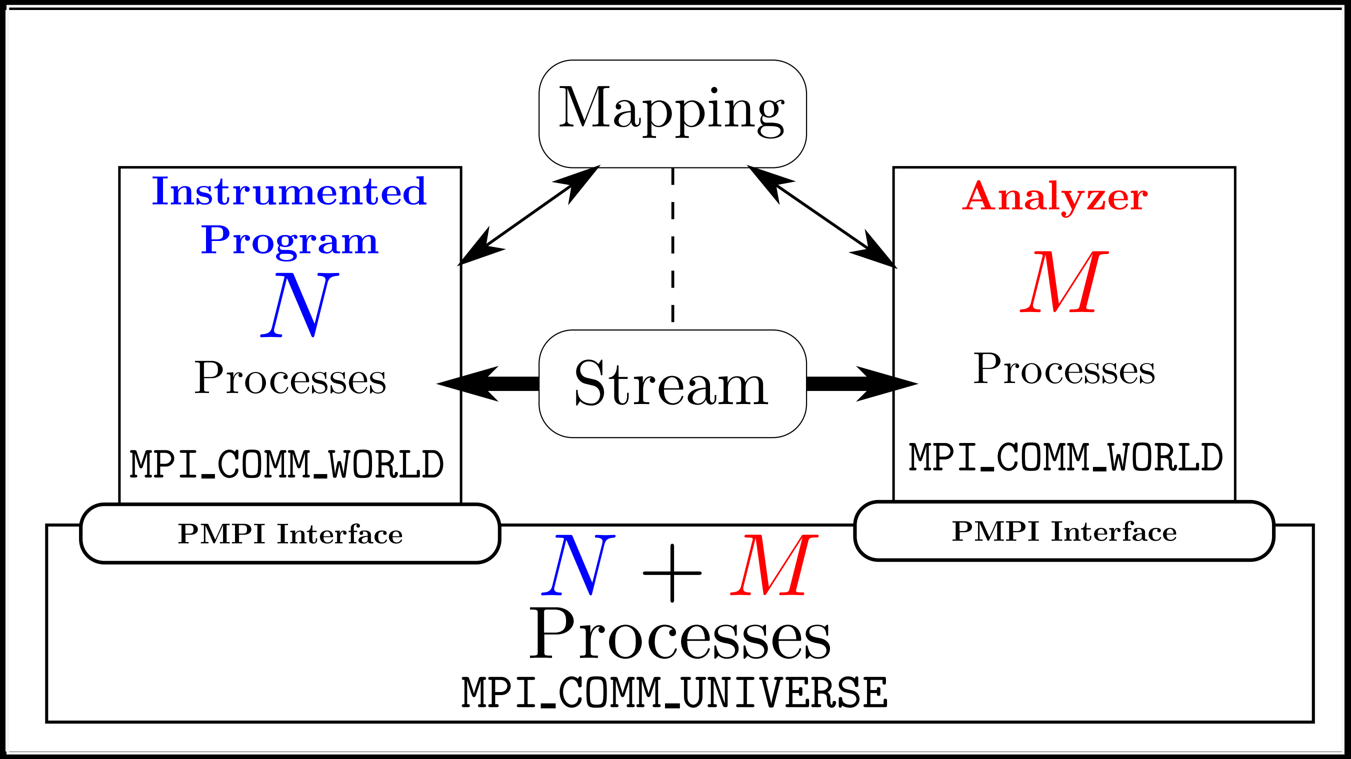
Some features we are working on:

- **Better support of Process-Based MPI:**
  - Ease the transition to thread-based / MPI + X
  - Provide an *mpirun* interface
  - Implement a state-of-the-art SHM (Linux Cross Memory Attach)
- **Enhance the MPI standard coverage:**
  - We recently merged ROMIO for MPI-IO and NBC
  - Data-types are almost 3.0
  - Low-level RDMAs are implemented, one sided are on the way
  - Dynamic processes are planned for next year
- **Enhance reliability and portability**

ParaTools

MPC Development Tracks (2/4)

"Constellation of Services"



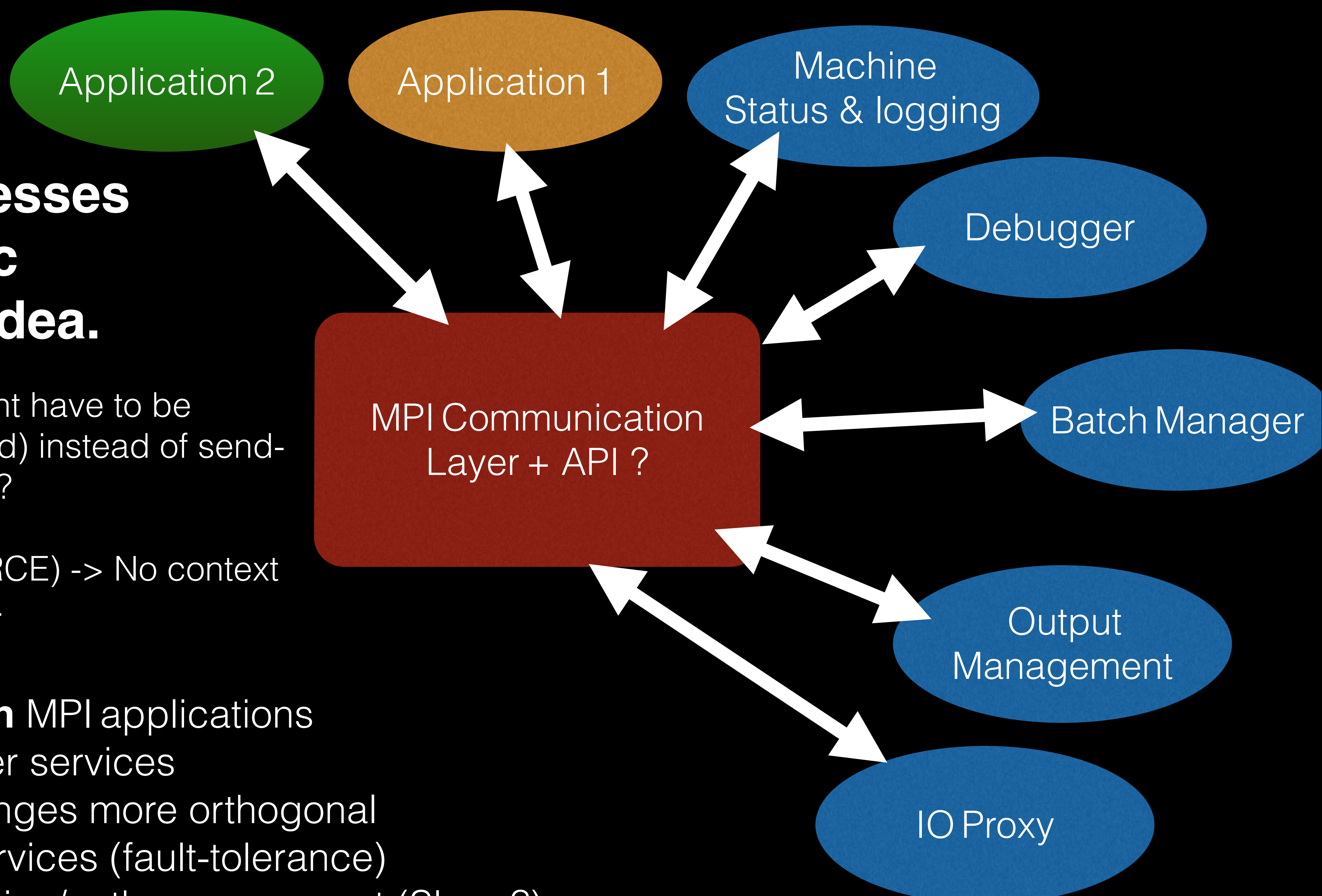
Scheme developed during my PhD thesis to manage profiling data using the P^{N}MPI[1] virtualization idea: we would like to apply this approach more widely.

1. Schulz, M., & De Supinski, B. R. (2007, November). **PN MPI tools: A whole lot greater than the sum of their parts**. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing* (p. 30). ACM.

ParaTools

MPC Development Tracks (3/4)

"Constellation of Services"



Dynamic processes provide a basic block for this idea.

Data management might have to be enhanced (query-based) instead of send-recv — "server" mode ?

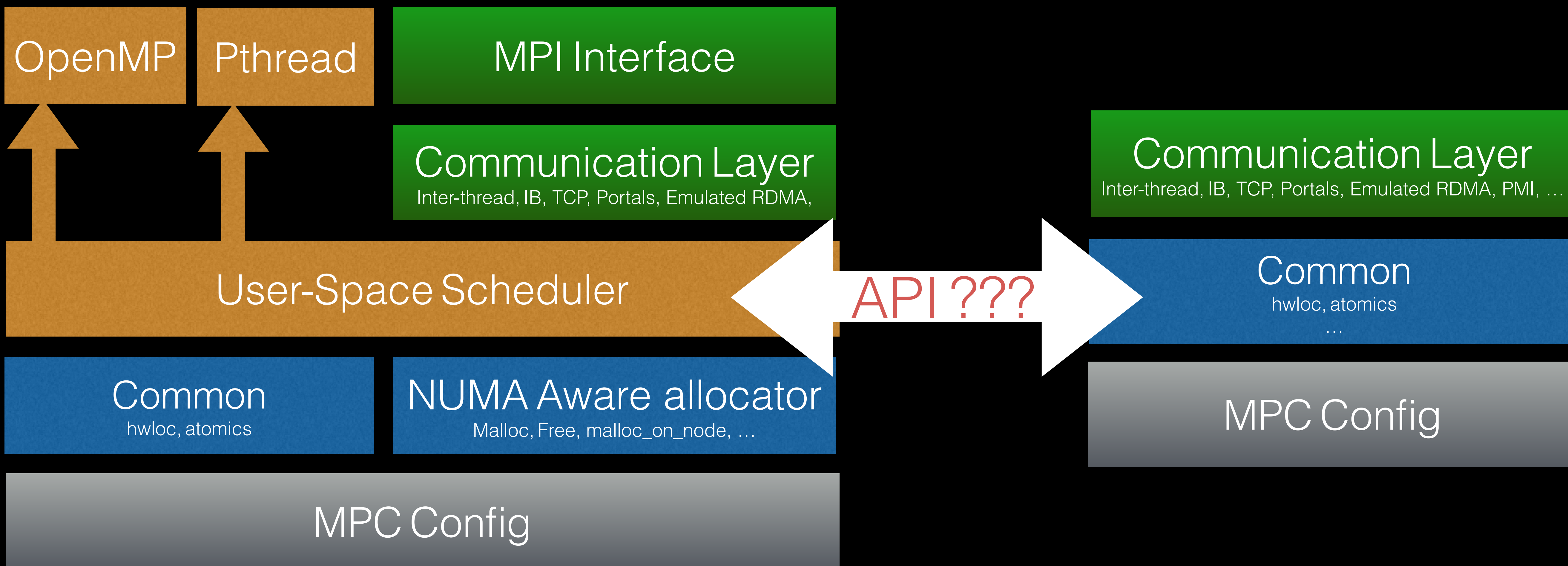
(ANY_TAG, ANY_SOURCE) -> No context
Why not RPCs, Streams, ….

- Attach and **detach** MPI applications to provide per-user services
- Make data-exchanges more orthogonal
- Support failing services (fault-tolerance)
- Provide a connection/auth management (Slurm?)
- Stream data between parallel instances

ParaTools

MPC Development Tracks (4/4)

"Constellation of Services"



Complete MPC

« Lib-mode » MPC

We were able to link the "lib-mode" MPC inside OpenMPI to provide an auxiliary communication library. Our purpose is to interface MPI instances with an existing service infrastructure taking advantage of RMDAs — trying to extract a standard interface.

ParaTools

MPC Validation Suite (1/3)

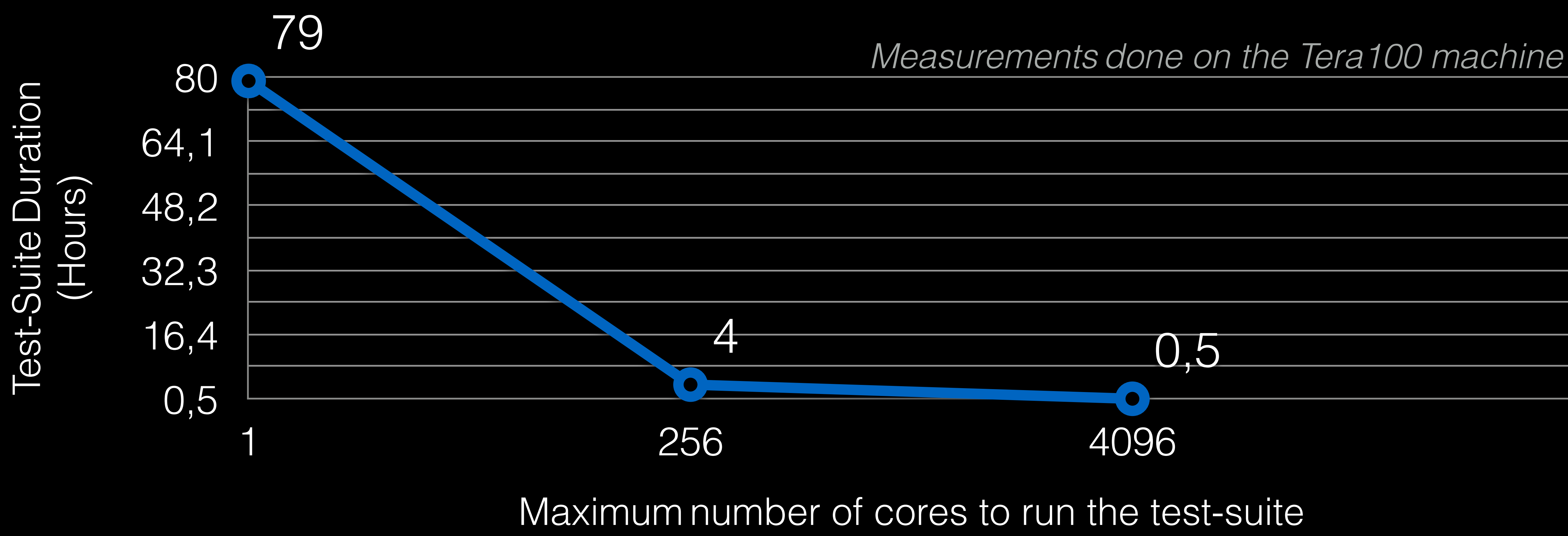
To reach its production milestone a test-suite engine has been developed for MPC then forked as **JCHRONOSS** to be used independently.

Released under CECILL-C (fully LGPL compatible)
http://jchronoss.hpcframework.com

Scheduling of **parallel tests in parallel** on a supercomputer

- Uses the batch manager (slurm, PBS…) with a surface-based scheduling policy
- Fault-tolerance support (resubmission)
- Support for dependencies (e.g., compilation -> run)
- Can be launched by individual developers (locally)
- Provides a lightweight html GUI for in-situ validation
- Stores the output of failed tests

MPC Validation Suite (2/3)



Reduces the validation time for the 40,307 tests of the MPC test-suite from 3.2 days to less than an hour (45 minutes), taking advantage of supercomputer parallel resources to validate our parallel runtime in production environment.

MPC Validation Suite (3/3)

```
 STATUS      |  ID  /  SZ  | LEFT |    TIME    |    NAME    |
NOT RUNNABLE :    -/-     (------)    0.00    test_08
NOT RUNNABLE :    -/-     (------)    0.00    test_17
PASSED       :    1/2     (    18)    0.00    compile_01
PASSED       :    2/2     (    18)    0.00    compile_04
PASSED       :    1/1     (    16)    0.00    test_13
PASSED       :    1/1     (    15)    0.01    test_01
PASSED       :    1/1     (    11)    0.00    test_06
FAILED       :    1/1     (    13)    0.00    compile_02
PASSED       :    1/1     (    12)    0.00    test_03
NOT RUNNABLE :    -/-     (------)    0.00    test_05
PASSED       :    1/1     (     7)    0.00    test_12
PASSED       :    1/1     (     8)    0.00    test_09
PASSED       :    1/1     (    10)    0.00    test_07
PASSED       :    1/1     (     6)    0.01    test_14
PASSED       :    1/1     (     4)    0.00    test_16
PASSED       :    1/1     (     5)    0.00    test_15
PASSED       :    1/1     (    14)    5.01    test_01
PASSED       :    1/1     (     3)    0.00    test_02
NOT RUNNABLE :    -/-     (------)    0.00    test_10
NOT RUNNABLE :    -/-     (------)    0.00    test_11
--------------------- RUN SUMMARY ---------------------
| * Executed jobs  : 20
|    --> Fails     : 1
|    --> Errors    : 0
|    --> Skips     : 5
|    --> Successes : 14
| * Elapsed time   : 5 second(s)
| * Slave launchs  : 14
```

Console Output

test_cas_ethread_mxn_tcp_proc_nb_1_2	success	4.11
test_cas_long_pthread_ib_proc_nb_1_2	**failed**	**2.26**

Message :

```
mpcrun --clean --autokill=500 -N=1 -n=2 --share-node -p=1 -net=ib -m=pthread -c=8 -l=srun TESTSUITE_PATH/build/test_suite/./MPI
/RDMA/atomics/executables//cas_long
```

Failure :

```
Autokill in 500s
MPC version 3.0.0_devel C/C++ (2 tasks 1 processes 8 cpus (2.80GHz) pthread) STD allocator  Debug MODE none
Initialization time: 0.1s - Memory used: 4MB
Error wrong CAS ret 18090
srun: error: inti1018: task 0: Aborted
```

test_cas_long_ethread_mxn_ib_proc_nb_1_2	success	3.86

Summary

Status	Nombre	Rapport	
success	36988		96 %
failed	1323		3 %
error	0		0 %
skipped	20		0 %
total	*38331*	*failed*	*100 %*

Test groups

Directory	Errors	Failures	Skipped	Success ▴	Total	% failed	Status
Threads-interfaces	0	354	16	2489	2859	12	Failed
MPI-ANL_error-grp_ctxt_comm	0	0	0	1826	1826	0	Success
MPI-MPICH	0	520	0	1613	2133	24	Failed
MPI-ANL_error-datatype	0	0	0	1504	1504	0	Success
MPI-ANL_functional-datatype	0	0	0	1475	1475	0	Success
MPI-ANL_error-collective	0	0	0	1453	1453	0	Success
MPI-ANL_error-topo	0	0	0	1325	1325	0	Success
MPI-Intel_ANL-types	0	0	0	1150	1150	0	Success
fortran-MPI_Intel_ANL-types	0	13	0	1112	1125	1	Failed
MPI-Intel_ANL-sync	0	7	0	1055	1062	1	Failed
MPI-ANL_error-persist_request	0	0	0	964	964	0	Success
fortran-MPI_Intel_ANL-sync	0	0	0	959	959	0	Success
MPI-ANL_error-blocking	0	1	0	937	938	0	Failed
MPI-ANL_functional-grp_ctxt_comm	0	32	0	918	950	3	Failed
MPI-ANL_error-nonblocking	0	24	0	863	887	3	Failed
MPI-MPICH_TEST_SUITE-pt2pt	0	6	0	845	851	1	Failed
MPI-ANL_functional-nonblocking	0	2	0	782	784	0	Failed
MPI-Intel_ANL-collectives	0	0	0	750	750	0	Success
fortran-MPI_Intel_ANL-communicators	0	11	0	714	725	2	Failed
MPI-Intel_ANL-async	0	4	0	703	707	1	Failed
MPI-ANL_functional-persist_request	0	1	0	680	681	0	Failed
fortran-MPI_Intel_ANL-collectives	0	0	0	678	678	0	Success
fortran-MPI_Intel_ANL-async	0	5	0	645	650	1	Failed
MPI-Intel_ANL-communicators	0	32	0	645	677	5	Failed
MPI-ANL_error-sendrecv	0	0	0	604	604	0	Success
MPI-ANL_functional-blocking	0	0	0	531	531	0	Success
MPI-Intel_ANL-others	0	25	0	529	554	5	Failed

Static HTML Interface

Test-suite result field visualization

ParaTools

Conclusion

We introduced the
Multi-Processor Computing framework

- **Key features**
  - MPI & OpenMP implementations (based on unified thread scheduler)
  - Automatic privatization (GNU and Intel compilers)
  - HLS directives (GNU compilers)
  - Collaborative Polling
  - Generic rail configuration and topology management
  - Control messages
- **Development tracks for MPC**
  - Process-based mode (for transition)
  - Constellation of services (low-level back-end)
- **Test-suite engine**
  - Validating on a daily basis a complete test-suite (w/ 30k MPI tests)
  - Taking advantage of a parallel machine

ParaTools

Try MPC

Fully LGPL compatible (French CECILL-C licence)

Is an interesting prototyping/research platform

http://paratools.fr/mpc

ParaTools

The Multi-Processor Computing Framework (MPC)

MPI Forum
September 24-25, Bordeaux

Jean-Baptiste BESNARD
jbbesnard@paratools.fr