MUST Correctness Checking for YML and XMP Programs

Second French-Japanese-German Workshop on Programming and Computing for Exascale and Beyond

Japan Oct. 30, 2018

Presenter: Matthias S. Müller

Project partners:

RWTH Aachen University, Germany
University of Tsukuba, Japan
Maison de la Simulation, France

IT Center | RWTH AACHEN UNIVERSITY

Consortium

- MYX builds on successful preliminary work and collaboration:
  - FP3C: French-Japanese collaboration on YML and XMP for over 10 years
  - JST-CREST: Japanese Exascale research program supporting XMP
  - MUST: scalable correctness checking tool for MPI (and OpenMP)

Partner from Germany (project coordinator)

- RWTH Aachen University
  IT Center and Institute for High Performance Computing
- Prof. Dr. Matthias S. Müller, Joachim Protze,
  Dr. Christian Terboven

Partner from Japan

- University of Tsukuba, Center for Computational Sciences, and
  Advanced Institute of Computational Science, RIKEN
- Prof. Taisuke Boku, Dr. Hiroshi Murai, Miwako Tsuji

Partner from France

- Maison de la Simulation
- Prof. Serge Petiton, Prof. Nahid Emad

How many errors can you spot in this tiny example?

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

int main (int argc, char** argv)
{
   int rank, size, buf[8];

   MPI_Comm_rank (MPI_COMM_WORLD, &rank);
   MPI_Comm_size (MPI_COMM_WORLD, &size);

   MPI_Datatype type;
   MPI_Type_contiguous (2, MPI_INTEGER, &type);

   MPI_Recv (buf, 2, MPI_INT, size - rank, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

   MPI_Send (buf, 2, type, size - rank, 123, MPI_COMM_WORLD);

   printf ("Hello, I am rank %d of %d.\n", rank, size);

   return 0;
}
```

At least 8 issues in this code example

How many errors can you spot in this tiny example?

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

int main (int argc, char** argv)
{
    int rank, size, buf[8];

    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &size);

    MPI_Datatype type;
    MPI_Type_contiguous (2, MPI_INTEGER, &type);

    MPI_Recv (buf, 2, MPI_INT, size - rank, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    MPI_Send (buf, 2, type, size - rank, 123, MPI_COMM_WORLD);

    printf ("Hello, I am rank %d of %d.\n", rank, size);

    return 0;
}
```

No MPI_Init before first MPI-call
Fortran type in C
Recv-recv deadlock
Rank0: src=size (out of range)
Type not committed before use
Type not freed before end of main
Send 4 int, recv 2 int: truncation
No MPI_Finalize before end of main

Do we need a safety net in HPC?

Source: David Madison, gettyimages

Overview of defect classification from tiny MPI example

	Static	Dynamic
Local	Fortran Type in C	MPI_Init before first MPI call Rank out of range Type not committed before use Type not freed before finalize No call to MPI_Finalize
Distributed / Global		Recv-Recv Deadlock Type matching in messages

Examples of defect classification for XMP

	Static	Dynamic
Local	• *async-id* must be of type default integer • *array-name* must be declared before align directive	• The node set specified in the on clause must be a subset of the parent node set. • The source node specified by the from clause must belong to the node set specified by the on clause of bcast.
Distributed / Global		• Collective consistency • Deadlock • Co-array data race

Approach taken in MYX for Correctness Checking

- Automatic runtime correctness checking avoids the state explosion problem of model checking

- MUST: scalable correctness checking of MPI, OpenMP and hybrid programs
  - PnMPI used to intercept MPI calls

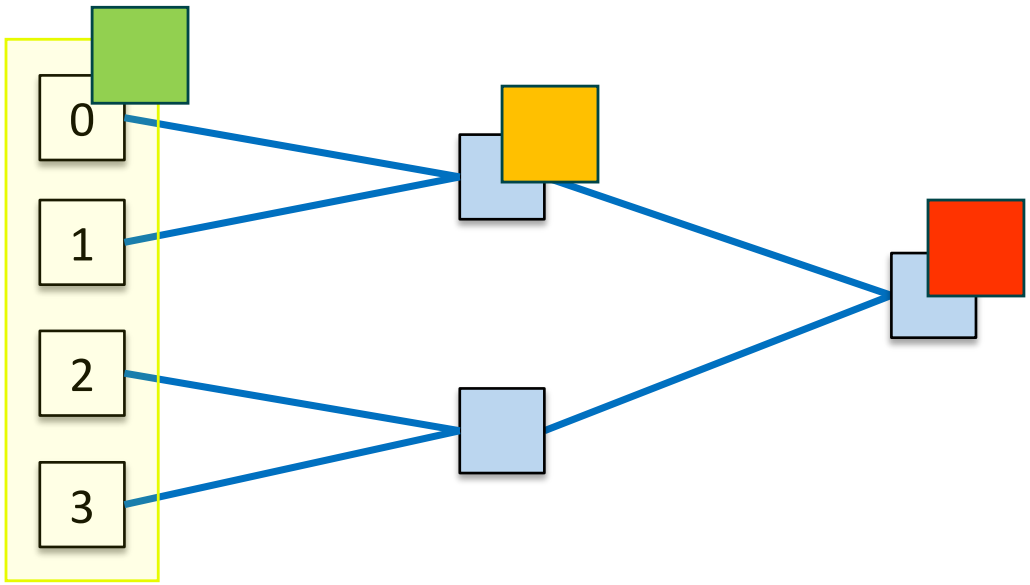
- Current use case example for MPI
  - MPI lacks type safety in the functions to send and receive messages
  - This may lead to invalid data, if receiver expects different data from what sender sent
  - Runtime analysis can find actual and potential instances of this problem

- Example for development in MYX
  - XMP as a PGAS language supports the distribution of globally shared data
  - XMP implementation relies on one-sided communication feature of MPI
  - Access to distributed data with the wrong access pattern may result in error

- MYX is a research project: focus lies on development of scalable correctness checking methods and answering of programming language design questions

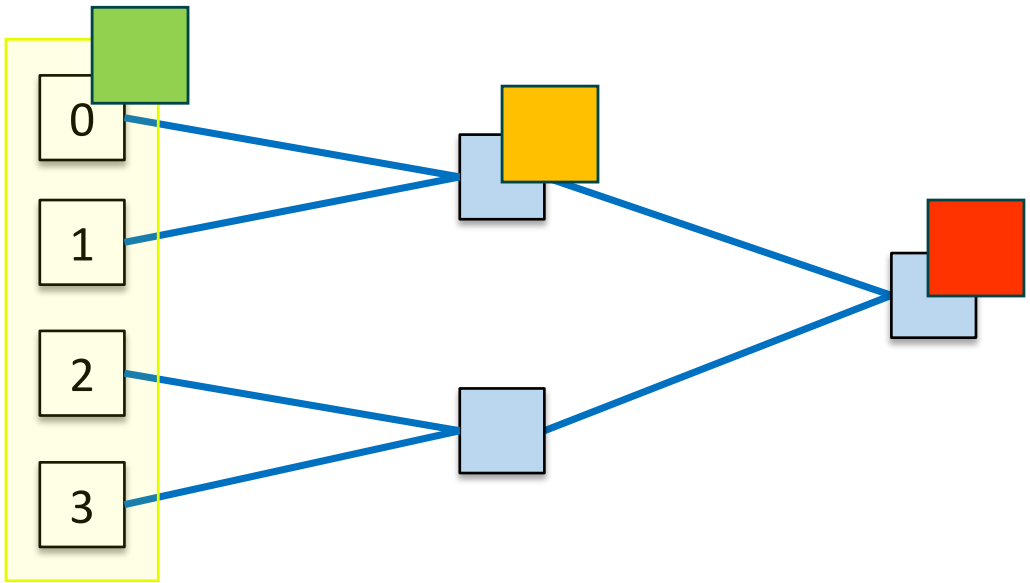
Distributed Agent-based Runtime Correctness Analysis

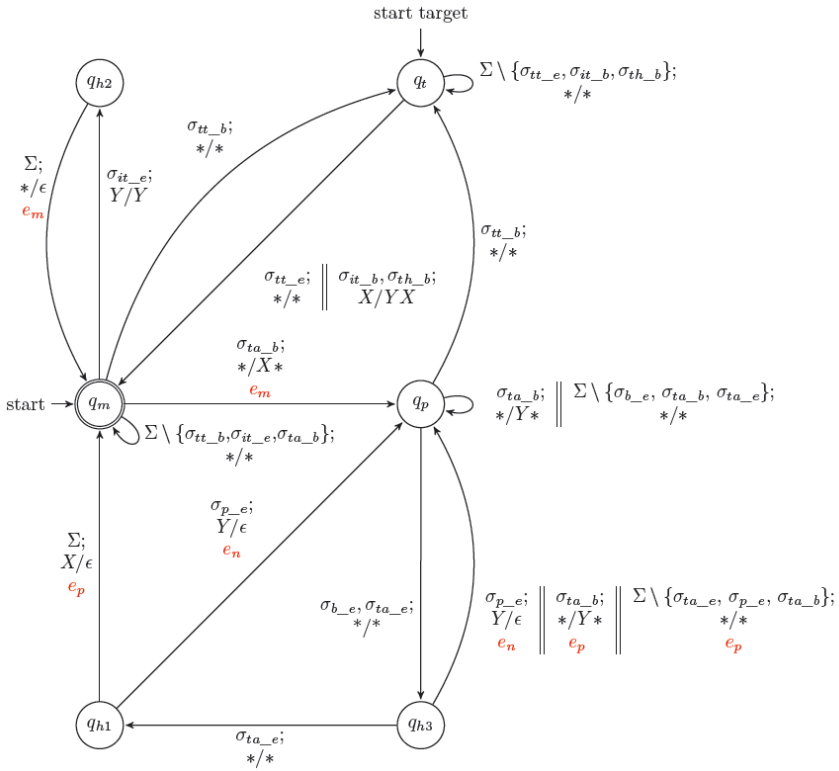
- Correctness analysis is done by specialized agents
- Locally whenever applicable
- Distributed for scalability
- Centralized only for global knowledge



Distributed Agent-based Runtime Correctness Analysis

- Correctness analysis is done by specialized agents
- Locally whenever applicable
- Distributed for scalability
- Centralized only for global knowledge

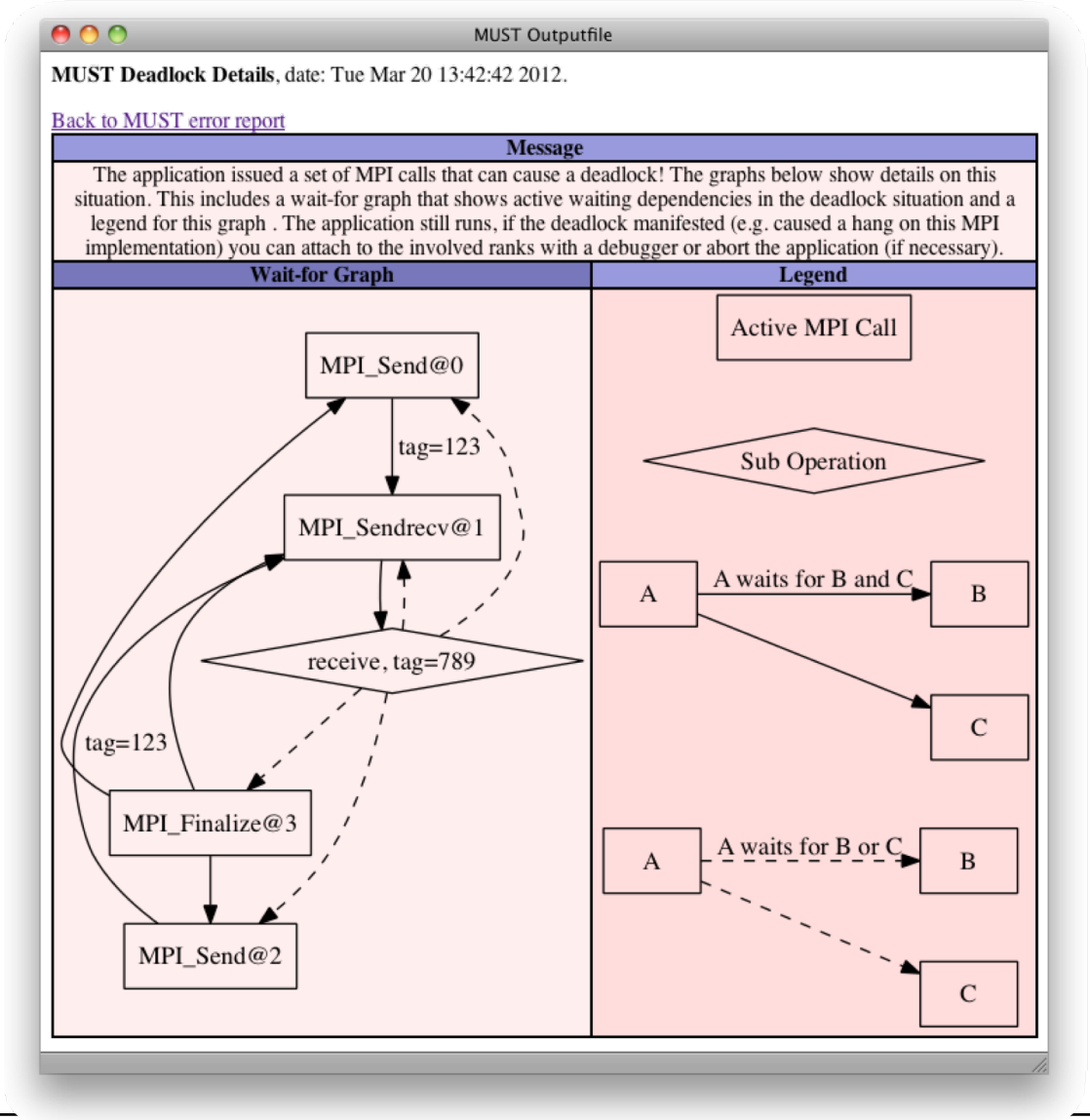




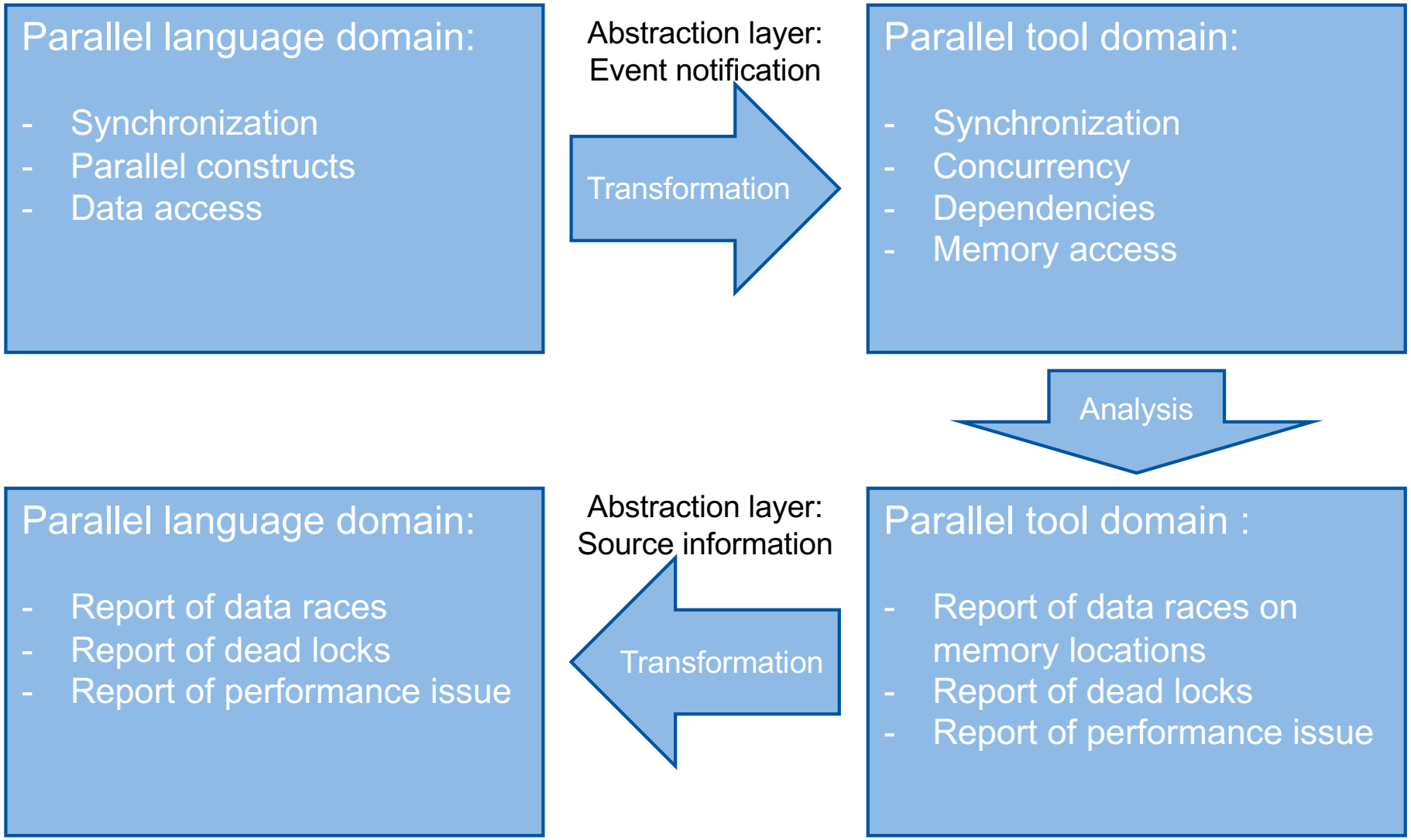
Examples for defects detected by MUST: type mismatch

Rank	Thread	Type	Message	From	References	MPI-Standard Reference
1		**Error**	A send and a receive operation use datatypes that do not match! Missmatch occurs at (CONTIGUOUS)[0] (MPI_INT) in the send type and at (MPI_BYTE) in the receive type (consult the MUST manual for a detailed description of datatype positions). The send operation was started at reference 1, the receive operation was started at reference 2. (Information on communicator: MPI_COMM_WORLD) (Information on send of count 1 with type:Datatype created at reference 3 is for C, commited at reference 4, based on the following type(s): { MPI_INT}Typemap = {(MPI_INT, 0), (MPI_INT, 4)}) (Information on receive of count 8 with type:MPI_BYTE)	**MPI_Sendrecv** from: #0 main@test.c:54 #1 start_main@libc.so	reference 1 rank 0: **MPI_Sendrecv** from: #0 main@test.c:54 #1 start_main@libc.so reference 2 rank 1: **MPI_Sendrecv** from: #0 main@test.c:54 #1 start_main@libc.so reference 3 rank 0: **MPI_Type_contiguous** from: #0 main@test.c:35 #1 start_main@libc.so reference 4 rank 0: **MPI_Type_commit** from: #0 main@test.c:51 #1 start_main@libc.so	

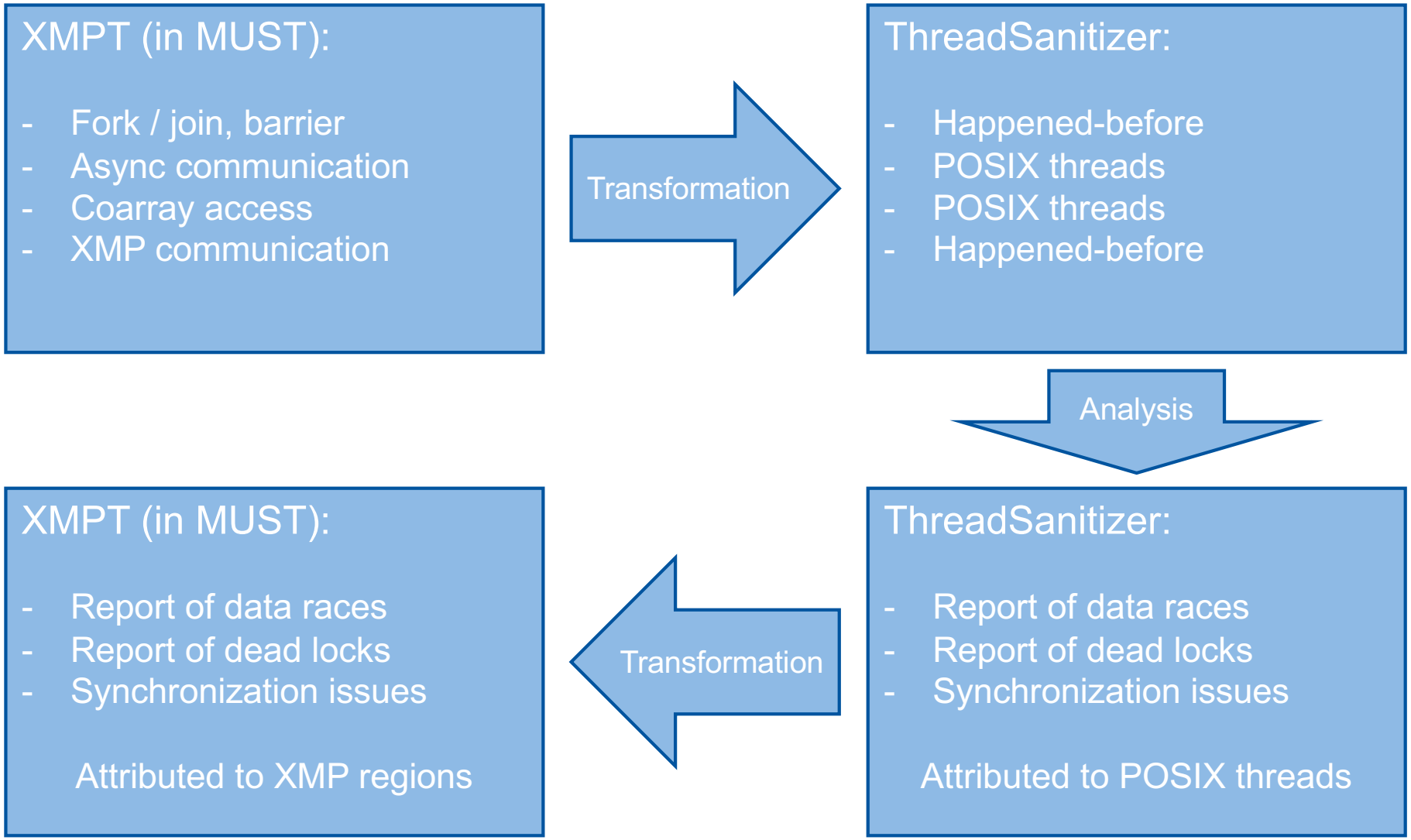
Examples for defects detected by MUST: deadlock



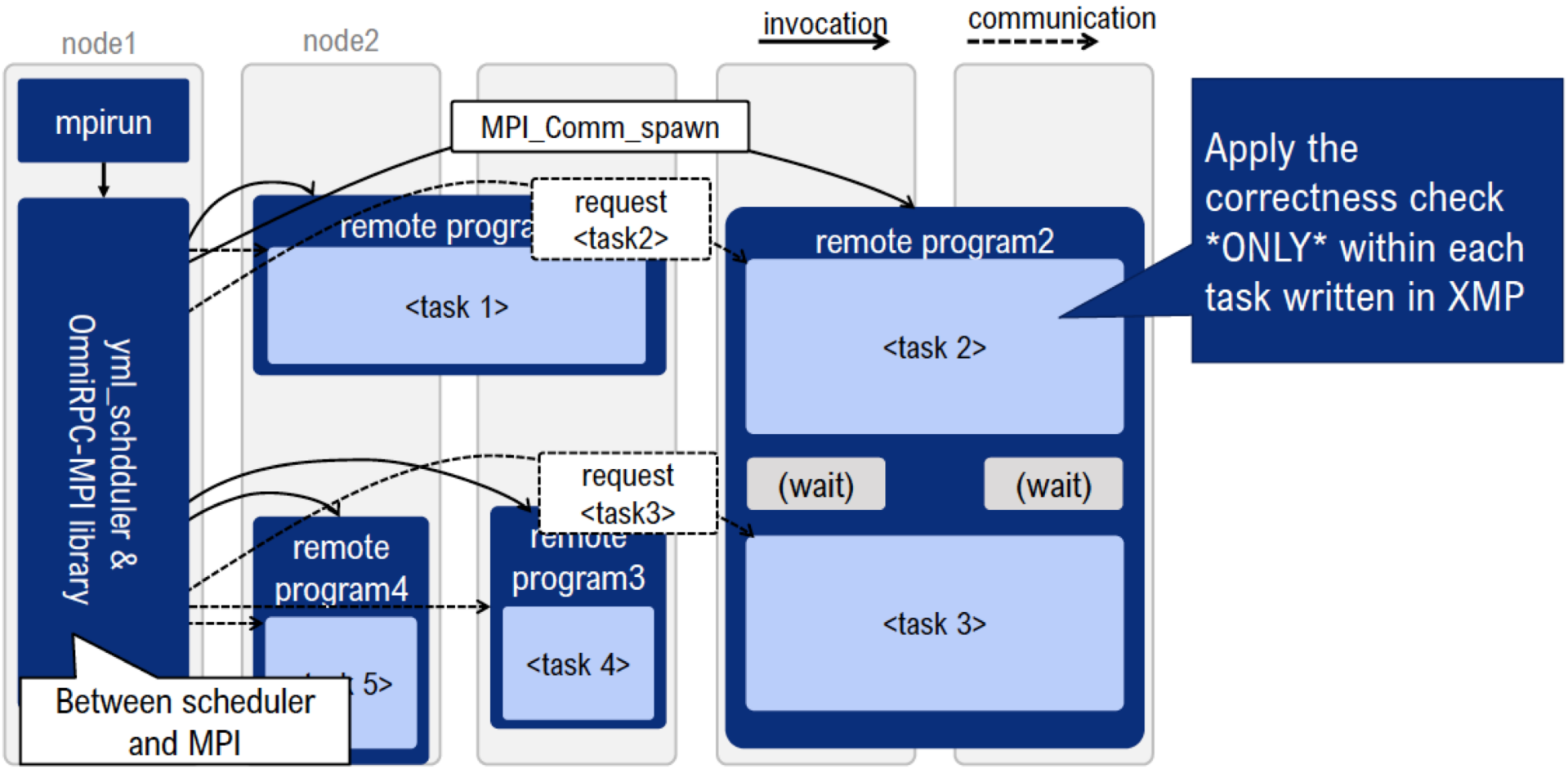
Abstract data / controlflow for correctness tools



Example: Data race detection for XMP



Correctness checking for YML+XMP programs



Applicability for XMP Coarray

- XMP Coarray do not allow local Coarray access beyond API usage
  - No conflict of API and non-API memory access possible
  - Full memory access tracking not necessary

- Analysis based on:
  - Tracking API memory accesses
  - Tracking API synchronization information

- Information is provided by the XMPT interface
  - Callbacks for coarray memory access
  - Callbacks for XMP synchronization

Due to the API and semantics of XMP the runtime analysis can be implemented much more efficiently (compared to MPI)

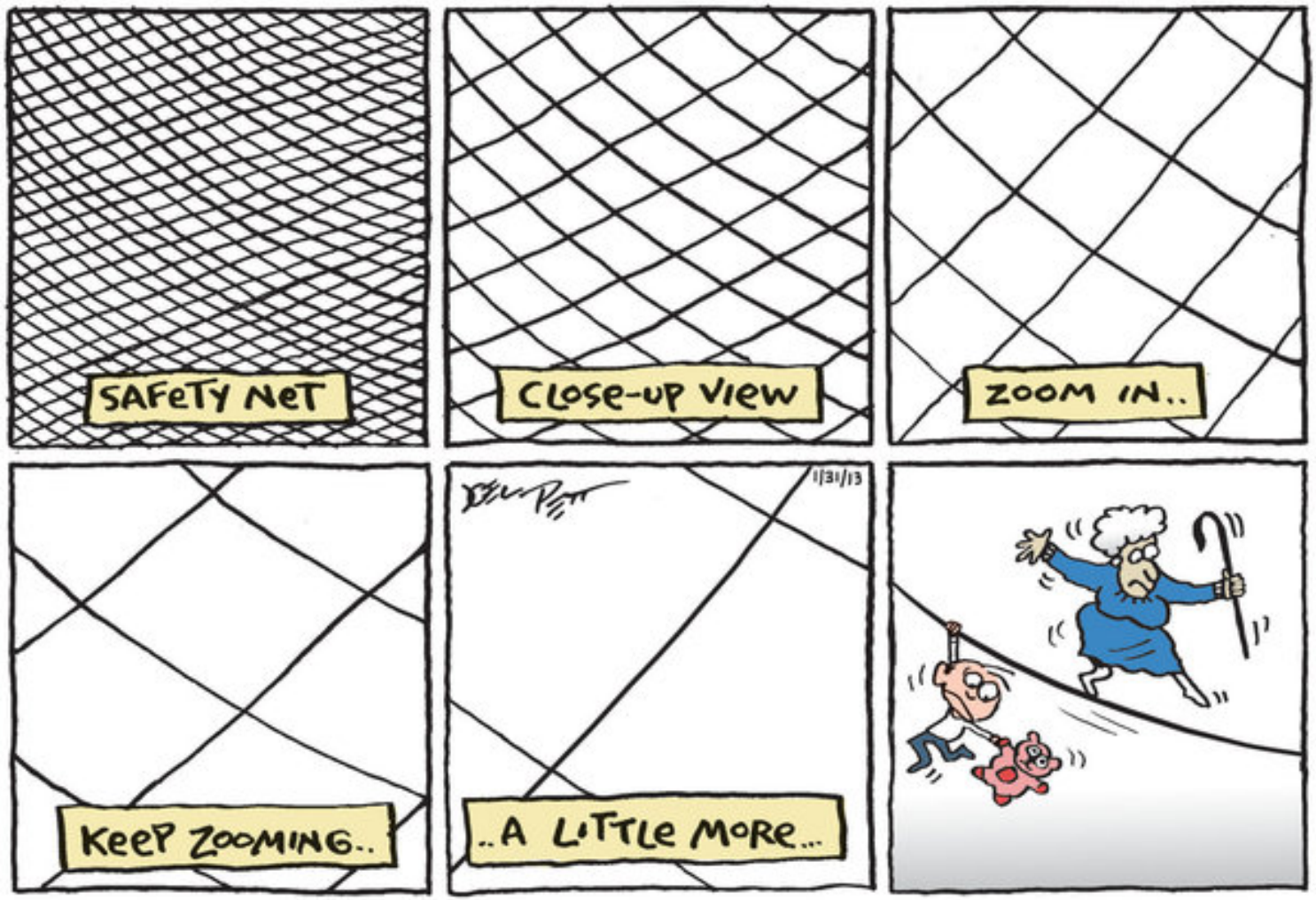
Summary and outlook

- Improved programming models and environments are important for Exascale and beyond.

- **Project goals and achievements of MYX**
  - **Extend correctness checking to XMP and YML**
  - **Transfer the tools interface from OpenMP to XMP**
  - **Improve existing parallel programming paradigms**
  - **Develop high level abstractions to express parallelism on modern HPC systems:**
    - **Tasking**
    - **Support for combination of static and dynamic analysis for correctness checking**

- More details:
  Joachim Protze, Christian Terboven, Matthias S. Mueller, Serge Petiton, Nahid Emad, Hitoshi Murai and Taisuke Boku:
  Runtime Correctness Checking for Emerging Programming Paradigms. First International Workshop on Software Correctness for HPC applications, Nov. 2017, Supercomputing

Safety net: the current approach is good, but they are still holes …



Source: Joel Pett in the Lexington Herald-Leader.
http://2.bp.blogspot.com/-ERKbUp4-7XI/URBmWgfkpAI/AAAAAAAAuHs/SU3bsu7Aop4/s1600/safety+net.jpg

Questions?

mueller@itc.rwth-aachen.de