MYX

MUST Correctness Checking for YML and XMP Programs

SPPEXA Annual Plenary Meeting 2018

Presenter: Matthias Müller / Joachim Protze

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, Thomas Dufaud

SPPEXA Workshops on Parallel Programming Models - Productivity and Applications

SPPEXA Workshop in Japan Spring 2017

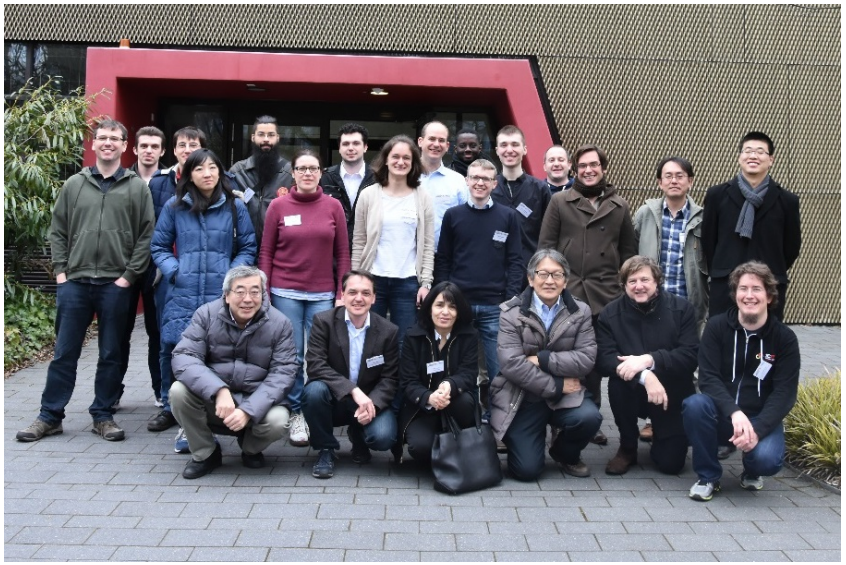
- Thursday, April 6th, 2017, 9:00 – 18:00
- Organized by University of Tsukuba

SPPEXA Workshop in France Fall 2017

- Wednesday, Octobre 18th, 2017, 9:00 – 18:00
- Organized by University of Versailles

SPPEXA Workshop in Germany Spring 2018

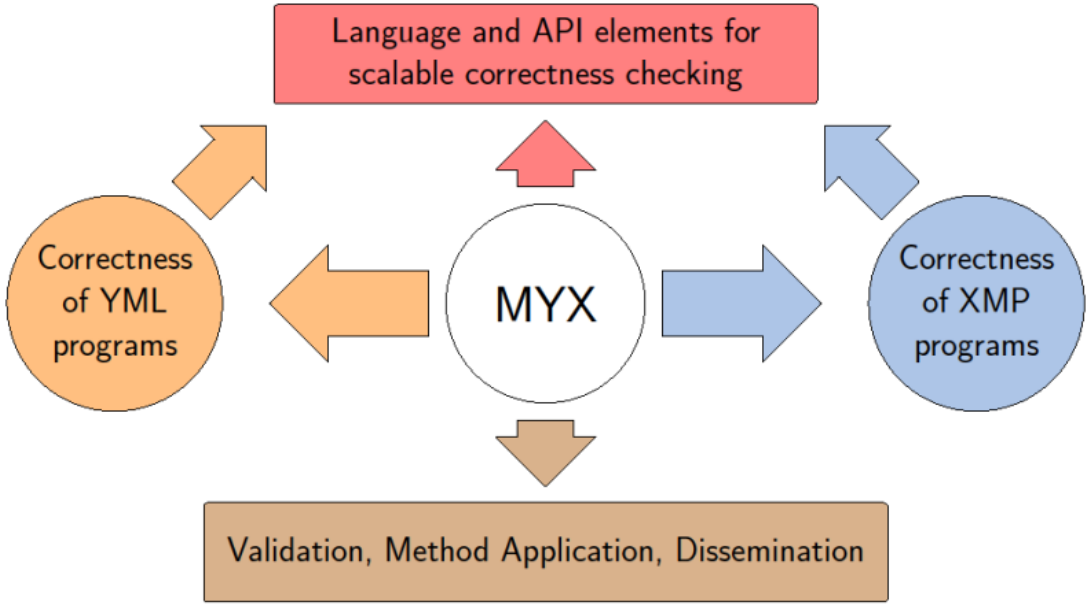
- Thursday, March 16th, 2018, 9:00 – 18:00
- Organized by RWTH Aachen University



Research Challenges and Project Results

- The more parallelism expressed, the higher the chance of errors being made
- Time of programming error search and fix: productivity loss!
  - Automatic correctness checking may be used to avoid that

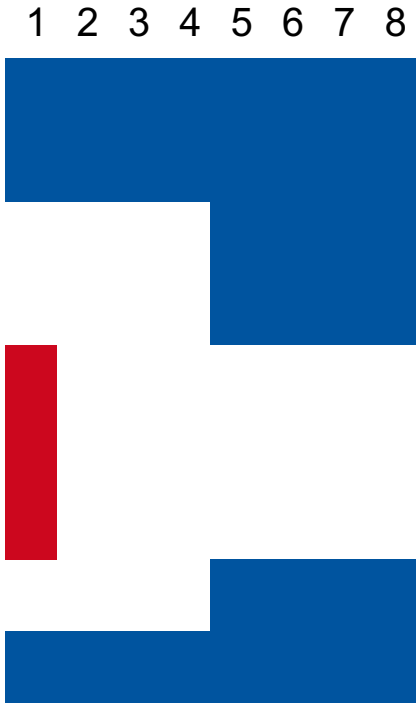
- MYX objectives are
  - enable productivity improvements by means of scalable correctness checking
  - of YML- and XMP-programs
    - XMP: PGAS, with both global-view and local-view
    - YML: graph of components language
  - guide the development of future programming models



First results of XMP + MUST integration

- We successfully reused parts of the MUST infrastructure for MPI handles
- We generated first correctness reports for XMP applications
  - Analyzing local properties, we test that MUST has the right understanding of XMP applications

```
#pragma xmp nodes p(8)
int main(int argc, char** argv){
  xmp_init(&argc, &argv);
#pragma xmp task on p(5:8)
  { do_some_stuff();
#pragma xmp task on p(1)
    {
      printf("PASS\n");
    }
  do_some_other_stuff(); }
  return 0;
}
```

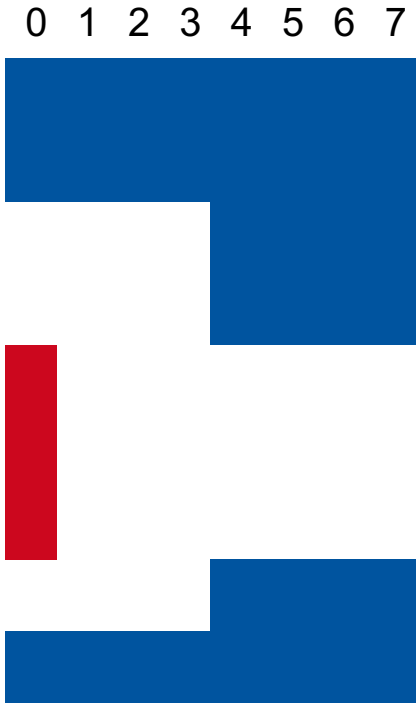


First results of XMP + MUST integration

- In the equivalent MPI code, a runtime tool would not be able to detect the issue

- XMP programs express more semantic execution logic in code and enables more runtime analysis

```
int main(int argc, char** argv){
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(…, &rank);
  if(rank >= 4)
  { do_some_stuff();
    if(rank == 0)
    {
      printf("PASS\n");
    }
  do_some_other_stuff(); }
  return 0;
}
```



Next step: analyses with need for distributed / global knowledge

- On collective operations, all processes need to provide matching arguments
  - This is similar to the requirements for MPI collective communication
  - MUST needs to understand who is expected to participate on the call

- Analysis of XMP Coarray memory accesses
  - Important preparatory work: MA by Simon Schwitanski as presented yesterday
  - Information of XMPT events will be feed into MPI-One sided analysis

- Future plans:
  - Analysis of DASH

Technology used for implementation: MUST + ThreadSanitizer

- MUST provides:
  - API function tracking
  - Communication and cross-process analysis

  - MPI runtime correctness checking framework

- ThreadSanitizer provides:
  - Memory access tracking
  - Analysis of conflicting memory accesses

  - Data race detection tool delivered with LLVM/GNU compilers

Example: Data race detection for XMP

We use ThreadSanitizer as established tool and research vehicle (In theory ThreadSanitizer can be replaced by any other data race detection engine)

But ThreadSanitizer has no idea of XMP one-sided memory accesses and synchronization

ThreadSanitizer:

- Happened-before
- POSIX threads

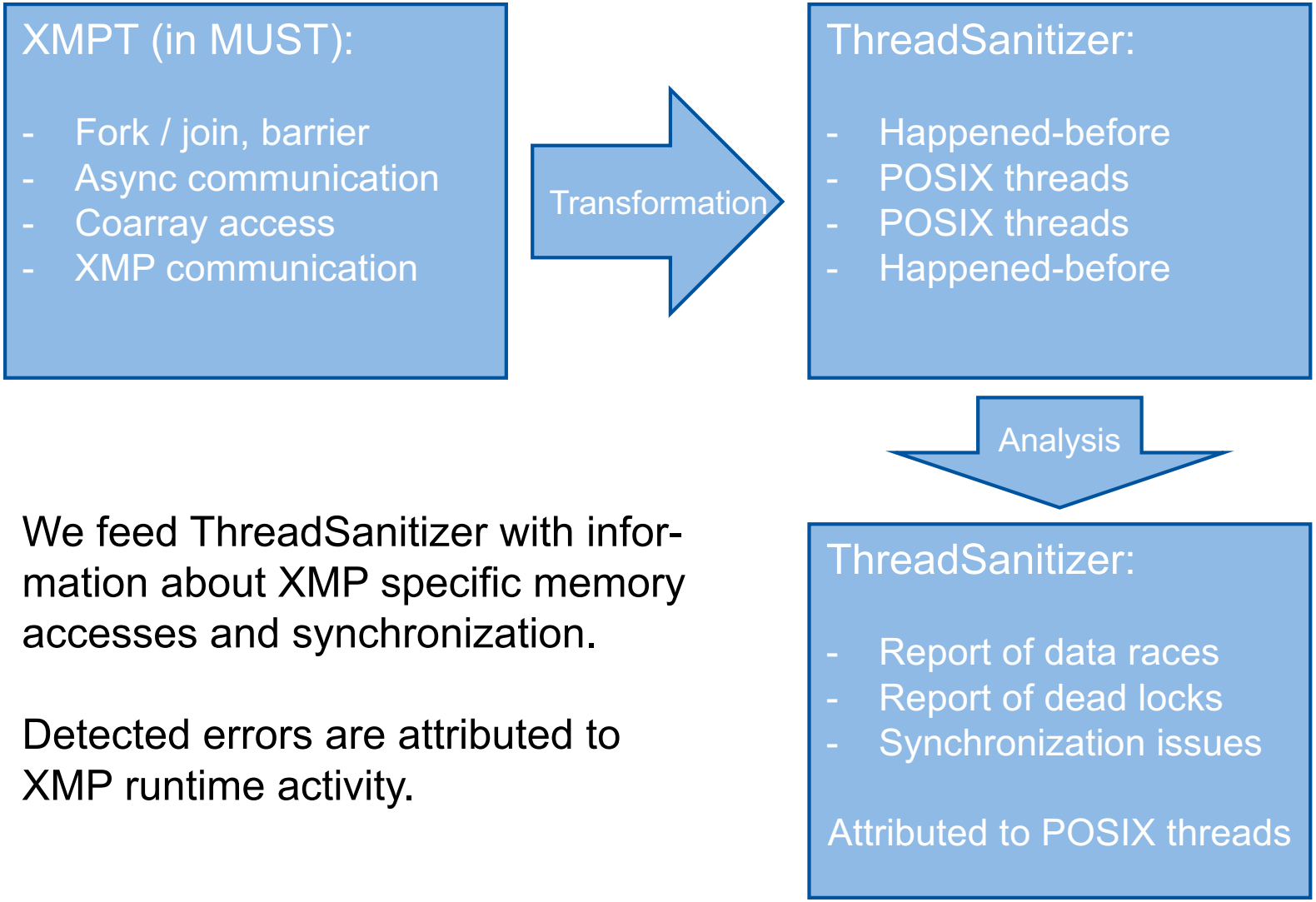
Analysis

ThreadSanitizer:

- Report of data races
- Report of dead locks
- Synchronization issues

Attributed to POSIX threads

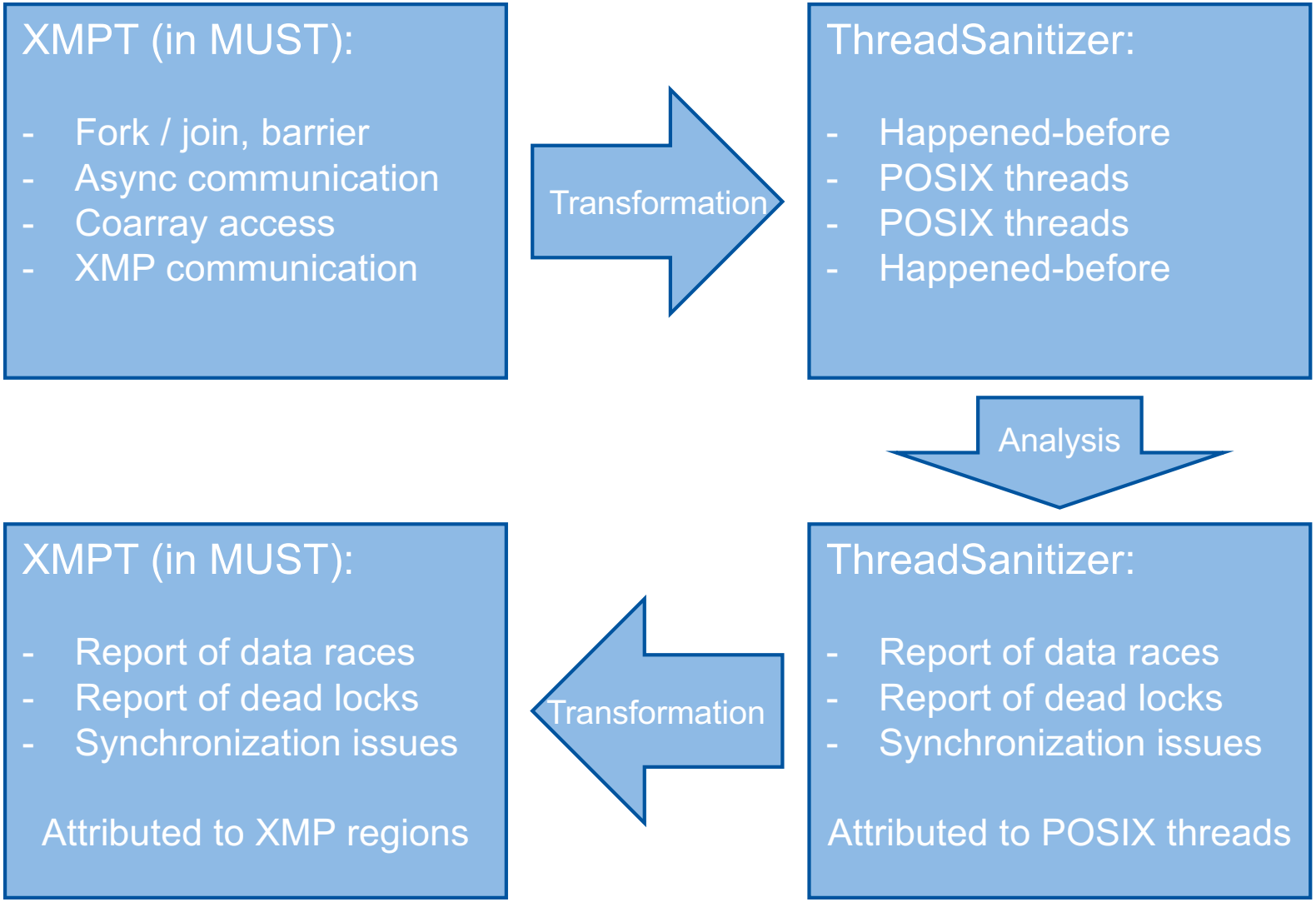
Example: Data race detection for XMP



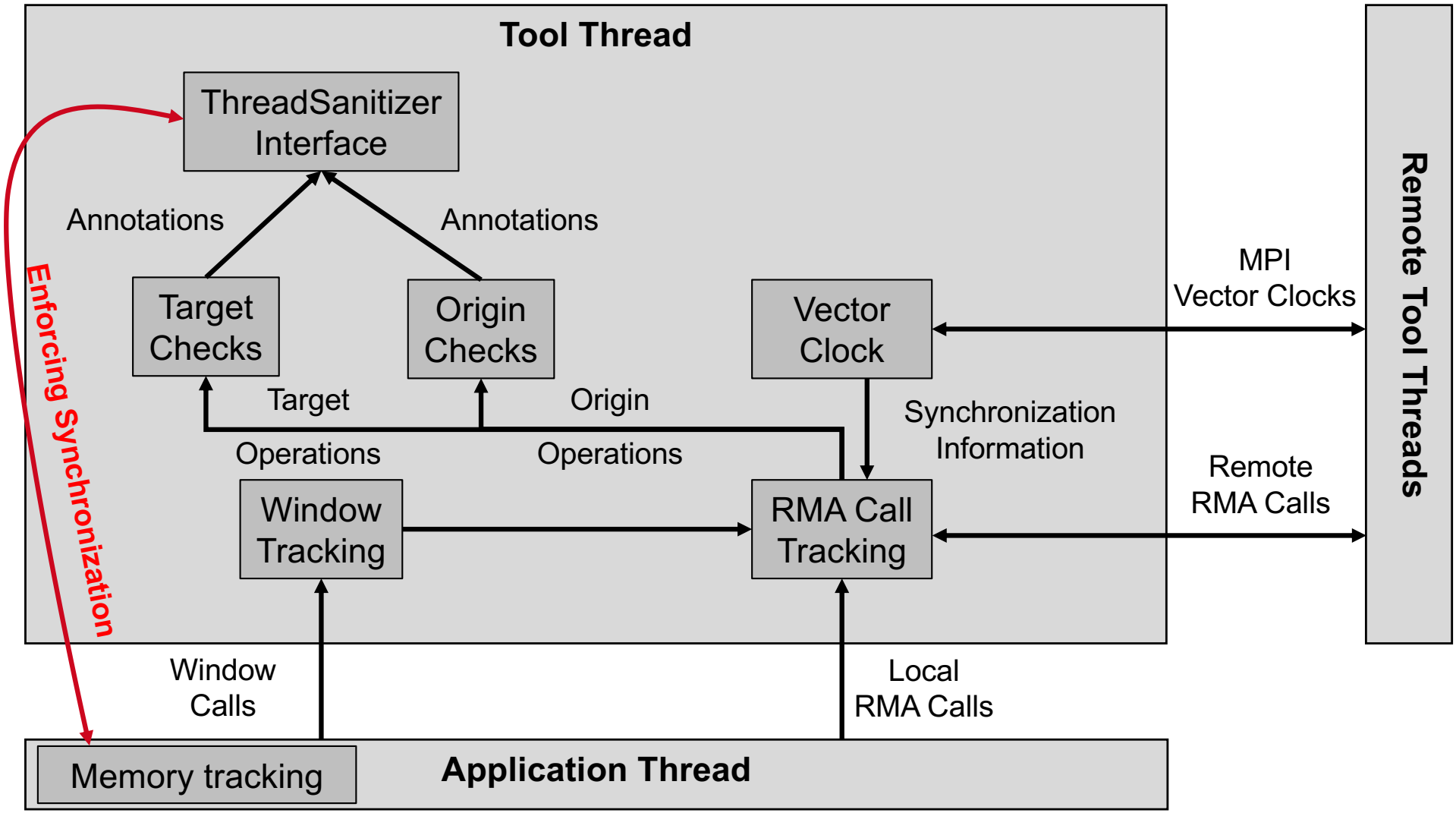
We feed ThreadSanitizer with information about XMP specific memory accesses and synchronization.

Detected errors are attributed to XMP runtime activity.

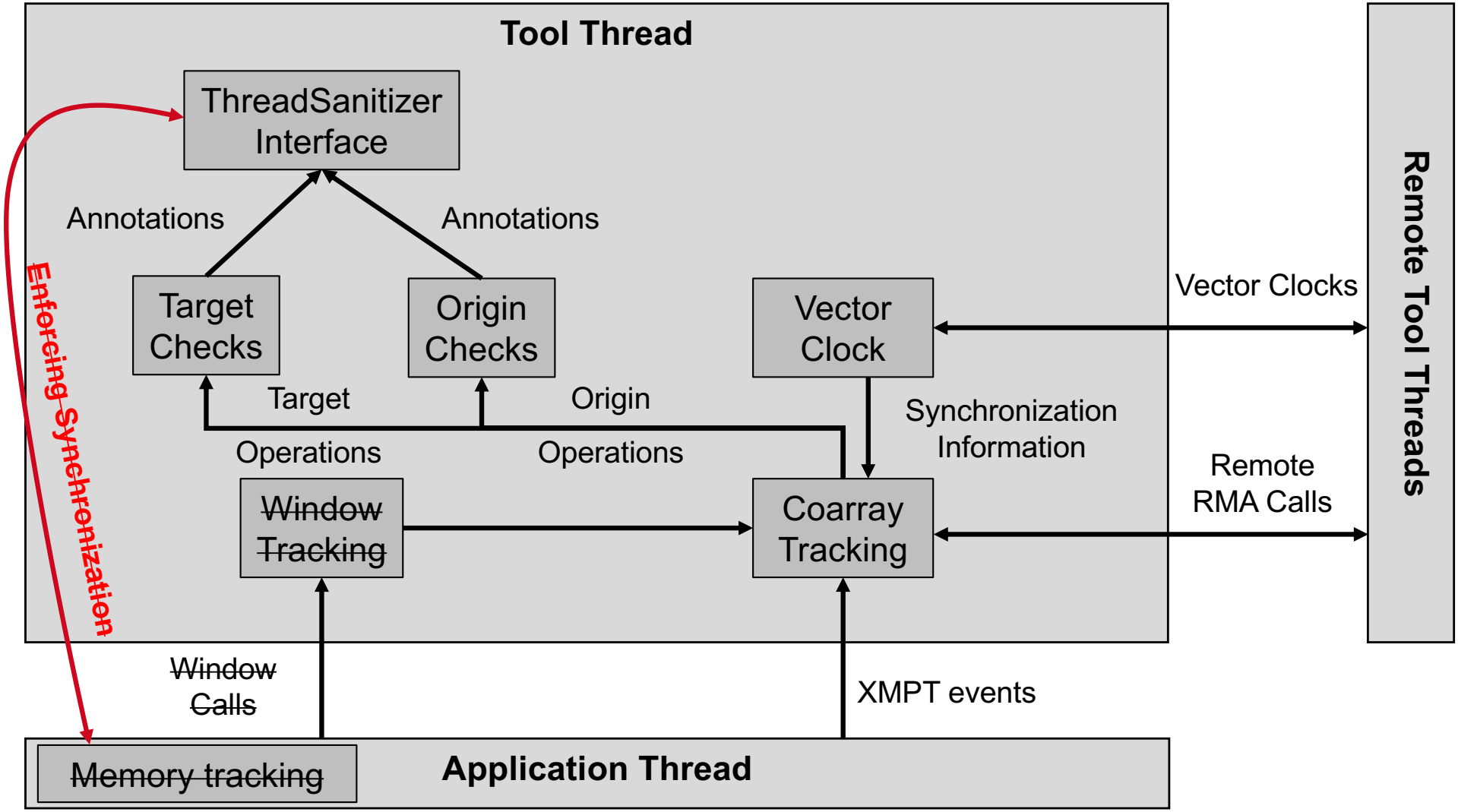
Example: Data race detection for XMP



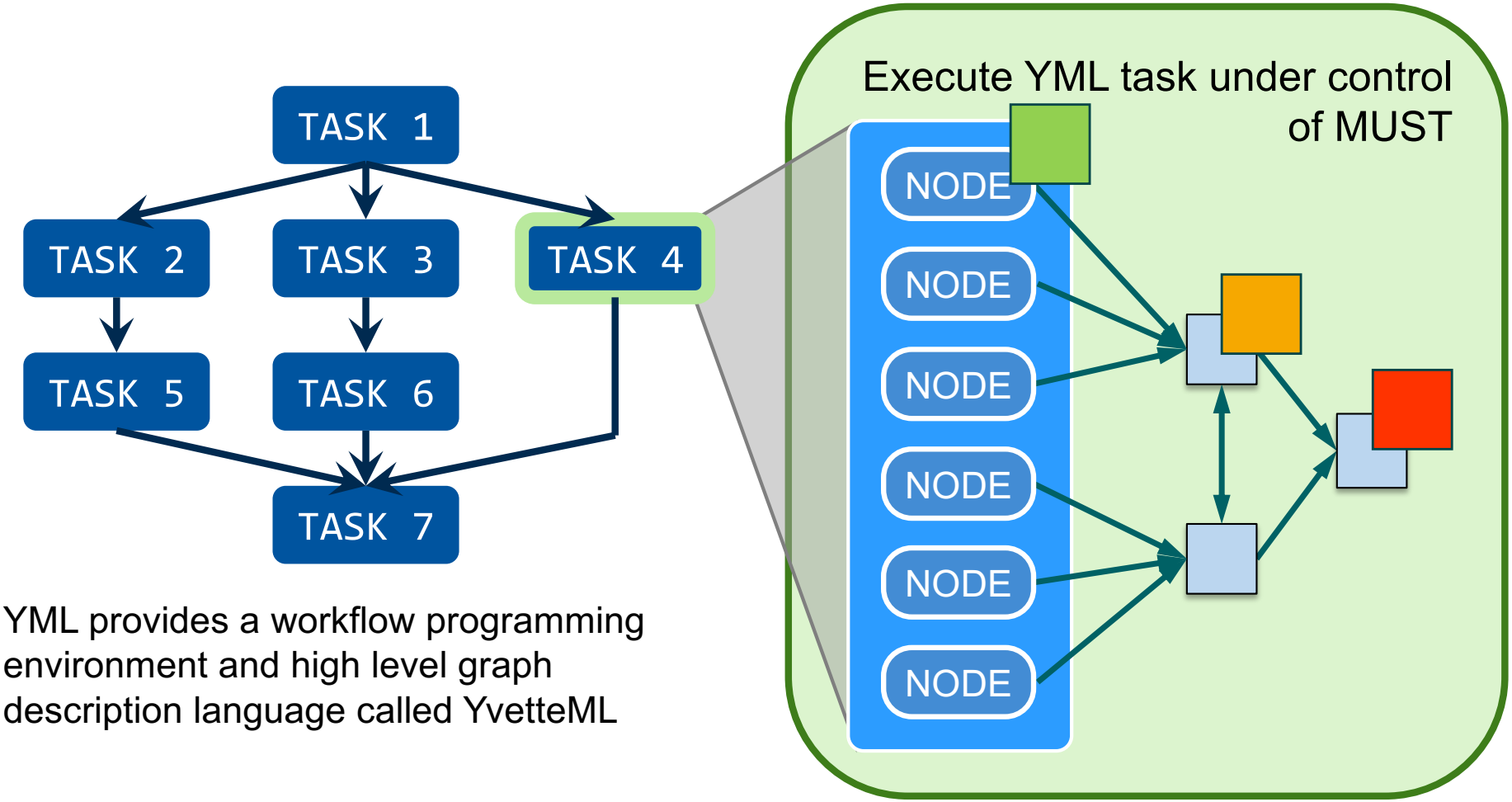
Tool Architecture: MPI Onesided



Tool Architecture: XMP Coarray



MUST analysis of YML tasks



YML provides a workflow programming environment and high level graph description language called YvetteML

Results

- Specification and implementation of XMP tools interface
- First MUST analyses based on XMPT show advantages of XMP over MPI
- Prototype for data race detection in MPI one-sided communication
  - Base slow-down 5-20x for logging & analysis of memory access
  - Not yet scalable for MPI, but for XMP less synchronization needed
  - False alerts and omission for MPI, but no problem for XMP expected
- Workflow for YML applications:
  - MUST can be applied on a per-task basis
- Future work:
  - Analysis of XMP collective operations
  - Analysis of DASH

Vielen Dank für ihre Aufmerksamkeit