Correctness Analysis for One-Sided Communication in MUST

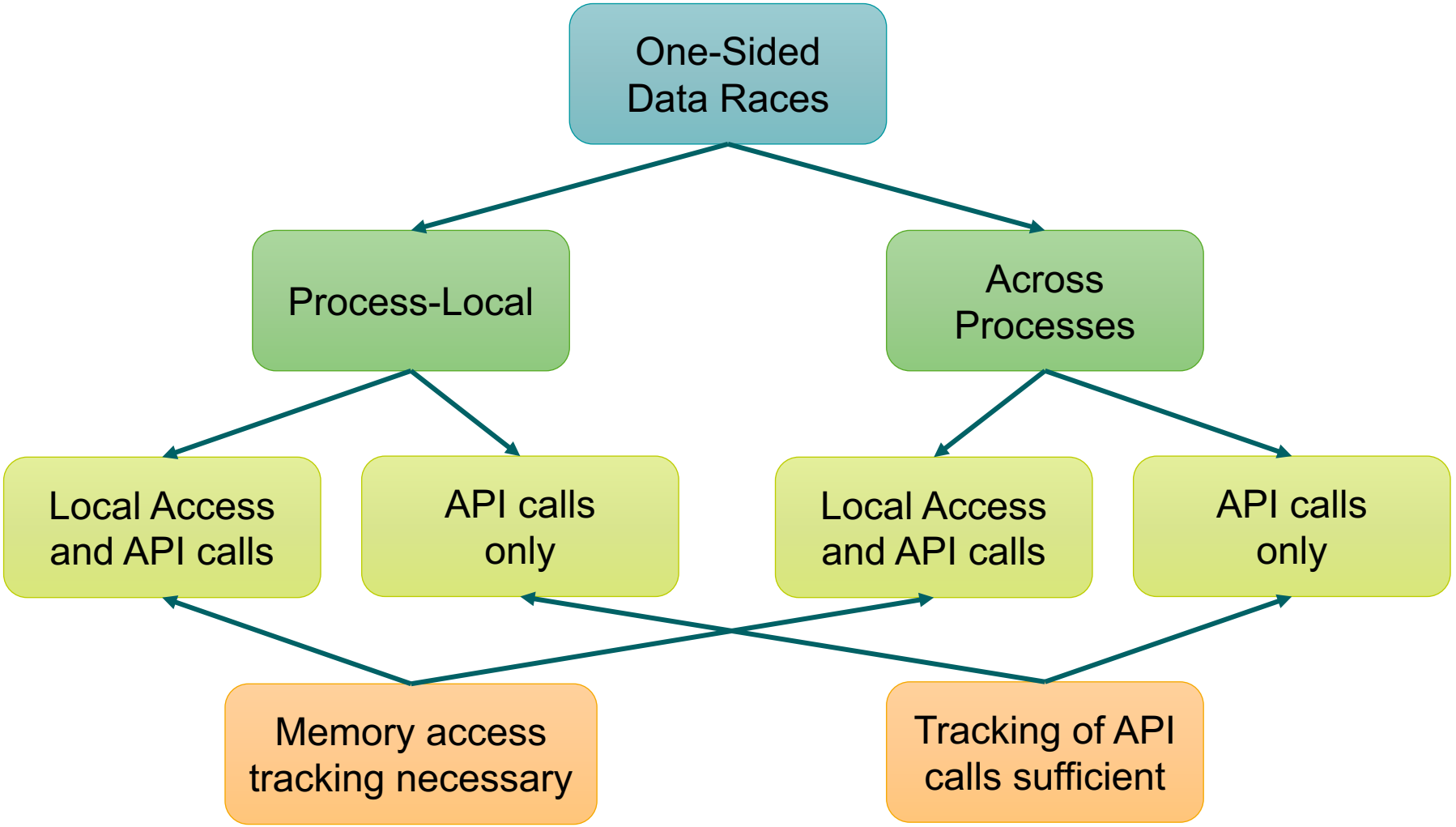
Prof. Dr. Matthias S. Müller (mueller@itc.rwth-aachen.de)
Joachim Protze (protze@itc.rwth-aachen.de)
Simon Schwitanski (simon.schwitanski@rwth-aachen.de)

High Performance Computing
RWTH AACHEN UNIVERSITY

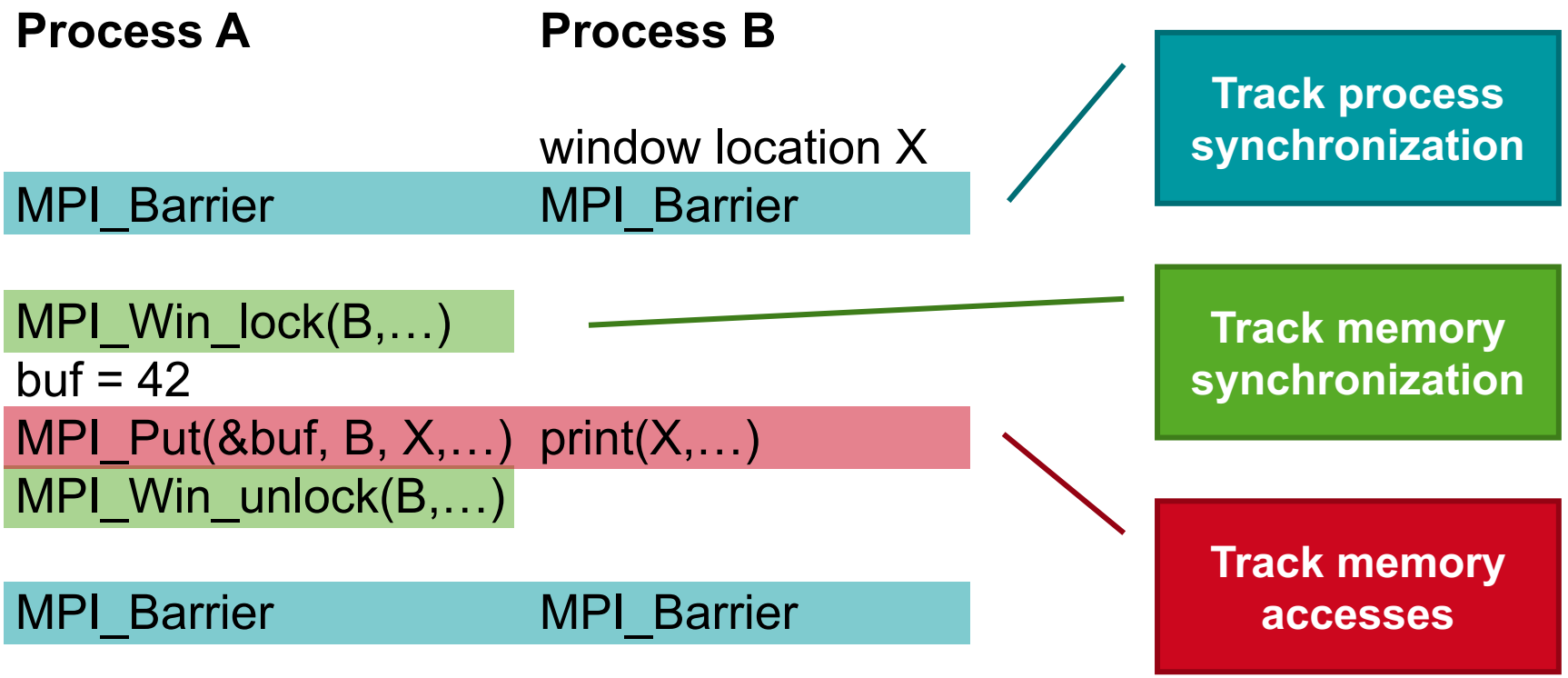
Motivation

- Many PGAS implementations base on a one-sided communication model
- One-sided communication is prone to data race
- What is necessary to analyze a PGAS application for data race?

Data race classes in PGAS

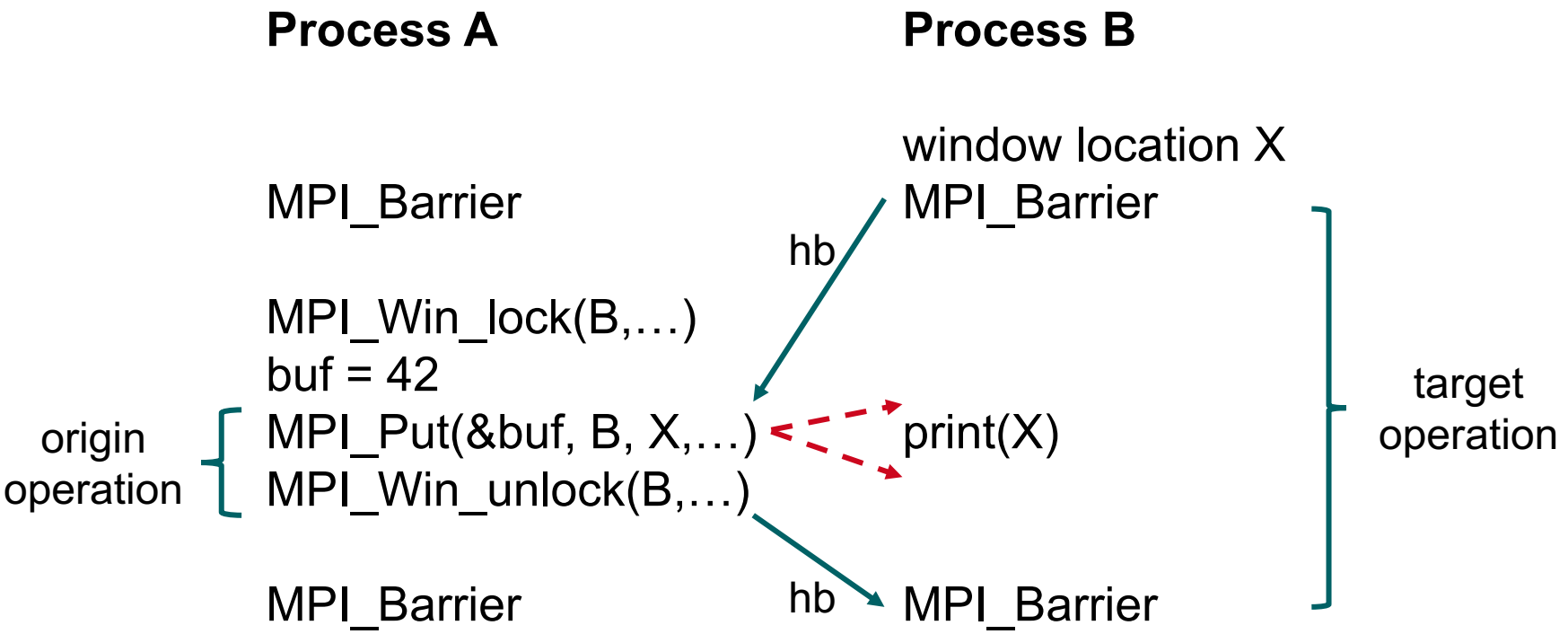


Example for a data race in MPI one-sided



Formal analysis

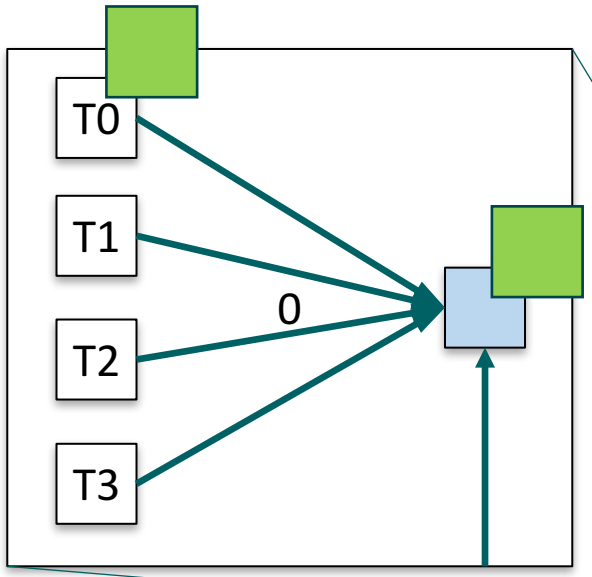
- Identify interval of origin operation
- Identify interval of target operation



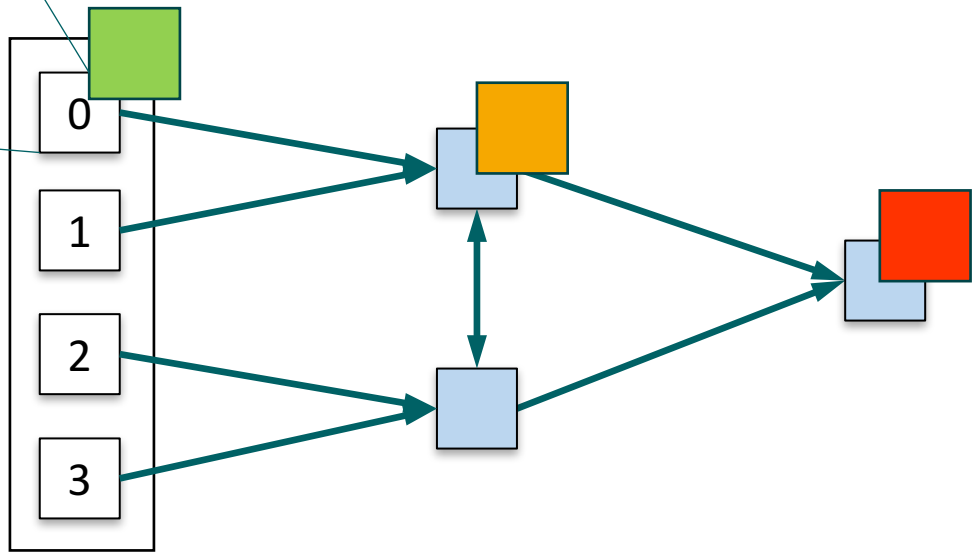
Technology used for implementation: MUST + ThreadSanitizer

- 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

MUST Analysis stages

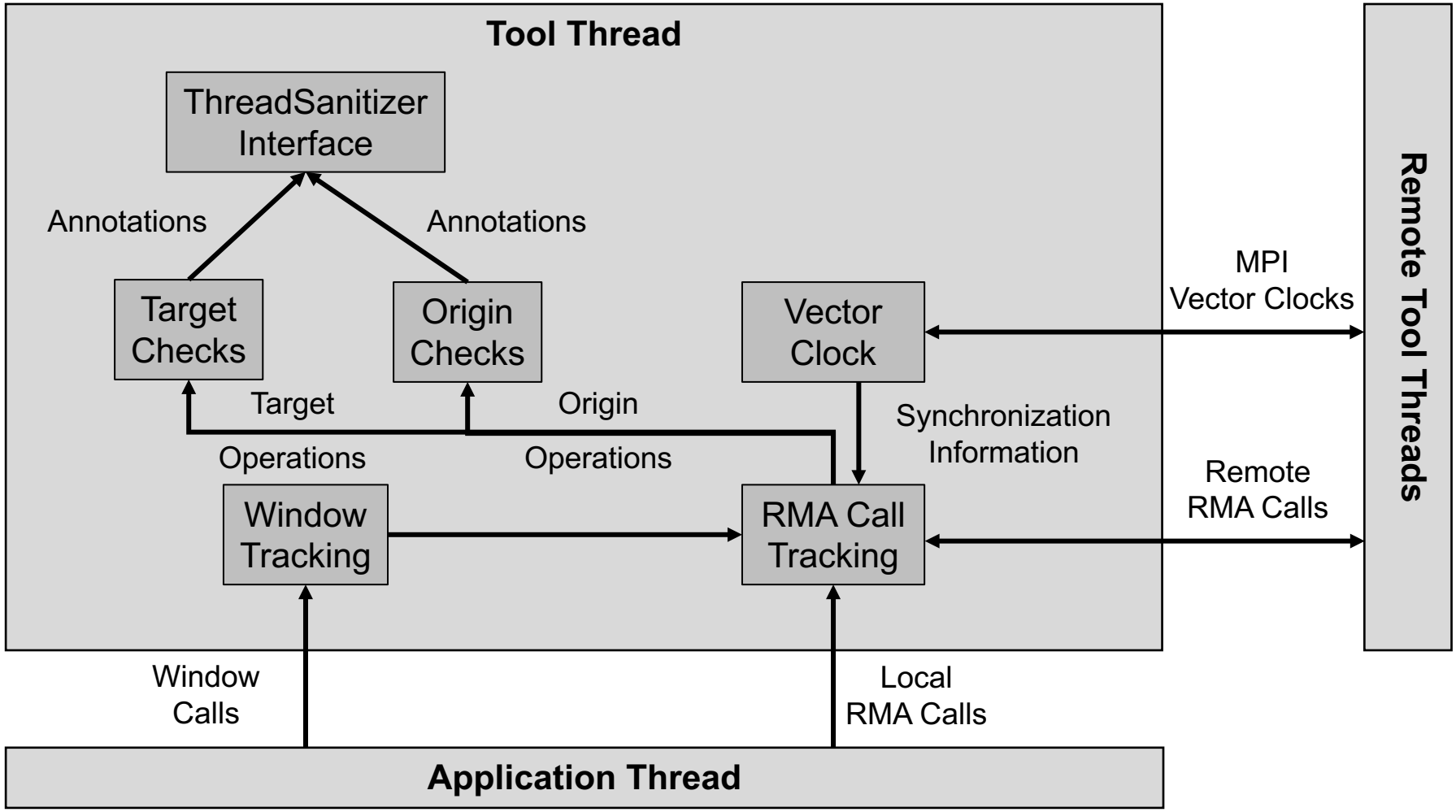


- Local memory access traced by TSan
- RMA access information propagated to target process
- Tool thread at target process reports memory access to TSan

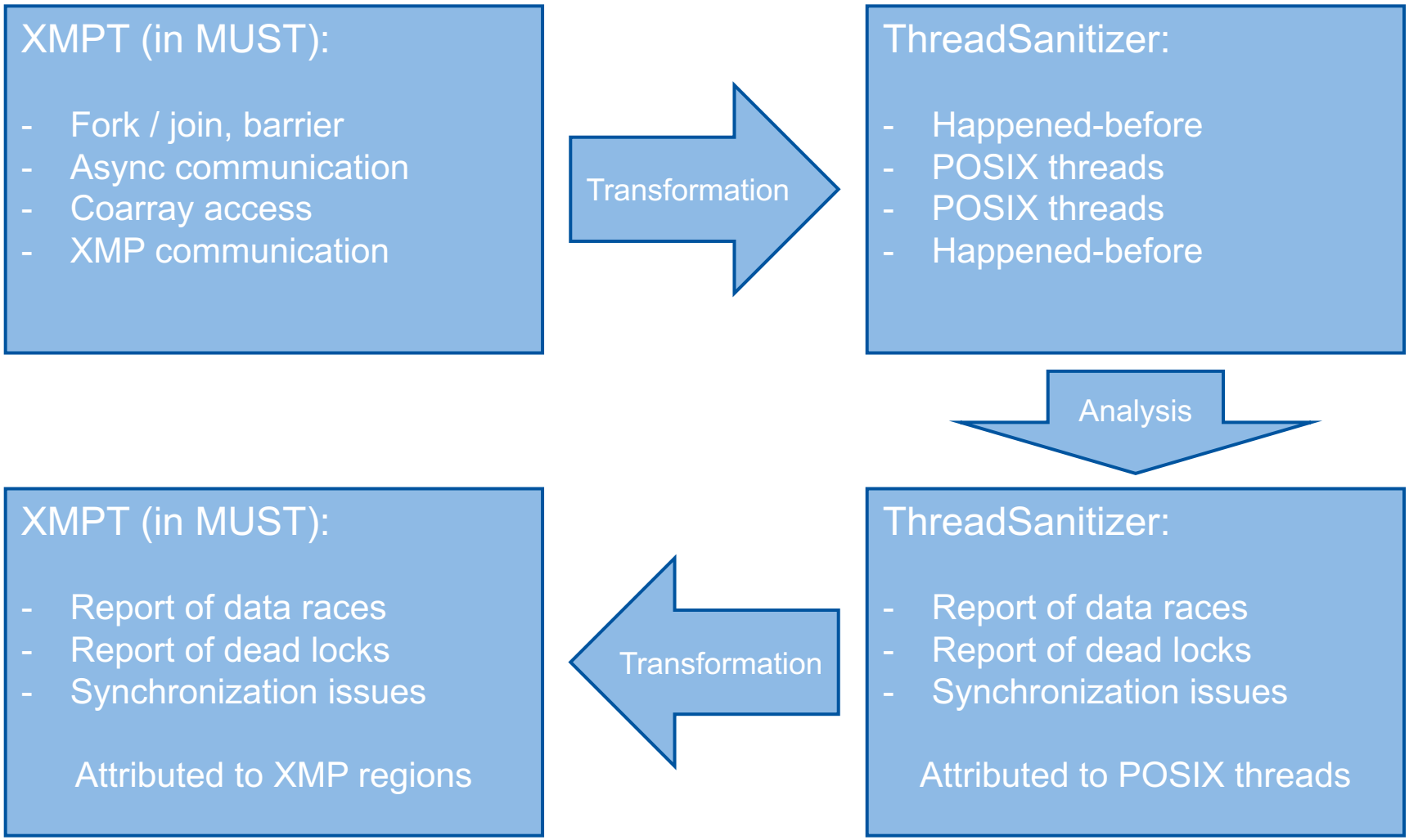


- Tool thread enables communication between application processes
  - ➢ Guarantee of progress in communication

Tool Architecture



Example: Data race detection for XMP



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

Summary and Conclusion

- Technologies developed for runtime correctness checking of MPI one sided communication can be applied to PGAS languages

- Current prototype implementation still has some significant disadvantages:
  - False positives in case of polling in unified memory model (benign data races)
  - slow-down 5-20x for logging & analysis of memory access

- XMP language specifications allow for low overhead run time checking. IMHO this is an important design point of a parallel programming paradigm.

- More information, including a formal model:
  Master Thesis, Simon Schwitanski “On-the-Fly Data Race Detection in MPI One-Sided Communication”

Thank you for your attention.

Acknowledgement:
Most of the work was implemented by Simon Schwitanski as part of his master thesis.