On-the-Fly Data Race Detection in MPI One-Sided Communication

Presentation – Master Thesis

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

SPPEXA Annual Plenary Meeting 2018
March 21, 2018

i12 High Performance Computing | RWTH AACHEN UNIVERSITY

Motivation

- Traditional MPI communication model is two-sided
- Modern HPC cluster networking architectures: RDMA
  - Increasing popularity of one-sided communication models
- MPI Remote Memory Access (RMA)
  - Used in implementations of PGAS paradigms
- **Problem:** MPI RMA programming error-prone
- New kind of programming mistakes: **Data Races**

Motivation

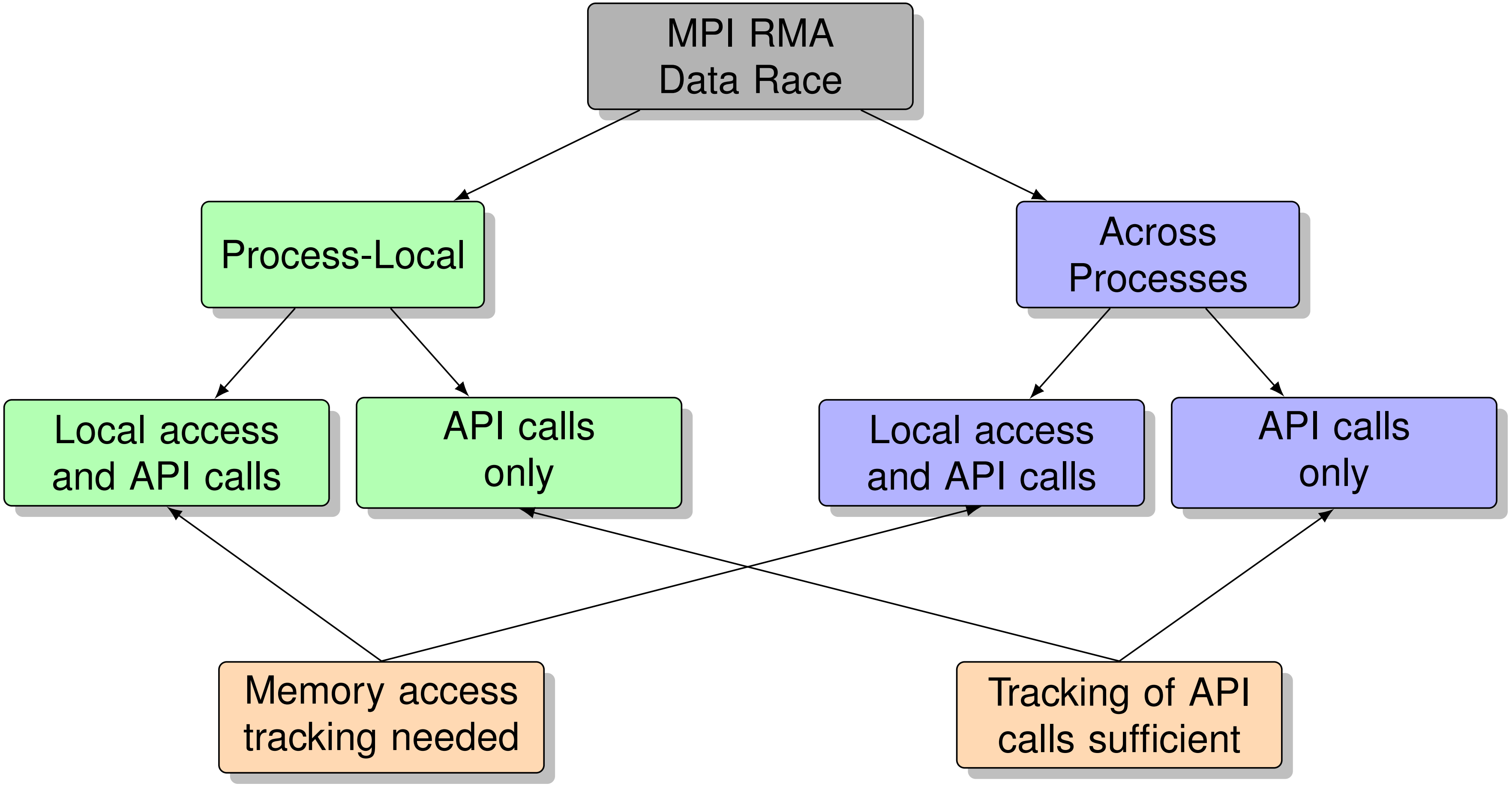
Process A	Process B
	window location X
MPI_Barrier	MPI_Barrier
MPI_Win_lock(B)	
buf = 42	
MPI_Put(&buf, B, X)	print(X)
MPI_Win_unlock(B)	
MPI_Barrier	MPI_Barrier

- Data race between process A and B at window location X
- How can we detect such data races?

Thesis Goals

- Classification of data races in MPI RMA
- Design of an on-the-fly data race detection model
- Implementation in correctness checking framework MUST
- Evaluation with different test cases

Data Race Classes


MPI RMA Data Race
Process-Local
Across Processes
Local access and API calls
API calls only
Local access and API calls
API calls only
Memory access tracking needed
Tracking of API calls sufficient

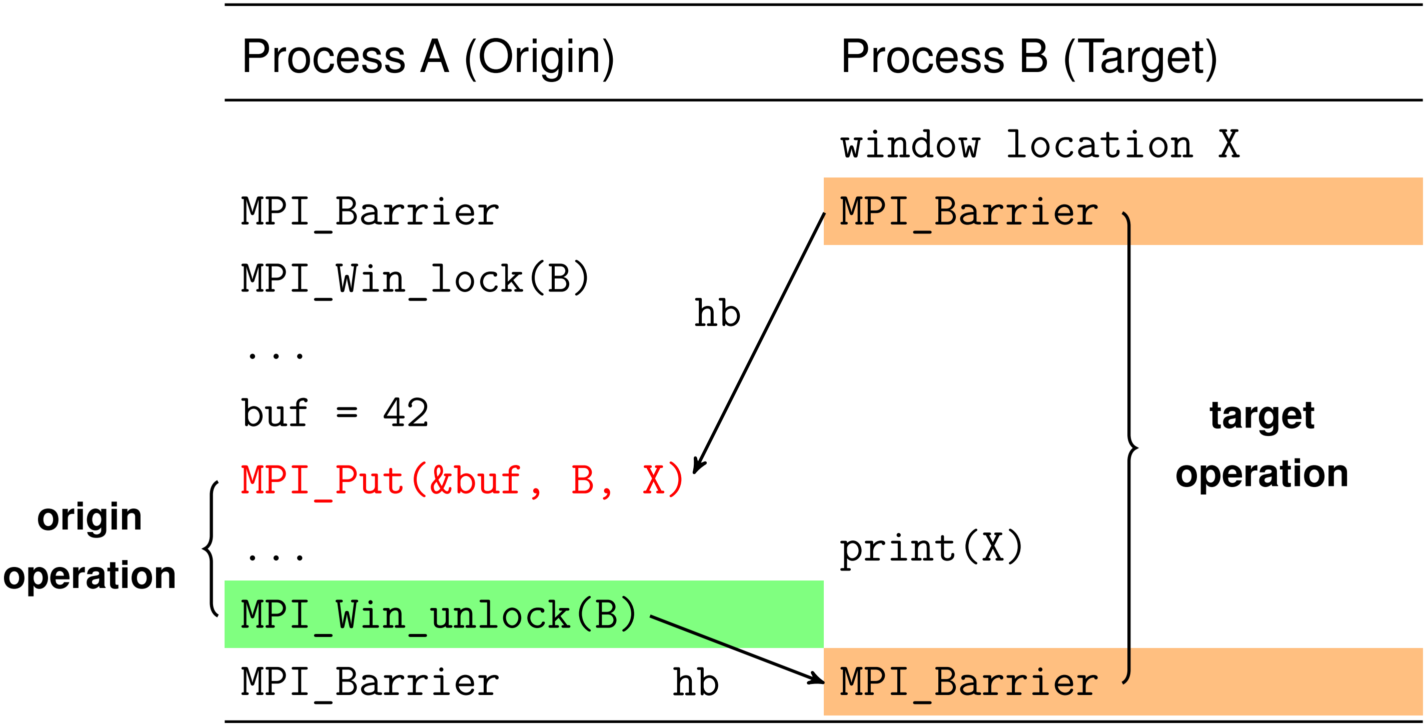
Race Detection – Formalization

- Formalize MPI RMA semantics
  - Hoefler et al. “Remote Memory Access Programming in MPI-3”
  - Track consistency order $\xrightarrow{co}$ (memory synchronization)
  - Track happens-before order $\xrightarrow{hb}$ (process synchronization)

Process A (Origin)	Process B (Target)
	`window location X`
`MPI_Barrier`	`MPI_Barrier`
`MPI_Win_lock(B)`	
`buf = 42`	
`MPI_Put(&buf, B, X)`	`print(X)`
`MPI_Win_unlock(B)`	
`MPI_Barrier`	`MPI_Barrier`

Race Detection – Concurrent Regions

- Local access: Origin operation / Remote access: Target operation
- Identify *concurrent regions* of origin and target operations
  - Cover *earliest* and *latest* point in time when operation can take place

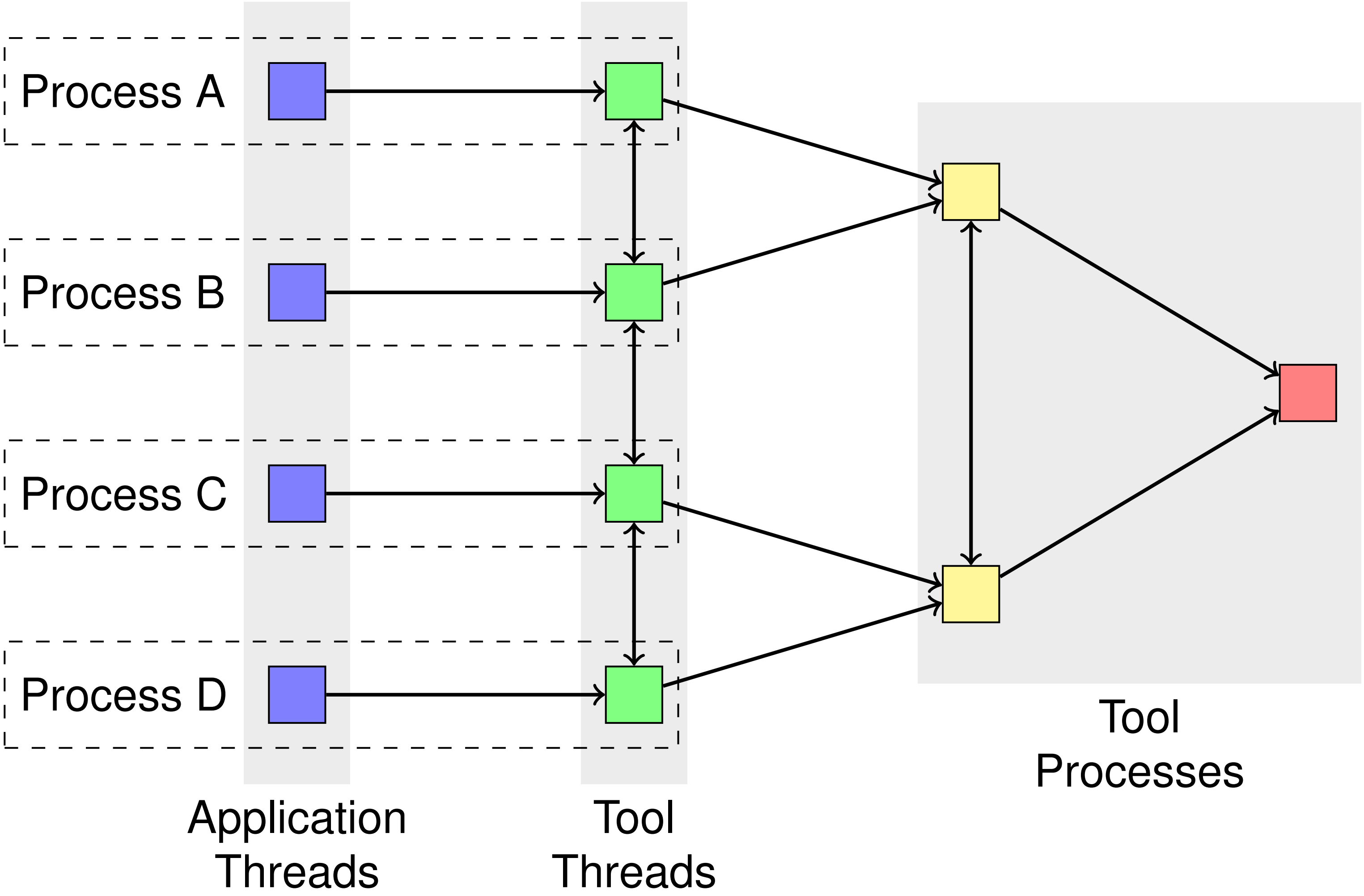


Process A (Origin)	Process B (Target)
	window location X
MPI_Barrier	MPI_Barrier
MPI_Win_lock(B)	
...	
buf = 42	
MPI_Put(&buf, B, X)	
...	print(X)
MPI_Win_unlock(B)	
MPI_Barrier	MPI_Barrier

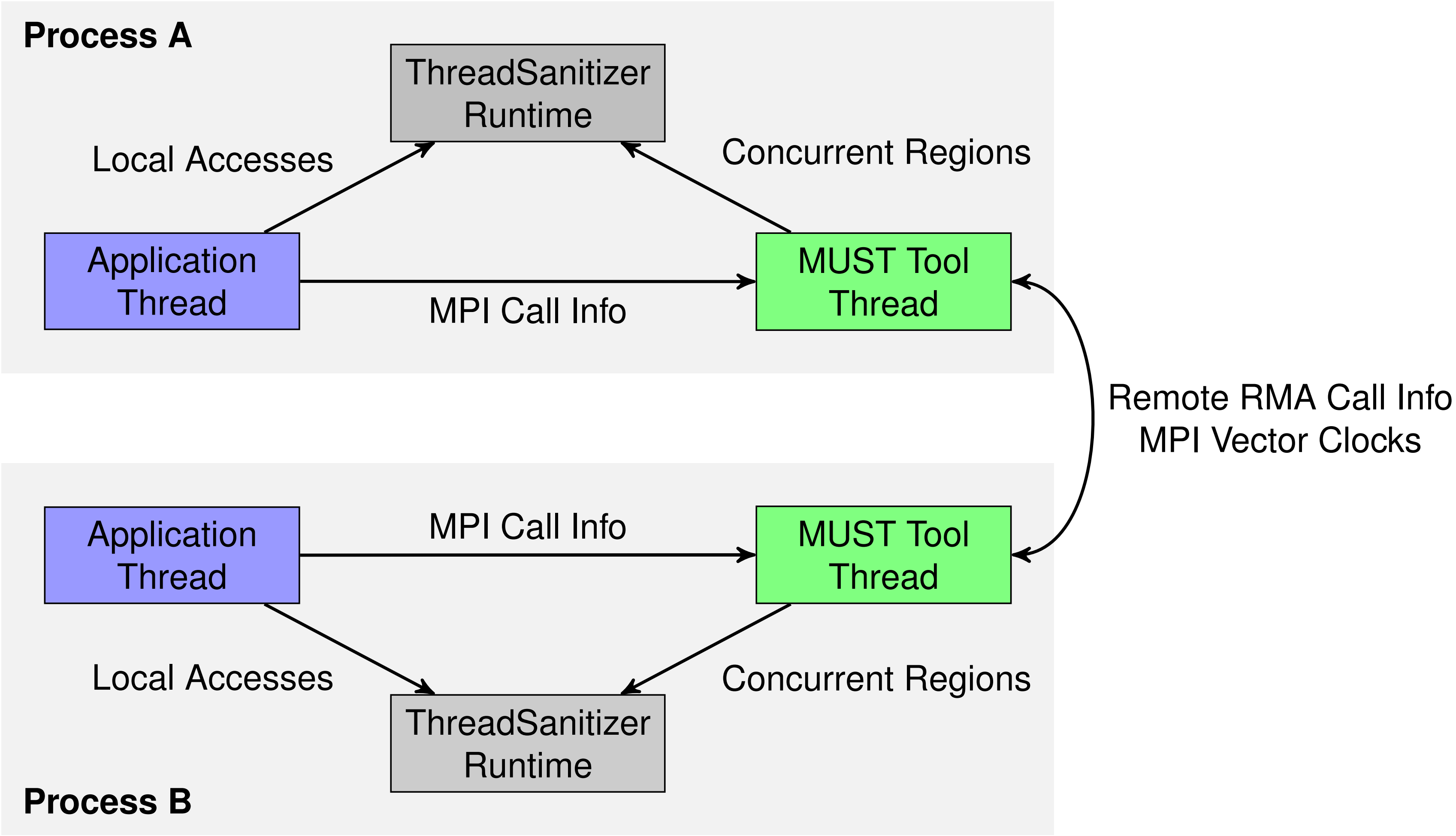
Implementation

- Tracking of MPI calls: MUST correctness checking framework
  - Capture memory and process synchronization in MPI programs
  - Determine concurrent regions of MPI RMA operations
- Actual data race detection: ThreadSanitizer
  - Runs locally on each process
  - Captures local memory accesses
  - Provides annotation API
- Idea: Annotate concurrent regions in ThreadSanitizer

MUST Infrastructure



Annotations



Process A

ThreadSanitizer Runtime

Local Accesses

Concurrent Regions

Application Thread

MUST Tool Thread

MPI Call Info

Remote RMA Call Info
MPI Vector Clocks

Application Thread

MPI Call Info

MUST Tool Thread

Local Accesses

Concurrent Regions

ThreadSanitizer Runtime

Process B

Conclusion

- MPI RMA introduces one-sided communication model
- Data races in MPI RMA as new error class
- Main concepts of on-the-fly data race detection
  - Tracking memory consistency and happens-before order
  - Identification and annotation of concurrent regions of RMA operations
- Prototype implementation in MUST and ThreadSanitizer
  - Potential of false positives (benign races)
  - Detects representatives of defined data race classes