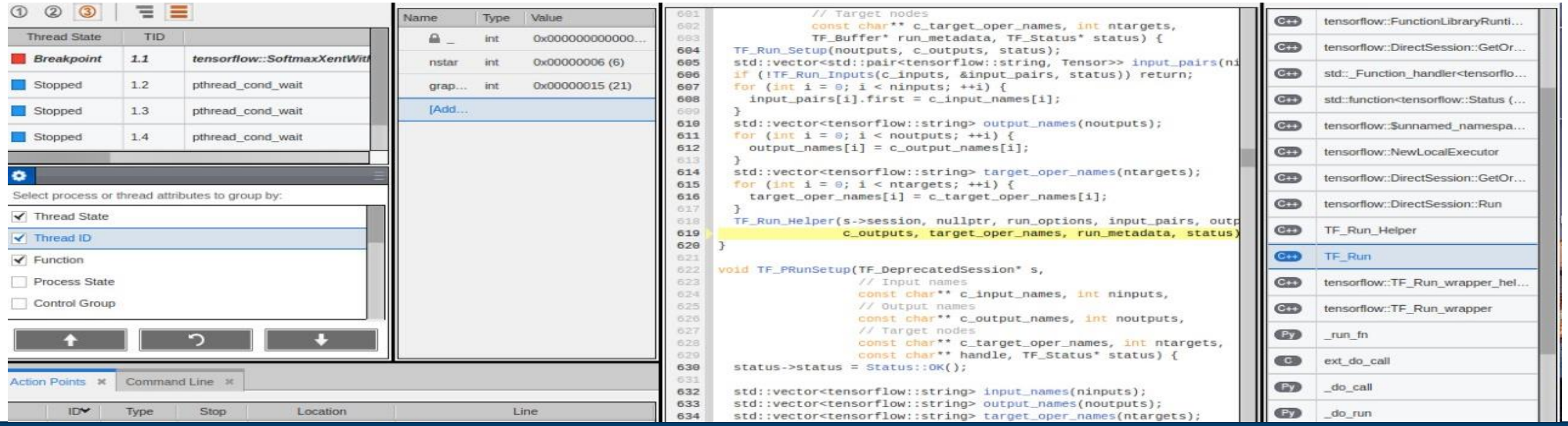
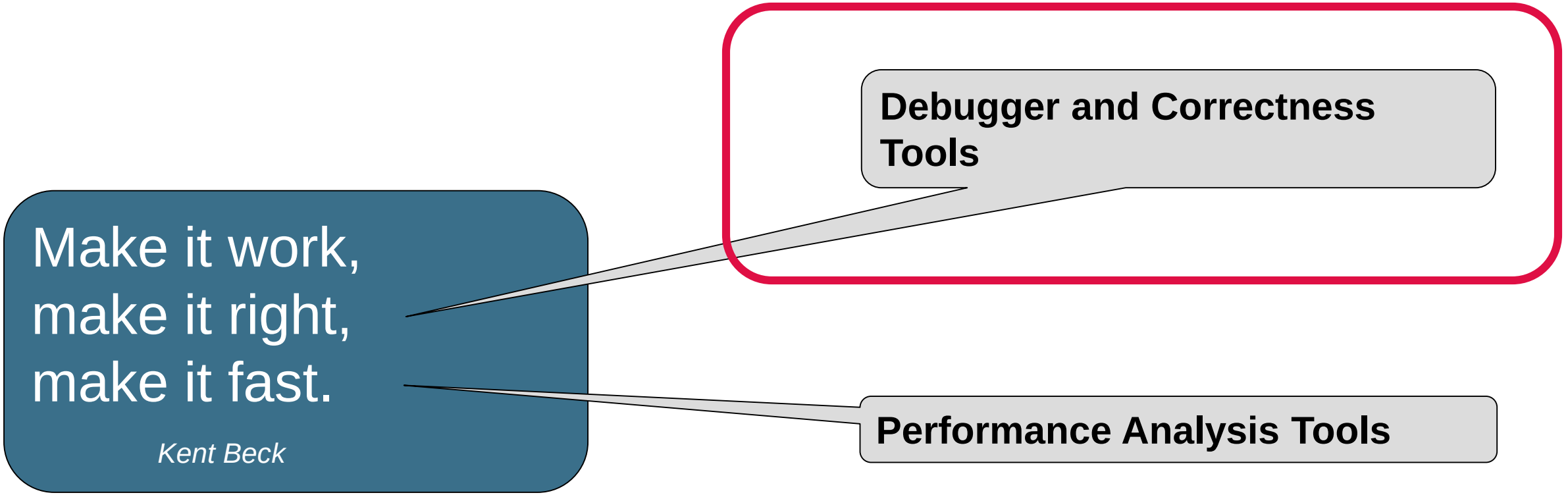


DEBUGGER & CORRECTNESS TOOLS

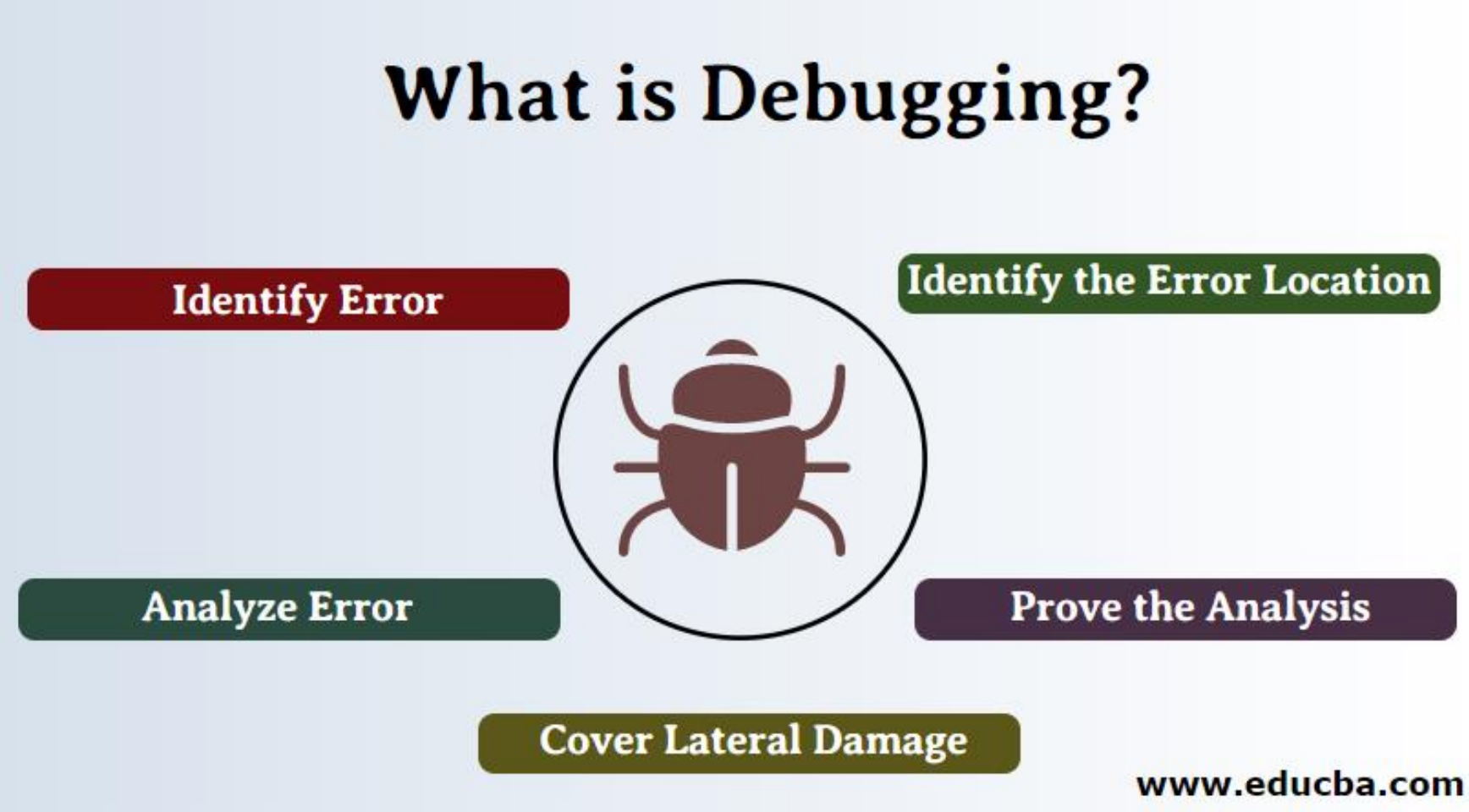
HPC SUPPORT CORNER

OCT 22, 2025 I MICHAEL KNOBLOCH I M.KNOBLOCH@FZ-JUELICH.DE

OUTLINE

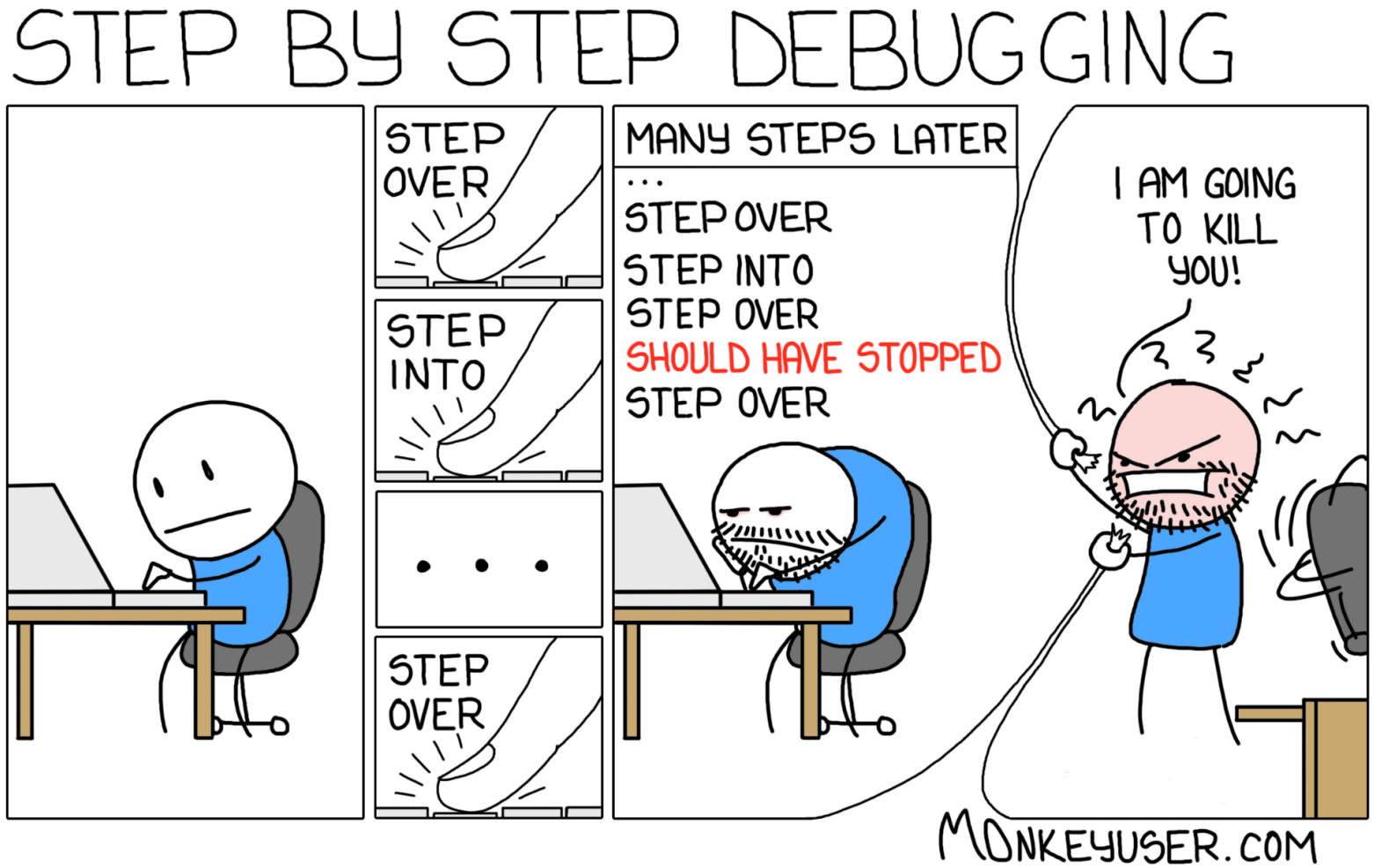

Make it work,
make it right,
make it fast.
Kent Beck
Debugger and Correctness Tools
Performance Analysis Tools

WHAT IS DEBUGGING?

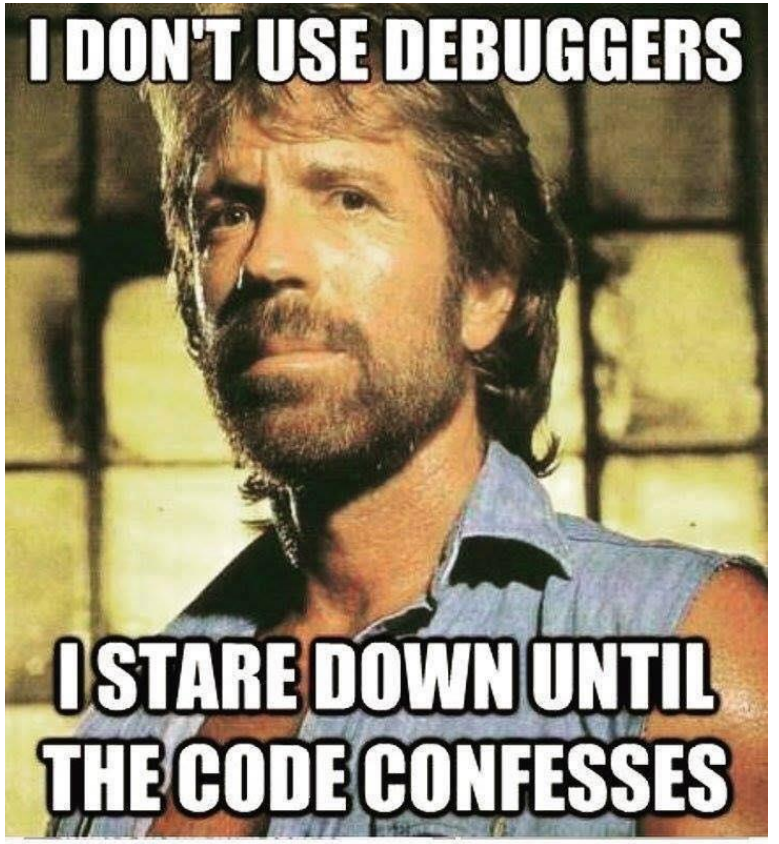


ATTENTION: DEBUGGING CAN BE TIME CONSUMING

ATTENTION: DEBUGGING CAN BE FRUSTRATING



BUT: DON‘T BE THESE GUYS

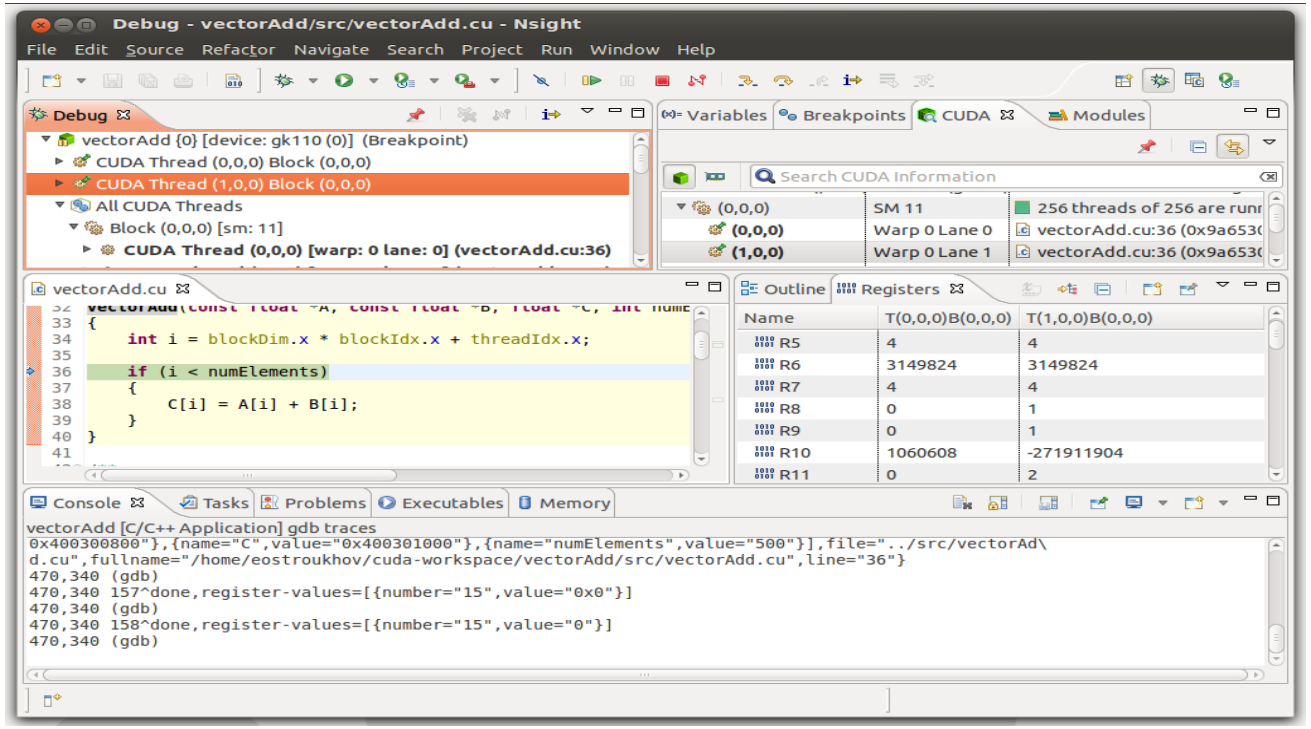


DEBUGGING TOOLS (STATUS: OCT 2025)

- **Debugger:**
  - (GDB)
  - CUDA-GDB
  - TotalView
  - LinaroForge - DDT
- **Correctness and Memory Analyzer:**
  - MUST
  - Archer
  - CUDA Compute Sanitizer

CUDA-GDB

- Part of the CUDA toolkit
- Extension to gdb
- CLI and GUI (Nsight)
- Simultaneously debug on the CPU and multiple GPUs
- Use conditional breakpoints or break automatically on every kernel launch
- Examine variables, read/write memory and registers
- Inspect GPU state when the application is suspended
- Identify memory access violations

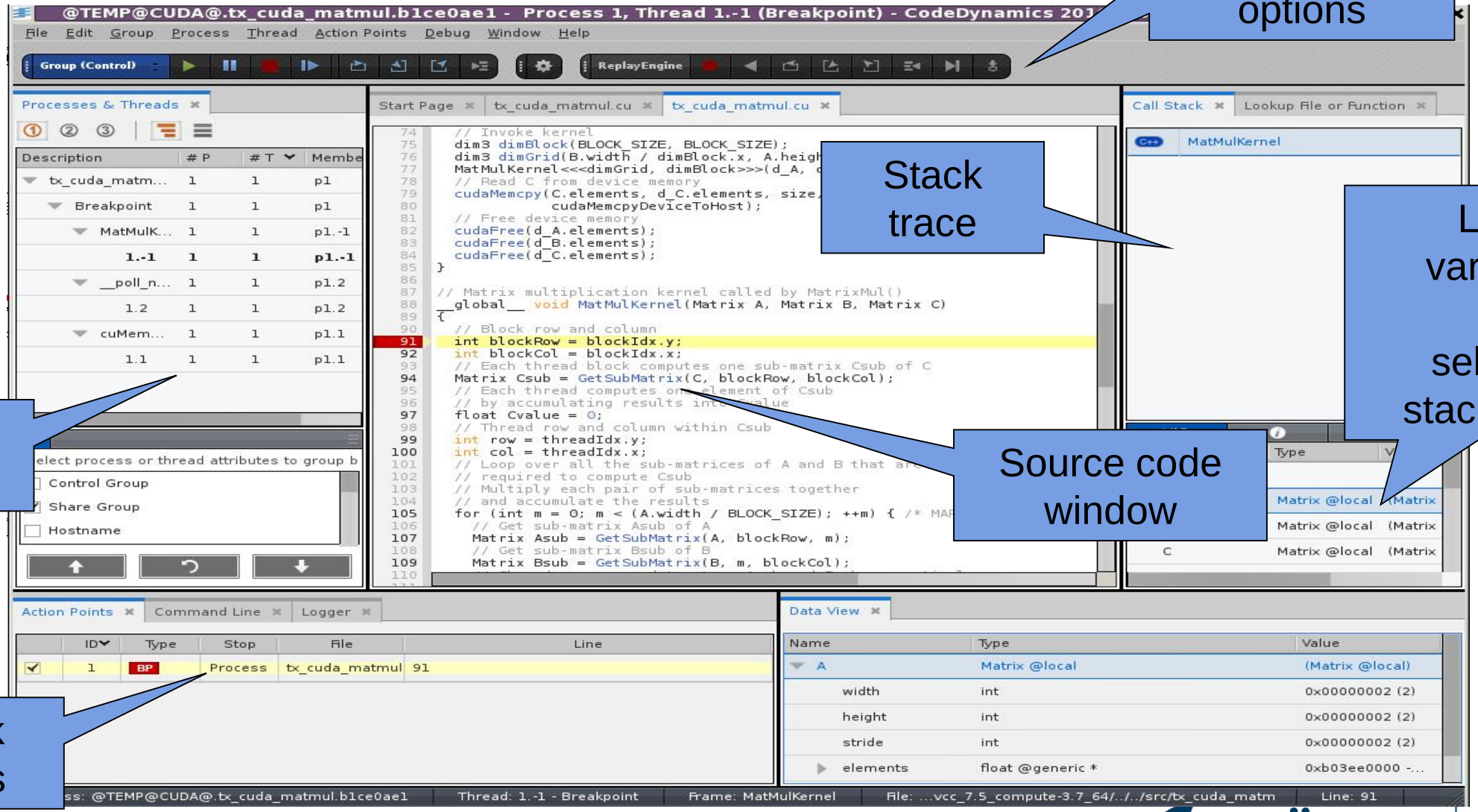


TOTALVIEW

- Symbolic Debugger for C/C++, Fortran, and mixed Python/C++ programs
- JSC's "standard" debugger
- Advanced features
  - Multi-process and multi-threaded
  - Multi-dimensional array data visualization
  - Support for parallel debugging (MPI: automatic attach, message queues, OpenMP, Pthreads)
  - CUDA and OpenACC support
  - Scripting and batch debugging
  - Advanced memory debugging
  - Reverse debugging
  - Remote debugging

- NOTE: JSC license limited to 2048 processes (shared between all users)

TOTALVIEW: MAIN WINDOW



Toolbar for common options

Stack trace

Local variables for selected stack frame

Thread control

Source code window

Break points

LINARO FORGE - DDT

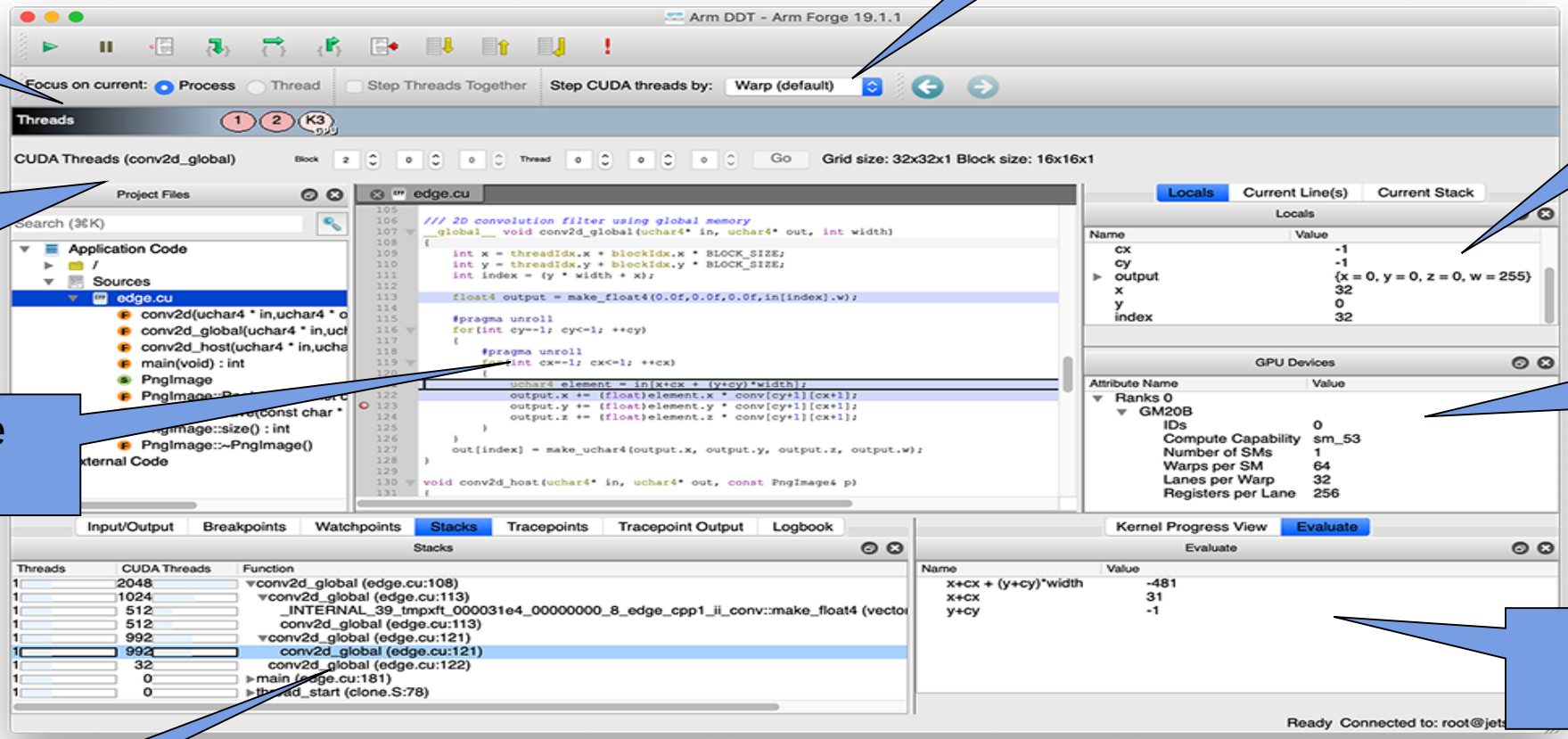
- Graphical Debugger for C/C++, Fortran, and Python programs

- Modern, easy-to-use debugger

- Advanced features
  - Multi-process and multi-threaded
  - Multi-dimesional array data visualization
  - Support for MPI parallel debugging
    (automatic attach, message queues)
  - Support for OpenMP (Version 2.x and later)
  - Support for CUDA and OpenACC
  - Job submission from within debugger

- https://linaroforge.com/linaroDdt
- NOTE: JSC license limited to 128 processes (shared between all users)

DDT: MAIN WINDOW



Process controls

CUDA Thread stepping

Variables

CUDA Thread control

GPU Device information

Source code

Expression evaluator

Stack trace

MUST

- Next generation MPI correctness and portability checker
- https://www.i12.rwth-aachen.de/go/id/nrbe

- MUST reports
  - Errors: violations of the MPI-standard
  - Warnings: unusual behavior or possible problems
  - Notes: harmless but remarkable behavior
  - Potential deadlock detection

- Usage
  - Compile with debug information (i.e. use the -g flag)
  - Run application under the control of mustrun (requires (at least) one additional MPI process)
    - E.g. on JUSUF: mustrun --must:mpiexec srun --must:np -n -n 4 ./app
  - Open output html report (might need to copy it to your local machine)

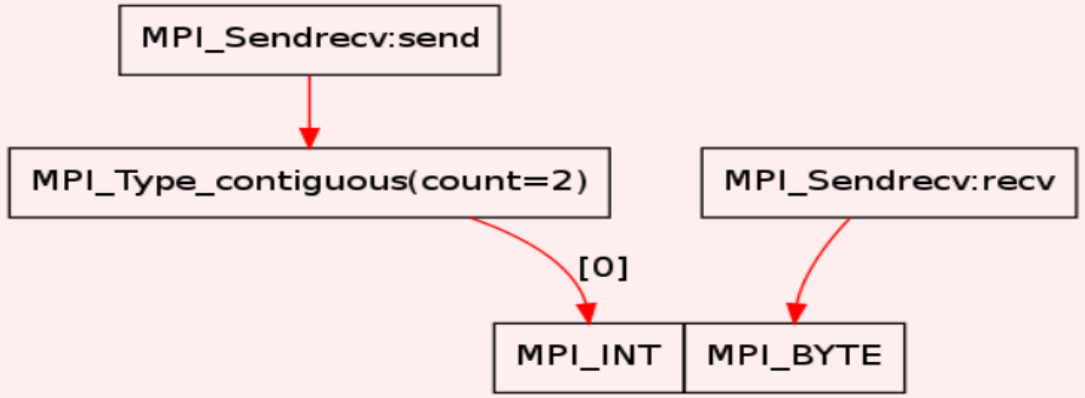
MUST DATATYPE MISMATCH

Rank	Type	Message	From	References
0	Error	A send and a receive operation use datatypes that do not match! Mismatch 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). A graphical representation of this situation is available in a detailed type mismatch view (MUST_Output-files/MUST_Typemismatch_0.html). 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** called from: #0 main@example.c:33	reference 1 rank 0: **MPI_Sendrecv** called from: #0 main@example.c:33 reference 2 rank 1: **MPI_Sendrecv** called from: #0 main@example.c:33 reference 3 rank 0: **MPI_Type_contiguous** called from: #0 main@example.c:29 reference 4 rank 0: **MPI_Type_commit** called from: #0 main@example.c:30

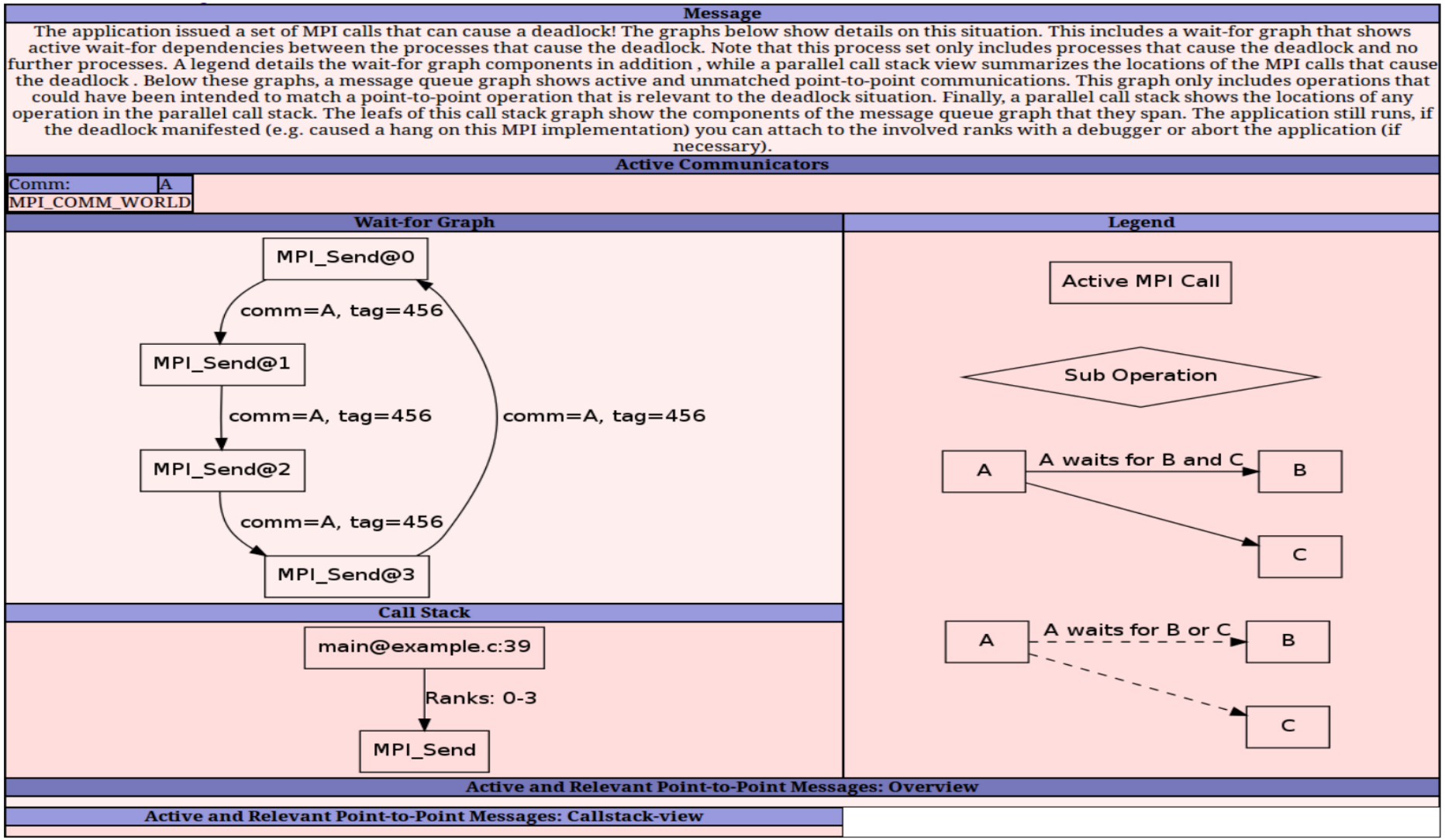
Message

The application issued a set of MPI calls that mismatch in type signatures! The graph below shows details on this situation. The first differing item of each involved communication request is highlighted.

Datatype Graph



MUST DEADLOCK DETECTION



ARCHER

- Data race detector for large OpenMP programs
- Combination of static and dynamic techniques
  - Low runtime and memory overhead
  - Still high accuracy and precision
- Now part of LLVM
- Compile with –fsanitize=thread
- Can be used with GCC, but CLANG OpenMP runtime must be linked
- Creates output in text format

ARCHER EXAMPLE

```
WARNING: ThreadSanitizer: data race (pid=2234)
  Write of size 4 at 0x7fff81d209d0 by thread T1:
    #0 .omp_outlined._debug__ /p/project/training2410/knobloch1/archer_test.c:11:10 (a.out+0xd1efb)
    #1 .omp_outlined. /p/project/training2410/knobloch1/archer_test.c:9:1 (a.out+0xd1f85)
    #2 __kmp_invoke_microtask <null> (libomp.so+0xb8782)
    #3 main /p/project/training2410/knobloch1/archer_test.c:9:1 (a.out+0xd1d55)

  Previous read of size 4 at 0x7fff81d209d0 by main thread:
    #0 .omp_outlined._debug__ /p/project/training2410/knobloch1/archer_test.c:11:12 (a.out+0xd1ed6)
    #1 .omp_outlined. /p/project/training2410/knobloch1/archer_test.c:9:1 (a.out+0xd1f85)
    #2 __kmp_invoke_microtask <null> (libomp.so+0xb8782)
    #3 main /p/project/training2410/knobloch1/archer_test.c:9:1 (a.out+0xd1d55)

  Location is stack of main thread.

  Location is global '??' at 0x7fff81d03000 ([stack]+0x1d9d0)

  Thread T1 (tid=2237, running) created by main thread at:
    #0 pthread_create /dev/shm/swmanage/jusuf/Clang/16.0.6/GCCcore-12.3.0/llvm-project-16.0.6.src/compiler-rt/lib/tsan/rtl/tsan_interceptors_posix.cpp:1048:3 (a.out+0x2678b)
    #1 __kmp_create_worker <null> (libomp.so+0x97676)

SUMMARY: ThreadSanitizer: data race /p/project/training2410/knobloch1/archer_test.c:11:10 in .omp_outlined._debug__
==================
ThreadSanitizer: reported 1 warnings
```

CUDA COMPUTE SANITIZER

- Valgrind for GPUs
- Monitors hundreds of thousands of threads running concurrently on each GPU
- Multiple Tools to detect various issues
  - Memcheck – Memory error and leak detection tool
  - Racecheck – Shared memory data access hazard detection tool
  - Initcheck – Uninitialized device global memory access detection tool
  - Synccheck – Thread synchronization hazard detection tool
- Included in the CUDA Toolkit

```
Applications  Places  System
File  Edit  View  Terminal  Help
linux64:~/demo2010$ ./ptrchecktest
unspecified launch failure : 79
linux64:~/demo2010$ cuda-memcheck ./ptrchecktest
========= CUDA-MEMCHECK
unspecified launch failure : 79
========= Invalid __global__ read of size 4
=========     at 0x00000158 in ptrchecktest.cu:27:kernel2
=========     by thread (0,0,0) in block (0,0)
=========     Address 0xfd00000001 is misaligned
=========
========= ERROR SUMMARY: 1 error
linux64:~/demo2010$ cuda-memcheck --continue ./ptrchecktest
========= CUDA-MEMCHECK
Checking...
Done
Checking...
Error: 3 (0)
Done
Checking...
Error: 1 (0)
Error: 3 (0)
Error: 5 (0)
Error: 7 (0)
Done
========= Invalid __global__ read of size 4
=========     at 0x00000158 in ptrchecktest.cu:27:kernel2
=========     by thread (0,0,0) in block (0,0)
=========     Address 0xfd00000001 is misaligned
=========
========= Invalid __global__ read of size 4
=========     at 0x00000198 in ptrchecktest.cu:18:kernel1
=========     by thread (3,0,0) in block (5,0)
=========     Address 0xfd00000028 is out of bounds
=========
========= Invalid __global__ write of size 8
=========     at 0x000001d0 in ptrchecktest.cu:38:kernel3
=========     by thread (1,0,0) in block (8,0)
=========     Address 0xfd00000204 is misaligned
=========
========= Invalid __global__ write of size 4
=========     at 0x000000f0 in ptrchecktest.cu:44:kernel4
=========     by thread (63,0,0) in block (22,0)
=========     Address 0x00000000 is out of bounds
=========
========= ERROR SUMMARY: 4 errors
```

DEBUGGING RECOMMENDATIONS

- Always debug at the lowest possible scale!

- GPU Applications:
  - Single Node / Workstation: Use CUDA-GDB
  - Multi-Node / Supercomputer: Use TotalView/DDT

- MPI Applications:
  - Check with MUST at least once
  - Use TotalView/DDT at small scale (if error occurs there), else attach to as few processes as neccessary

NEED HELP?

- Talk to the experts
  - Use local 1st-level support, e.g. SC support, SimLab
  - JSC/NVIDIA Application Lab
    - When porting/debugging on GPUs
  - ATML Parallel Performance
    - Provides tools
  - ATML Application Optimization and User Service Tools
    - EPICURE

QUESTIONS