Efficient Parallel Debugging for MPI, Threads, and Beyond

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5], Ganesh Gopalakrishnan[1], Mark O'Connor[3], Joachim Protze[4] and Simone Atzeni[1]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Booth #2143

Booth #1508

Booth #3624

Booth #1030

Organization

- Introduction and Tools
- Hands-On I

Coffee Break

- The Tao of Debugging
- Systematic Debugging

Lunch Break

- Defects in Practice
- MPI Applications
- Threaded Applications

Coffee Break

- Hands-On II
- Beyond MPI and Threads
- Discussion and Wrap-Up

Introduction and Tools

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], Joachim Protze[4] and Simone Atzeni[1]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Organization

- **Introduction and Tools**
- Hands-On I

Coffee Break

- The Tao of Debugging
- Systematic Debugging

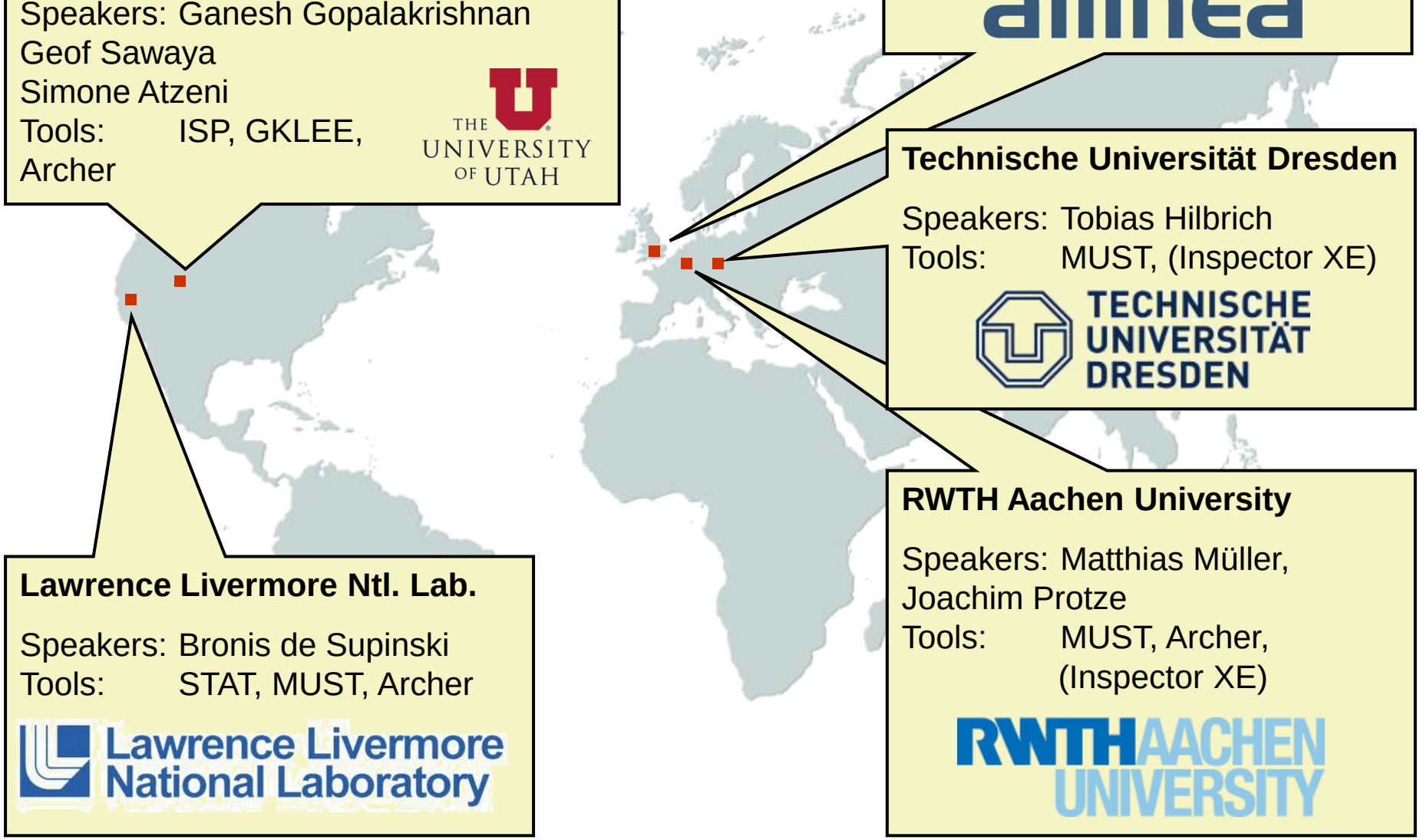
Lunch Break

- Defects in Practice
- MPI Applications
- Threaded Applications

Coffee Break

- Hands-On II
- Beyond MPI and Threads
- Discussion and Wrap-Up

Introduction and Tools
About us



University of Utah

Speakers: Ganesh Gopalakrishnan
Geof Sawaya
Simone Atzeni
Tools: ISP, GKLEE, Archer

THE UNIVERSITY OF UTAH

Allinea Software

Speakers: David Lecomber
Mark O'Connor
Tools: Allinea DDT

allinea

Technische Universität Dresden

Speakers: Tobias Hilbrich
Tools: MUST, (Inspector XE)

TECHNISCHE UNIVERSITÄT DRESDEN

Lawrence Livermore Ntl. Lab.

Speakers: Bronis de Supinski
Tools: STAT, MUST, Archer

Lawrence Livermore National Laboratory

RWTH Aachen University

Speakers: Matthias Müller,
Joachim Protze
Tools: MUST, Archer,
(Inspector XE)

RWTH AACHEN UNIVERSITY

Introduction and Tools
Correctness, Tools, Techniques – Why are we here?

"A hang only appeared when pf3d is scaled to half a million processes.
The user refused to debug for 6 months …"
—Dong Ahn, SC'13 BOF

„The scientific community places more faith in computation than is justified."
—Darrel C. Ince, Leslie Hatton, and John Graham-Cumming in Nature Perspectives

Its about time:
Time spent debugging is time subtracted from your research

„When knowledge is embedded in a tool, it frees the practitioner from the need to master that particular knowledge, it changes the skills needed, and it opens the way for development of new knowledge."
—Michael L. Van de Vanter, Douglass E. Post, and Mary E. Zosel

"Proceed systematically. Rather than observing values at random, let your search be guided by scientific method."
—Andreas Zeller, in Why Programs Fail

Introduction and Tools
What we cover today

HPC Debugging Tools

- Allinea DDT
  ⇒ A parallel debugger
- MUST
  ⇒ Automatic MPI checks
- ISP
  ⇒ MPI non-determinism coverage
- STAT
  ⇒ Scalable outlier detection
- Archer
  ⇒ Threading races / deadlocks
- Intel Inspector
  ⇒ Automatic thread checks
- GKLEE:
  ⇒ GPU race checks

Helpers, Techniques, and Good Practices for (Parallel) Debugging

- Debugging with intuition and tools “magic”
- We touch upon general debugging techniques:
  ⇒ Bisect
  ⇒ Version control
  ⇒ Bugtrackers
  ⇒ Logbooks
- Terms, non-determinism, parallelism
- Scientific debugging

Use Cases, Intuition, Experience

- Hands-on with the tools
- Example use cases
- In-depth MPI and OpenMP specifics:
  ⇒ Common sources for errors
  ⇒ Tool capabilities

Introduction and Tools
More Debugging on Monday

- More debugging …
  - Monday 8:30pm-5pm, room “250-A”
  - SC Tutorial: *“Debugging and Performance Analysis on Native and Offload HPC Architectures.”*
  - In-depth exploration of parallel debugging and optimization focused on techniques that can be used with accelerators and coprocessors

Introduction and Tools
Why tools?

Tools can handle complexities that are "very" hard to handle manually

"A hang only appeared when pf3d is scaled to half a million processes.
The user refused to debug for 6 month …"
—Dong Ahn, SC'13 BOF

Tools can speed up time to solution;

Sometimes they can work magic for you

Its about time:
Time spent debugging is time subtracted from your research

„When knowledge is embedded in a tool, it frees the practitioner from the need to master that particular knowledge, it changes the skills needed, and it opens the way for development of new knowledge."
—Michael L. Van de Vanter, Douglass E. Post, and Mary E. Zosel

Tools can understand details you might not want to be concerned (reminded) about

Introduction and Tools
Tools for Today

Allinea DDT
- Paradigms: MPI, OpenMP, CUDA, …
- Goal: Scalable parallel debugger
- License: Commercial (Demo available)

MUST
- Paradigms: MPI (OpenMP in progress)
- Goal: Automatically Detect MPI Usage Errors
- License: OpenSource

Archer
- Paradigms: OpenMP, pthreads
- Goal: Detect data races
- License: OpenSource

STAT
- Paradigms: MPI, OpenMP, pthreads, …
- Goal: Gather and merge stack traces to identify outliers
- License: OpenSource

Introduction and Tools: Complementary Tools (briefly covered in the afternoon)

Intel Inspector XE

- Paradigms: OpenMP, pthreads
- Goal: Detect data races, memory errors, and deadlocks
- License: Commercial (Demo available)

ISP

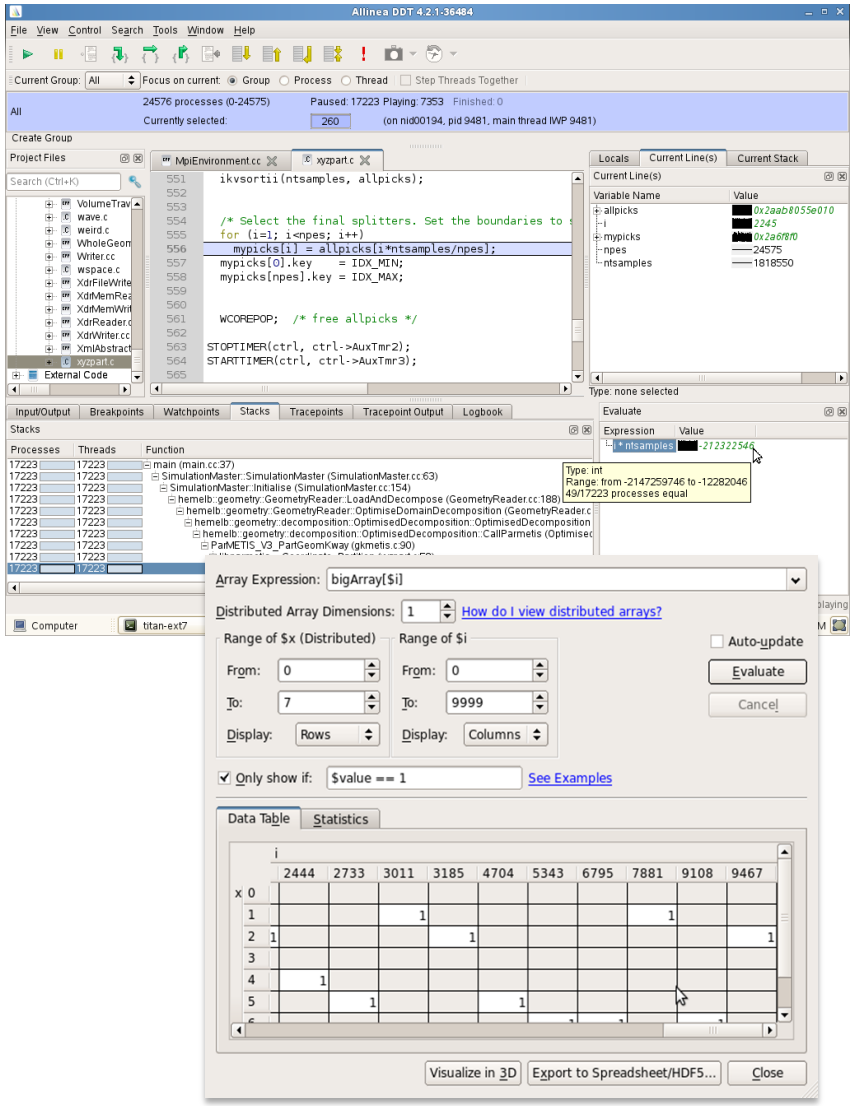
- Paradigms: MPI
- Goal: Detect MPI usage errors, while exploring alternative interleavings
- License: OpenSource

GKLEE

- Paradigms: CUDA
- Goal: Data Race Checking for C/C++ CUDA applications
- License: OpenSource

Introduction and Tools
Tutorial Tools – Allinea DDT

- Graphical debugger for:
  - Everything in HPC
    - OpenMP and pthreads, MPI, UPC, SHMEM, CUDA, OpenACC
    - Intel Xeon, Xeon Phi, ARM, POWER
  - Automatic problem analysis and diagnosis
    - Memory misuse and leaks
    - Data analysis across processes
    - Static analysis of code
  - Observing and controlling program
- Part of Allinea Forge tool suite
  - Edit, build, profile, commit



Introduction and Tools
Tutorial Tools – Allinea DDT, Usage

- Compile + link application with “-g”
- Option 1: Start application from debugger
  - Start GUI: “ddt myApp.exe”
  - Configure and start application run
- Option 2: Attach to running application
  - Start application
  - Start GUI: “ddt myApp.exe”
  - Select attach, select the respective processes
- Once Allinea DDT is attached:
  - Control the progress of the application
  - Investigate application state, memory usage, …

Introduction and Tools
Tutorial Tools – MUST

- MPI runtime error detection
- Goal: Report MPI usage errors to the user
- Open source (BSD license)
- Key strengths:
  - Distributed (and scalable) correctness checks
  - Detailed investigation of collectives, communication buffer use, and MPI datatypes
  - Deadlock detection
  - Nonblocking collectives
- Use of infrastructure for scalability + extensibility

Introduction and Tools
Tutorial Tools – MUST, Basic Usage

- Compile and link application with "-g"
  - If necessary use the shared version of the MPI library
- Replace "mpiexec" with "mustrun"
  - E.g.: mustrun –np 4 myApp.exe
- Inspect "MUST_Output.html" in the run directory
- Different MUST configurations possible, default:
  - MUST uses an extra process for non-local checks (Invisible to application)
  - I.e.: "mustrun –np 4 …" will need resources for 5 tasks
  - Make sure to allocate the extra task in batch jobs

Introduction and Tools
Tutorial Tools – MUST, Output Example

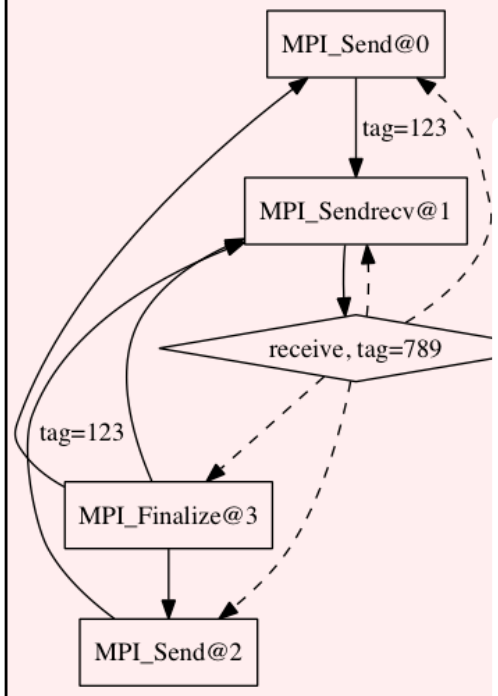
MUST Outputfile

MUST Deadlock Details, date: Tue Mar 20 13:42:42 2012.

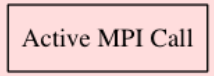
Back to MUST error report

Message
The application issued a set of MPI calls that can cause a deadlock! The graphs below show details on this situation. This includes a wait-for graph that shows active waiting dependencies in the deadlock situation and a legend for this graph . The application still runs, if the deadlock manifested (e.g. caused a hang on this MPI implementation) you can attach to the involved ranks with a debugger or abort the application (if necessary).

Wait-for Graph



Legend



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	

Introduction and Tools
Tutorial Tools – Archer

- Archer is a data race detector for OpenMP programs
- Archer combines static and dynamic techniques
- Lower overhead and higher precision than other tools.
- Successfully found races in large applications (>>10k LOC)
- Open source (BSD-like license)

Introduction and Tools
Tutorial Tools – Archer, Basic Usage

- Compile your code with Archer :
  - % clang-archer example.c -g -fopenmp -o example
- Execute your code:
  - % export OMP_NUM_THREADS=…
  - % ./example
- Error report appears on stderr

Introduction and Tools
Tutorial Tools – Archer

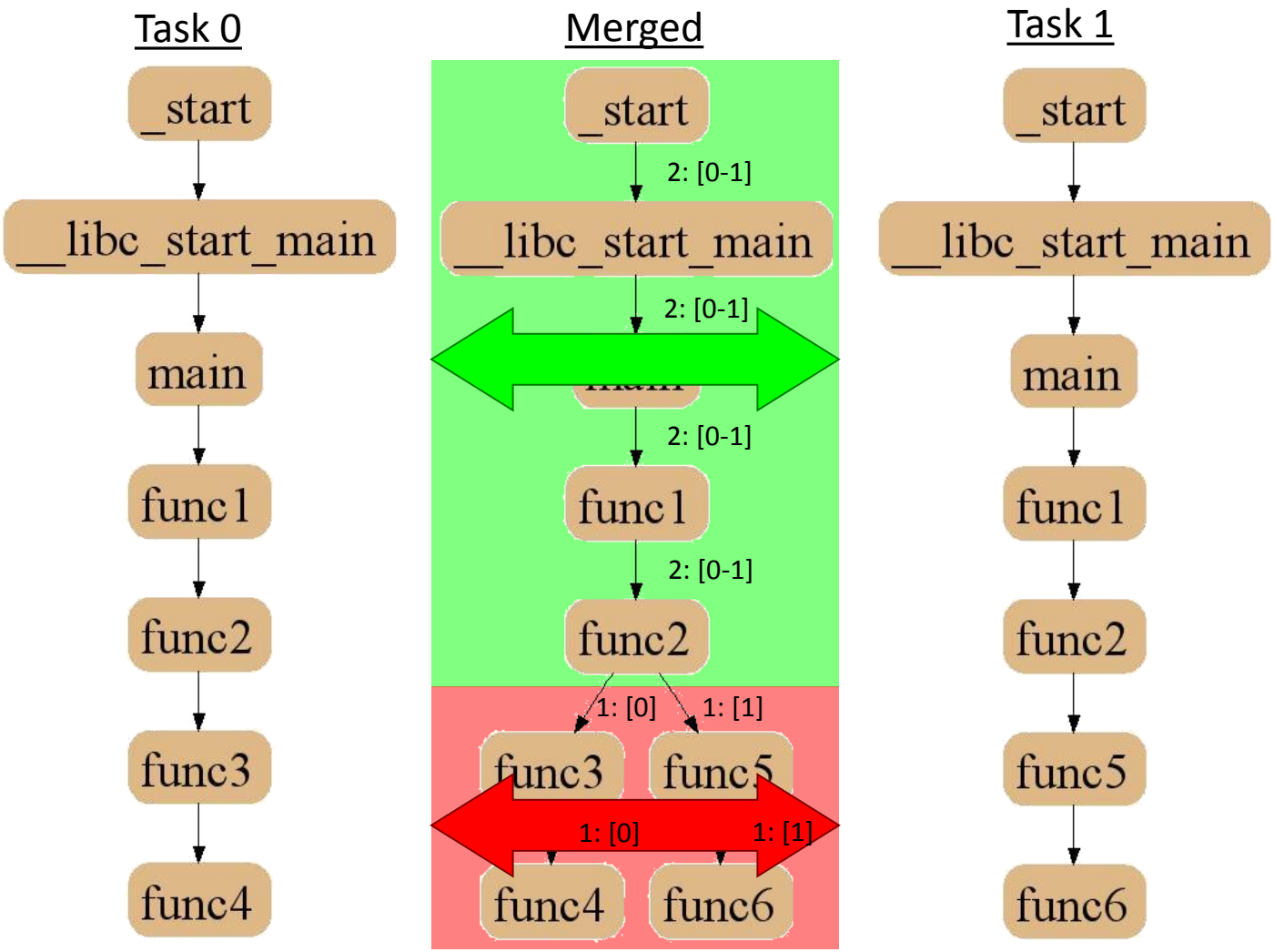
```
==================
WARNING: ThreadSanitizer: data race (pid=1356)
  Write of size 8 at 0x7ffc74026c70 by main thread:
    #0 .omp_microtask. red-race.c:16:5 (a.out+0x0000000c030f)
    #1 __kmp_invoke_microtask <null>:0 (libiomp5.so+0x000000067aa2)
    #2 __libc_start_main <null>:0 (libc.so.6+0x003f8881ed5c)

  Previous write of size 8 at 0x7ffc74026c70 by thread T4:
    #0 .omp_microtask. red-race.c:16:5 (a.out+0x0000000c030f)
    #1 __kmp_invoke_microtask <null>:0 (libiomp5.so+0x000000067aa2)

  Location is stack of main thread.

  Thread T4 (tid=1361, running) created by main thread at:
    #0 pthread_create $TSAN/llvm/projects/compiler-
rt/lib/tsan/rtl/tsan_interceptors.cc:848:3 (a.out+0x00000005c451)
    #1 __kmp_create_worker $TSAN/libomp_oss/src/z_Linux_util.c:1047:22
(libiomp5.so+0x0000000687b8)
    #2 __libc_start_main <null>:0 (libc.so.6+0x003f8881ed5c)

SUMMARY: ThreadSanitizer: data race red-race.c:16 .omp_microtask.
==================
```

Introduction and Tools
Tutorial Tools – STAT

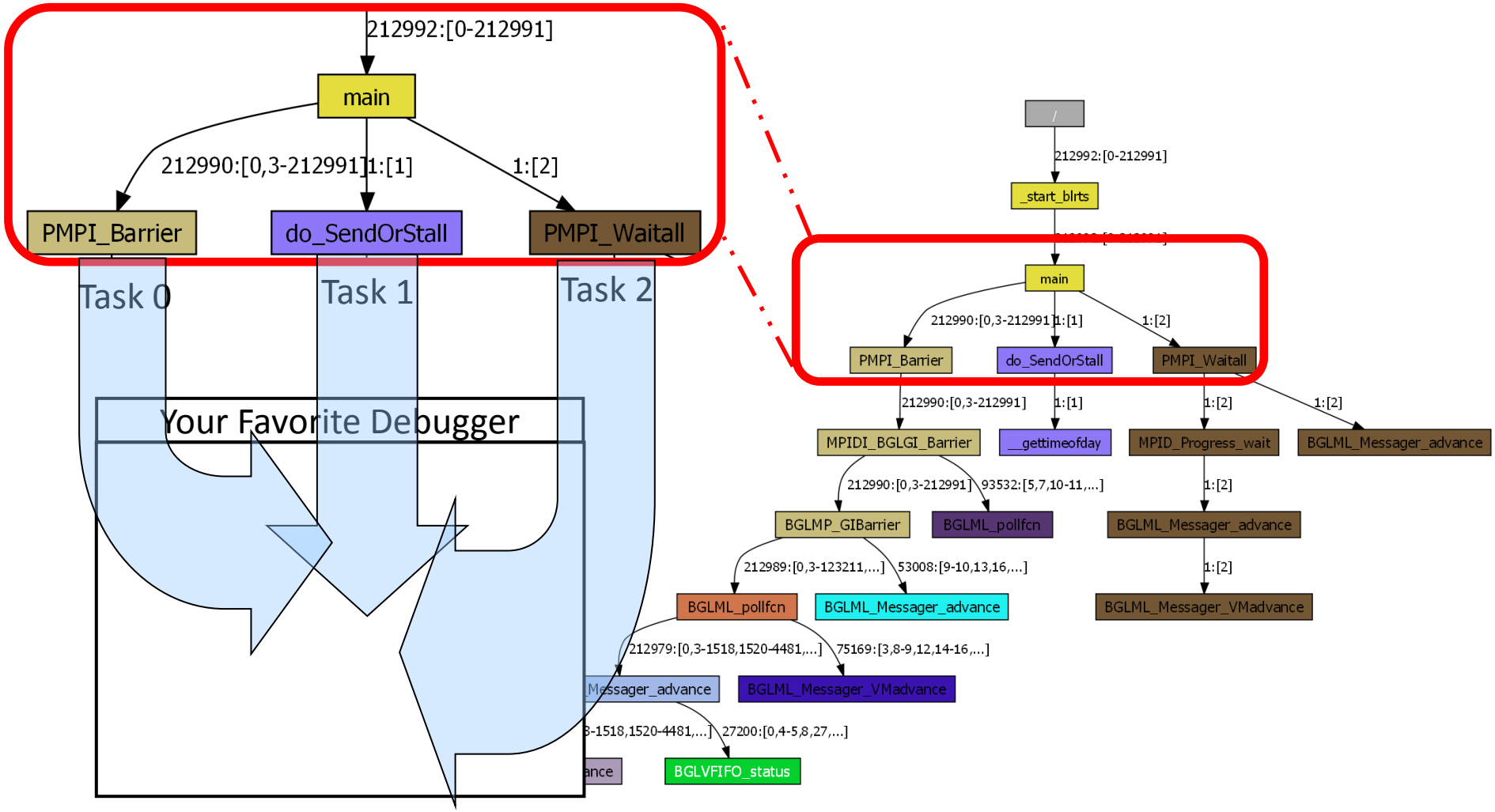
- Traditional debugging paradigm does not scale well.
  - Huge quantities of symbol table and process state information
  - Too much information to present to the user

- First need to reduce search space to manageable subset
- Apply traditional debugging techniques on subset

Introduction and Tools
Tutorial Tools – The Basis for STAT



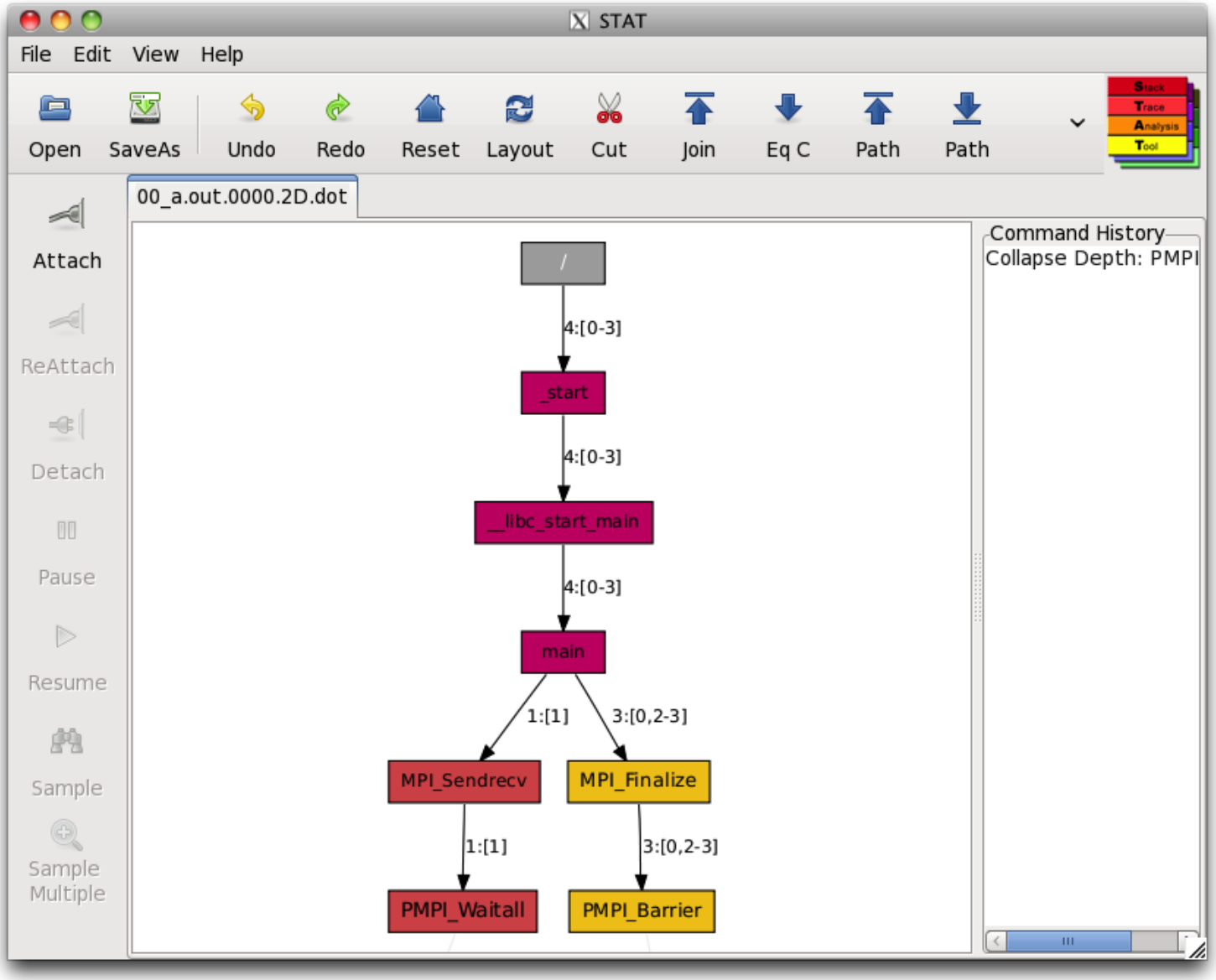
STAT Compliments Traditional Parallel Debuggers



Tutorial Tools – Basic Usage of STAT

- Compile + link application with "-g"
- Start application "mpirun -np X myApp.exe"
- Start STAT
  - stat-gui
  - Attach to the running mpirun process

Introduction and Tools
Tutorial Tools – STAT, GUI for a Simple Deadlock

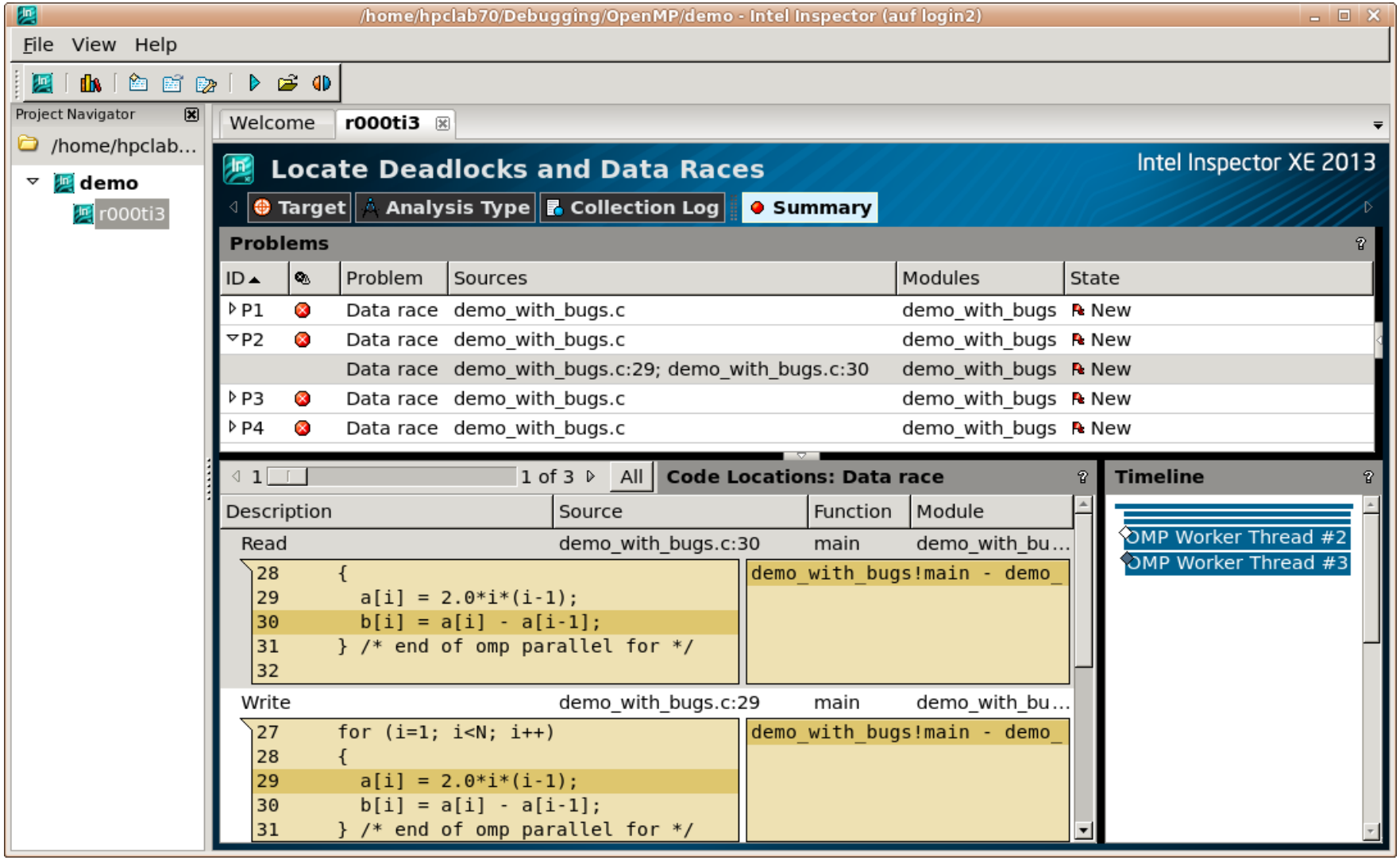


Introduction and Tools
Complementary Tutorial Tools – Intel Inspector XE

- Automatic error detection for:
  - Memory related errors (e.g., leaks)
  - Threading errors (OpenMP, Pthreads, Intel TBB)

    ⇒Data races and deadlocks
- Available for Linux and Windows
- Support for: C, C++, C#, F#, Fortran
- Graphical user interface
- Successor of Intel Thread Checker
- Commercial License

Introduction and Tools
Complementary Tutorial Tools – Intel Inspector XE GUI

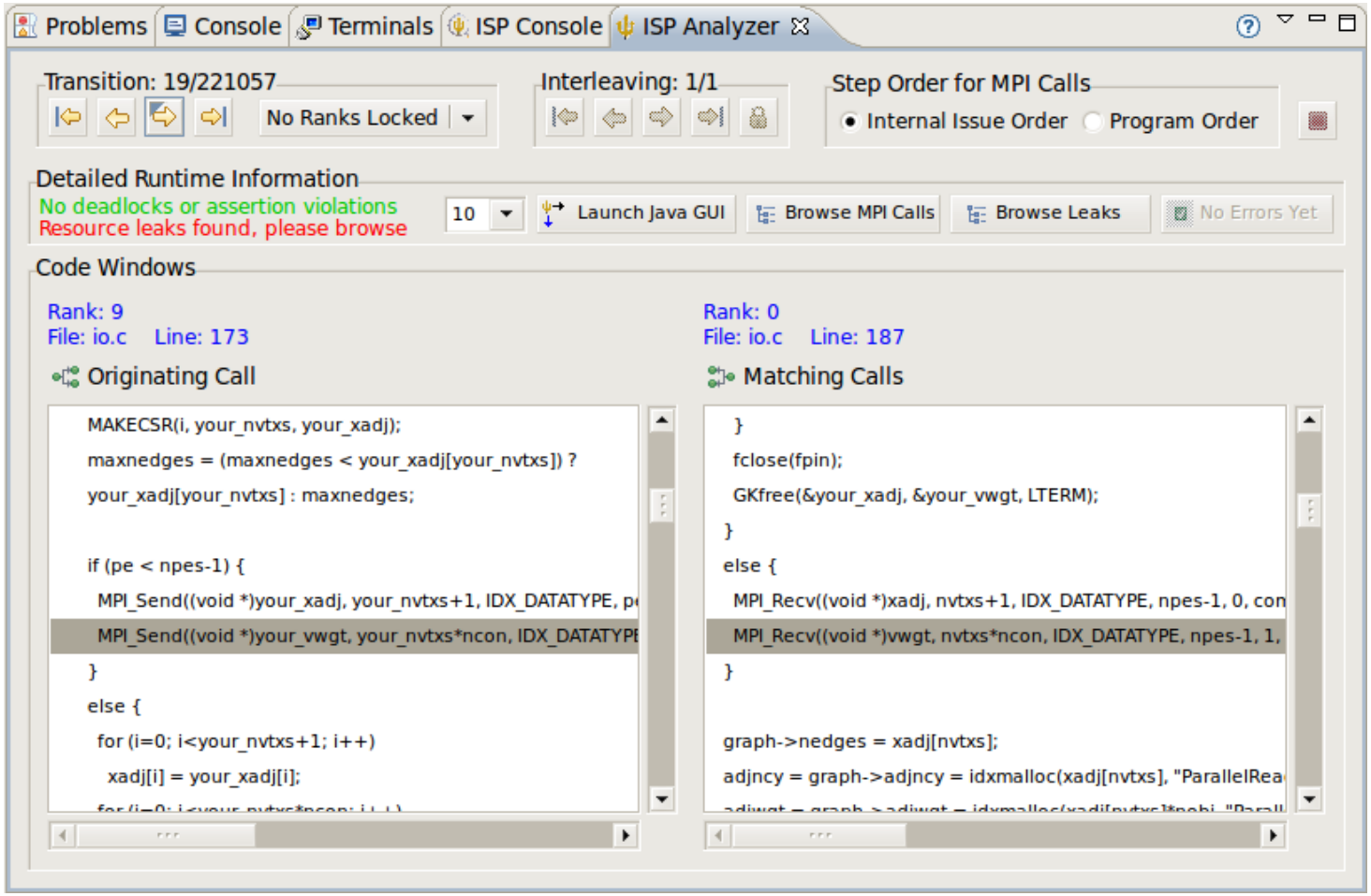


Introduction and Tools
Complementary Tutorial Tools – ISP

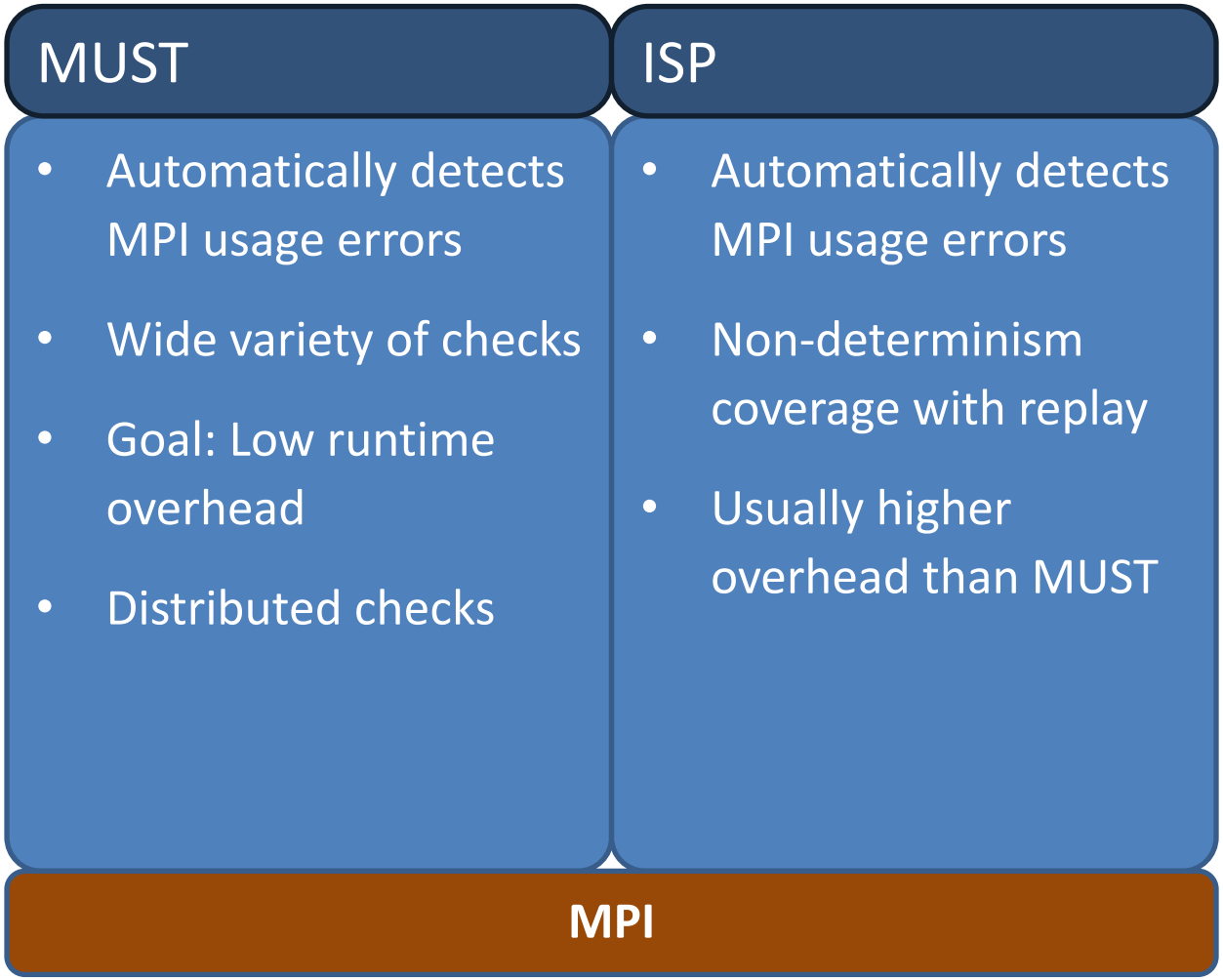
- ISP (In-situ Partial Order) detects MPI usage errors
- Replay based deadlock detection:
  - Forces non-determinism coverage
  - Eliminates redundant tests

  $\Rightarrow$Detects rare deadlocks (Even ones never manifested)
- Reports errors with:
  - Java based GUI or
  - GEM an Eclipse PTP component
- DAMPI: distributed and scalable version of ISP

Introduction and Tools
Complementary Tutorial Tools – ISP, GUI (GEM)



Introduction and Tool Overview – Tutorial Tools Summary (1)

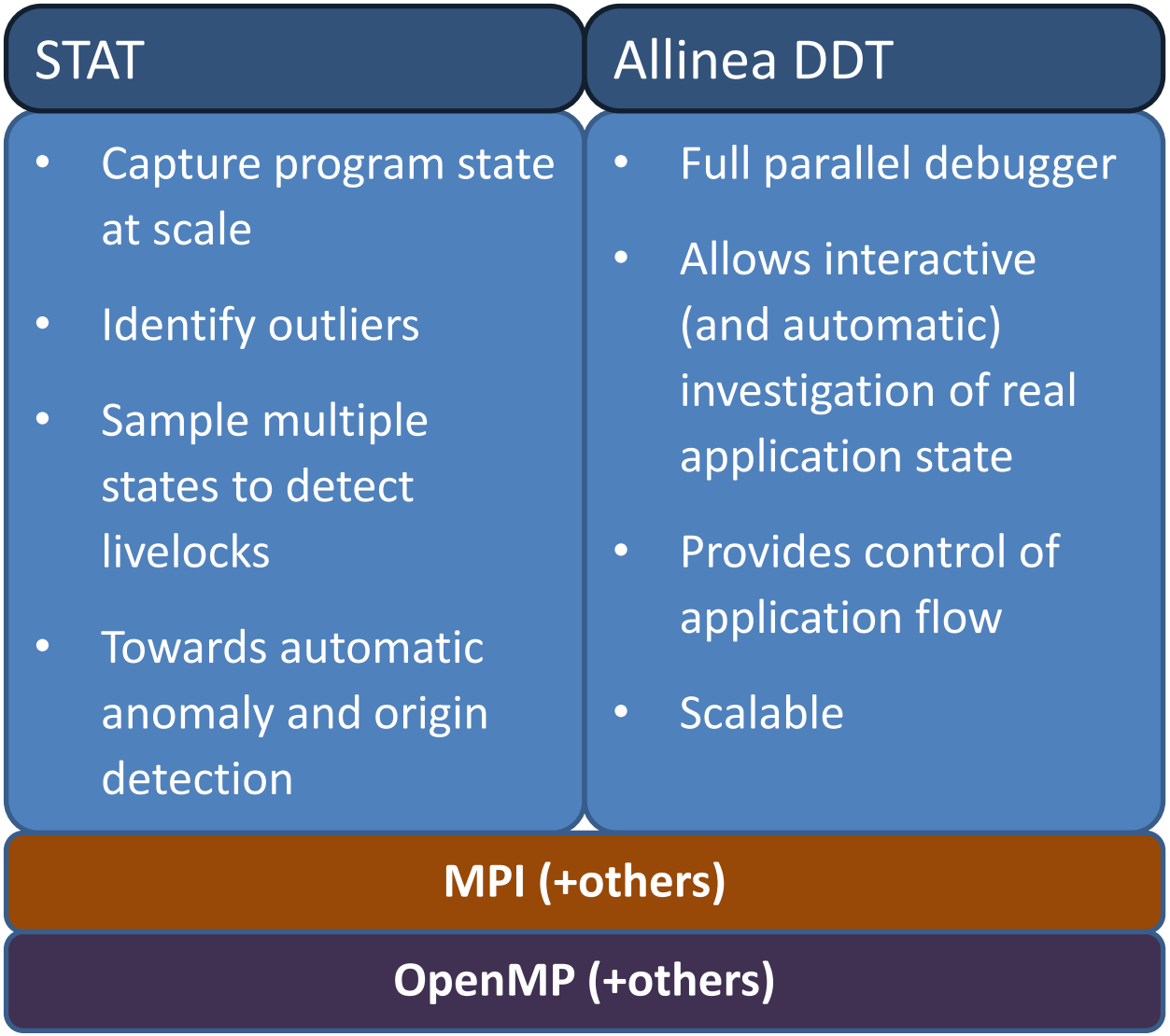


Introduction and Tool Overview – Tutorial Tools Summary (2)

Intel Inspector XE	Archer
• Investigation of threading errors ⇒ Data races ⇒ Deadlocks • Automatic, usually high overhead	• Investigation of threading races ⇒ Data races • Lower overhead than Intel Intel Inspector XE • High memory overhead

OpenMP (+others, e.g., pthreads)

Introduction and Tool Overview – Tutorial Tools Summary (3)



Introduction and Tool Overview – Tutorial Tools Summary (4)

MUST	ISP
Build: No modification ("-g" for MUST+Dyninst) *Run*: Replace "mpiexec" with "**mustrun**" (In batch jobs, allocate resources for 1 extra task) *Result*: Investigate "MUST_Output.html"	*Build*: Use compiler wrapper, e.g., replace "mpicc" with "**ispcc**" *Run*: Replace "mpiexec" with "**isp –l trace**" *Result*: Use ISP GUI: "**ispUI trace**"

Introduction and Tool Overview – Tutorial Tools Summary (5)

Intel Inspector XE

Build: Add "**-g**" to compiler flags

Usually also: "–O0"

Run:

- Use GUI: "**inspxe-gui**"
- Setup experiment
- Run program
- Investigate

Archer

Build: Use compiler wrapper: **"clang-archer"**

Run:

- Run program
- Find error report on stderr

Introduction and Tool Overview – Tutorial Tools Summary (6)

STAT	Allinea DDT
Build: Add "**-g**" to compiler flags *Run*: No modification *Attach*: Attach to the initial "mpirun" process with "**stat-cl**" *Result*: Investigate with "**stat-view**"	*Build*: Add "**-g**" to compiler flags *Run*: – Use DDT GUI: "**ddt**" – Control run with DDT – Investigate

Hands-On I

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], and Joachim Protze[4]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Booth #2143

Booth #1508

Booth #3624

Booth #1030

Organization

- Introduction and Tools
- **Hands-On I**

Coffee Break

- The Tao of Debugging
- Systematic Debugging

Lunch Break

- Defects in Practice
- MPI Applications
- Threaded Applications

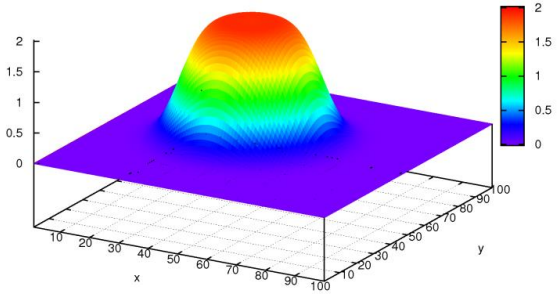
Coffee Break

- Hands-On II
- Beyond MPI and Threads
- Discussion and Wrap-Up

Hands-On I Content

1) An example

- To debug something “a touch real” we use:
  ⇒ 2D Heat Equation
- We provide a little example program called “heat”
- Available in:
  ⇒ MPI / OpenMP,
  ⇒ C / Fortran90



2) Connect to a Virtual Machine with all the tools

- For each participant we provide a 4 core virtual machine
- All tools and examples installed
- Connection via SSH or VNC (Browser or client)

amazon web services™

3) Let the tools detect some errors

- Initial examples highlight the use of each tool
- Tools point you almost directly onto the bugs we introduced for you

Afterwards:

- Reality:
  ⇒ Which tool do I use?
  ⇒ Often tools don’t put your nose to the right spot

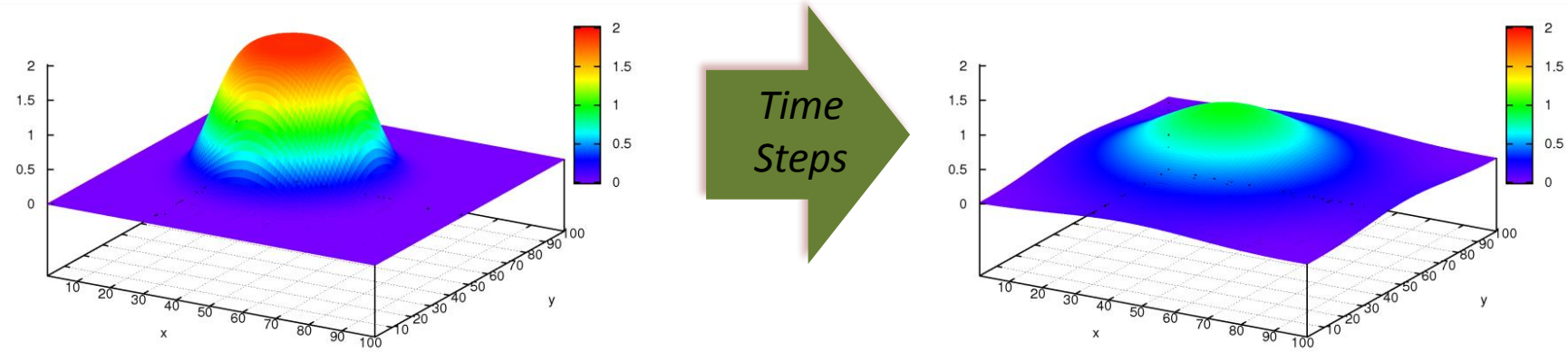
⇒ Thus, Systematic debugging

Hands-On I
The Heat Conduction Example – Overview

- Heat equation solver as debugging example
- Heat equation describes heat distribution in a region over time
- Equation (2D): $\frac{\partial u}{\partial t} = k\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$
- Example implements solution on a 2D grid
- Time step for a grid cell (x,y):

$$\frac{\Delta u[x,y]}{\Delta t} = k\left(\frac{u[x-1,y]+u[x+1,y]-2u[x,y]}{\Delta x^2} + \frac{u[x,y-1]+u[x,y+1]-2u[x,y]}{\Delta y^2}\right)$$

- Visualization as 3D chart:

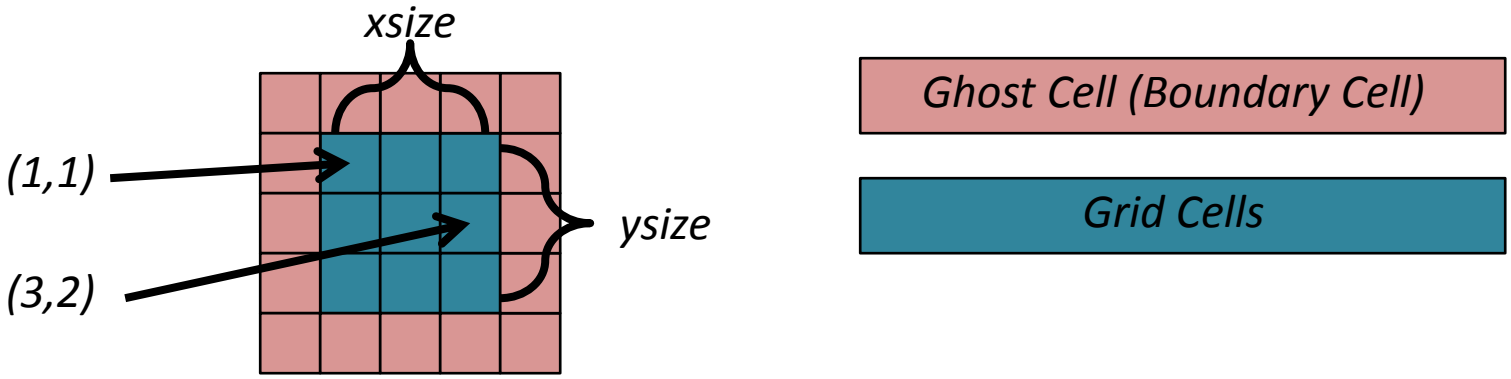


Hands-On I
The Heat Conduction Example – Source Code

- Available in C + Fortran90
- Parallel versions with MPI + OpenMP
- Key object is the structure for the 2D grid
- Functions:
  - heatAllocate & heatDeallocate – Creates/Frees the grid
  - heatInitialize & heatInitFunc – Sets the initial heat distribution
  - heatPrint – Prints the grid to stdout
  - heatTimestep – Calculates one timestep for the full grid
  - heatBoundary – Exchanges boundary data
  - heatTotalEnergy – Calculates overall energy amount
  - Main function – Contains main loop

Hands-On I
The Heat Conduction Example – 2D Grid

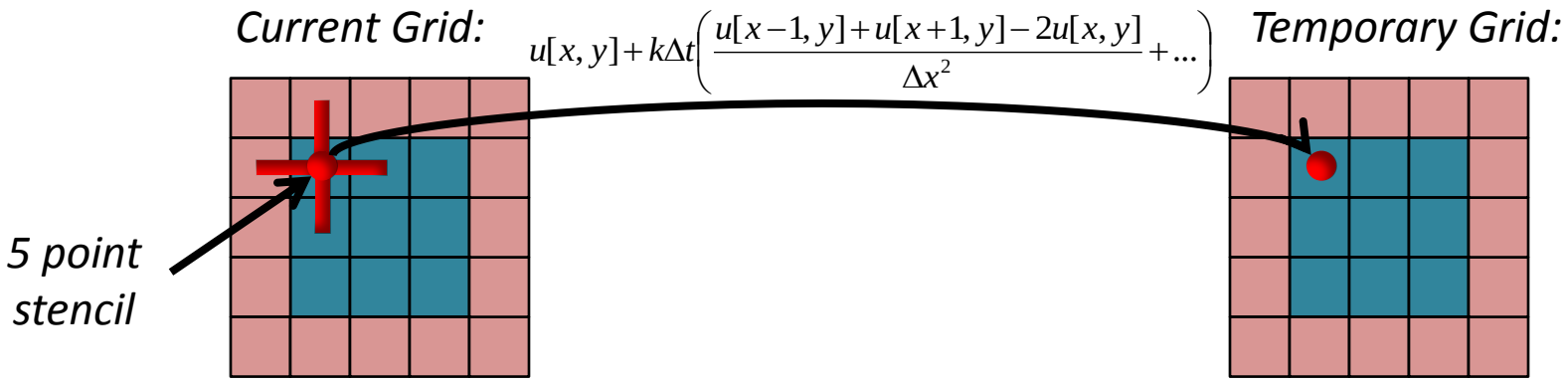
- Grid Structure contains 2 grids and the grid size
- Grid:



- Ghost cells used as neighbors, needed for border cells of actual grid

Hands-On I
The Heat Conduction Example – Time Steps

- Time steps are calculated for all actual grid cells, results stored in second grid:



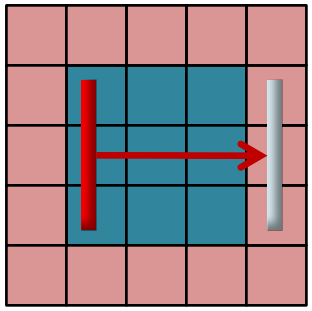
- Application switches grid after calculation for all grid cells

Hands-On I
The Heat Conduction Example – Boundary

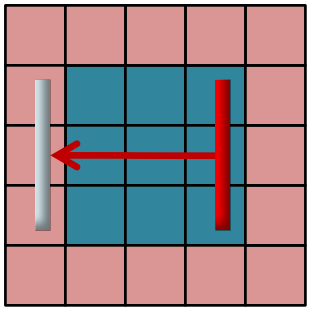
- 5 point stencil needs left, right, up, down neighbor for each grid point
- Ghost cells serve as neighbors
- Code uses periodic ghost cells
- After time step: copy operation to update ghost cells (=> “heatBoundary”)
- Update:

Grid:

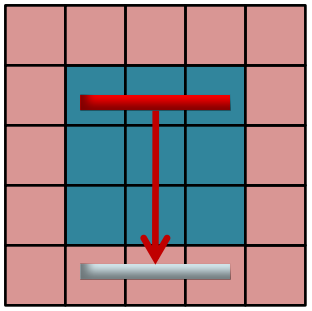
(1) Left border



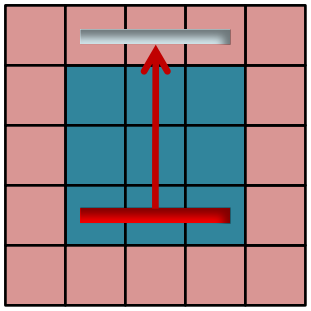
(2) Right border



(3) Top border



(4) Bottom border



Hands-On I
The Heat Conduction Example – Main Function

- Starts initialization
- Iterates time steps and boundary exchange (heatTimestep, heatBoundary)
- Uses 20 iterations
- Prints initial and final grid to standard output
- Calculates total energy at beginning and end (simple basic verification)

Hands-On I
The Heat Conduction Example – OpenMP Version

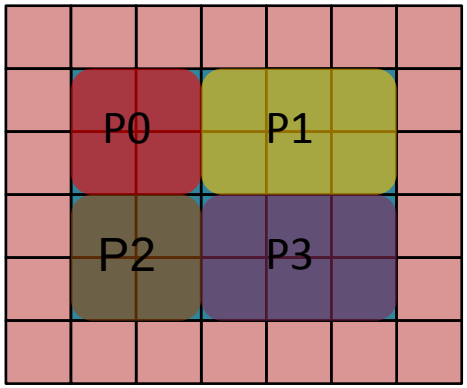
- Parallel for loop around the timestep function:

```
void heatTimestep(heatGrid* grid, double dt, double* dthetamax)
{
...
#pragma omp parallel  private(dtheta, x, y)
   {
#pragma omp for
      for (x=1; x <= grid->xsize;x++)
      {
         for (y=1; y <= grid->ysize; y++)
         {
            dtheta =…;
            grid->thetanew[x][y] = …;
         }
      }
```

⇒Each thread covers part of the 2D grid

Hands-On I
The Heat Conduction Example – MPI Version

- Each process has a copy of both grids
- Initialization and set-up done by all processes
- Each process calculates on a part of the overall grid:

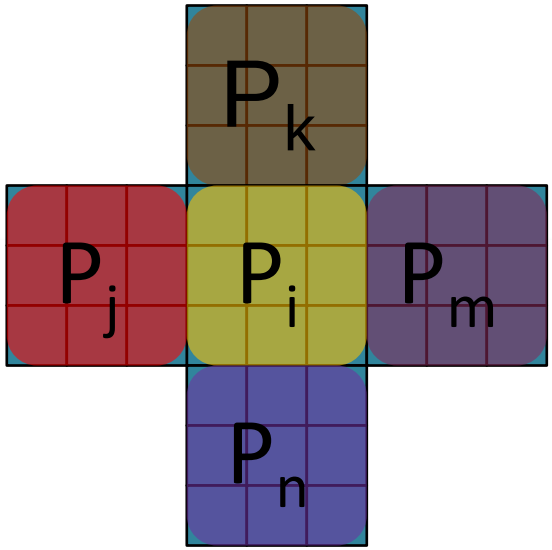


- Exchange with neighbors uses 2D Cartesian communicator
- Communications of a data columns uses a derived datatype
- Consequences:
  - New border exchange
  - Gathering of data necessary for output

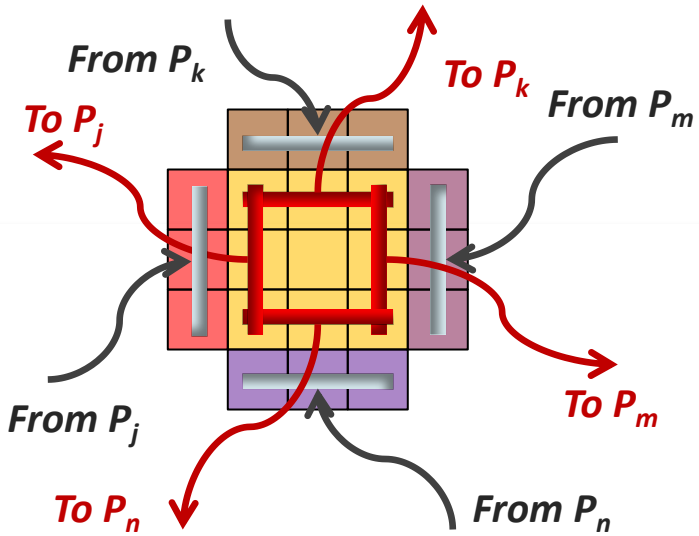
Hands-On I
The Heat Conduction Example – MPI Boundary

- MPI processes exchange borders with neighbors
- Cartesian grid provides up, down, left, right neighbors
- Periodic boundaries => grid is a donut
- Boundary exchange for process "P_i":

Neighborhood of P_i

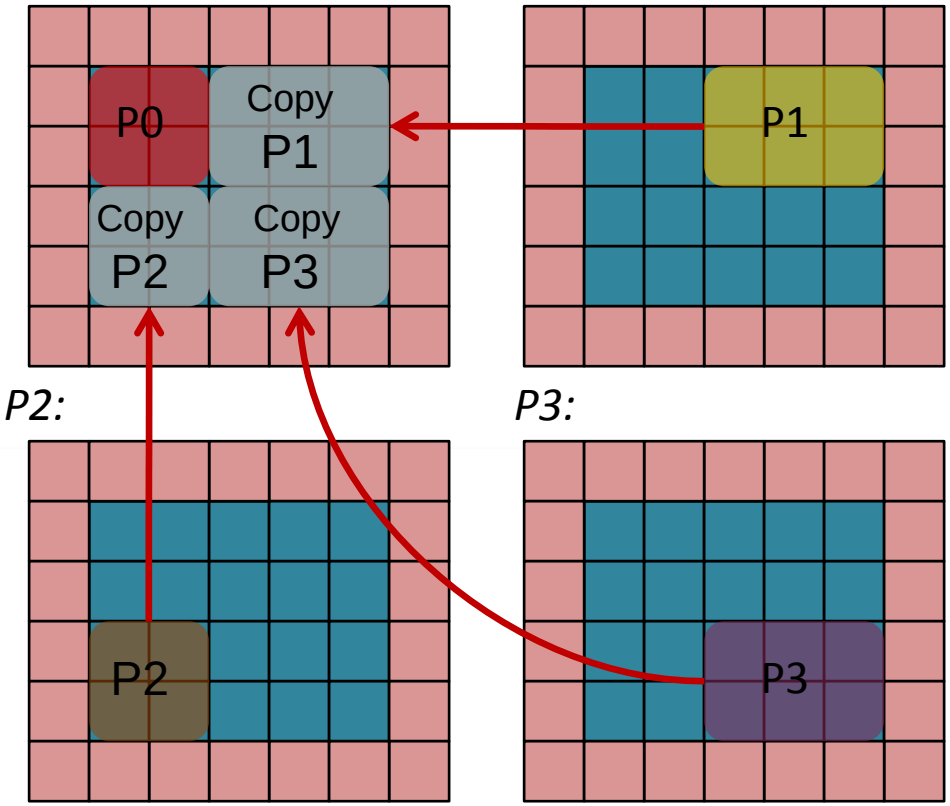


Exchange for P_i



Hands-On I
The Heat Conduction Example – MPI Gather

- Data distributed between processes
- Printing requires I/O or gathering data on one process
- Implementation gathers data on process 0:



Hands-On I
Cloud Access – Overview

- We provide access to an Amazon EC2 based installation of Allinea DDT, ISP, MUST, STAT, Intel Inspector XE, and ARCHER
- You will need a Laptop (a Mobile device may work, but is likely not convenient)

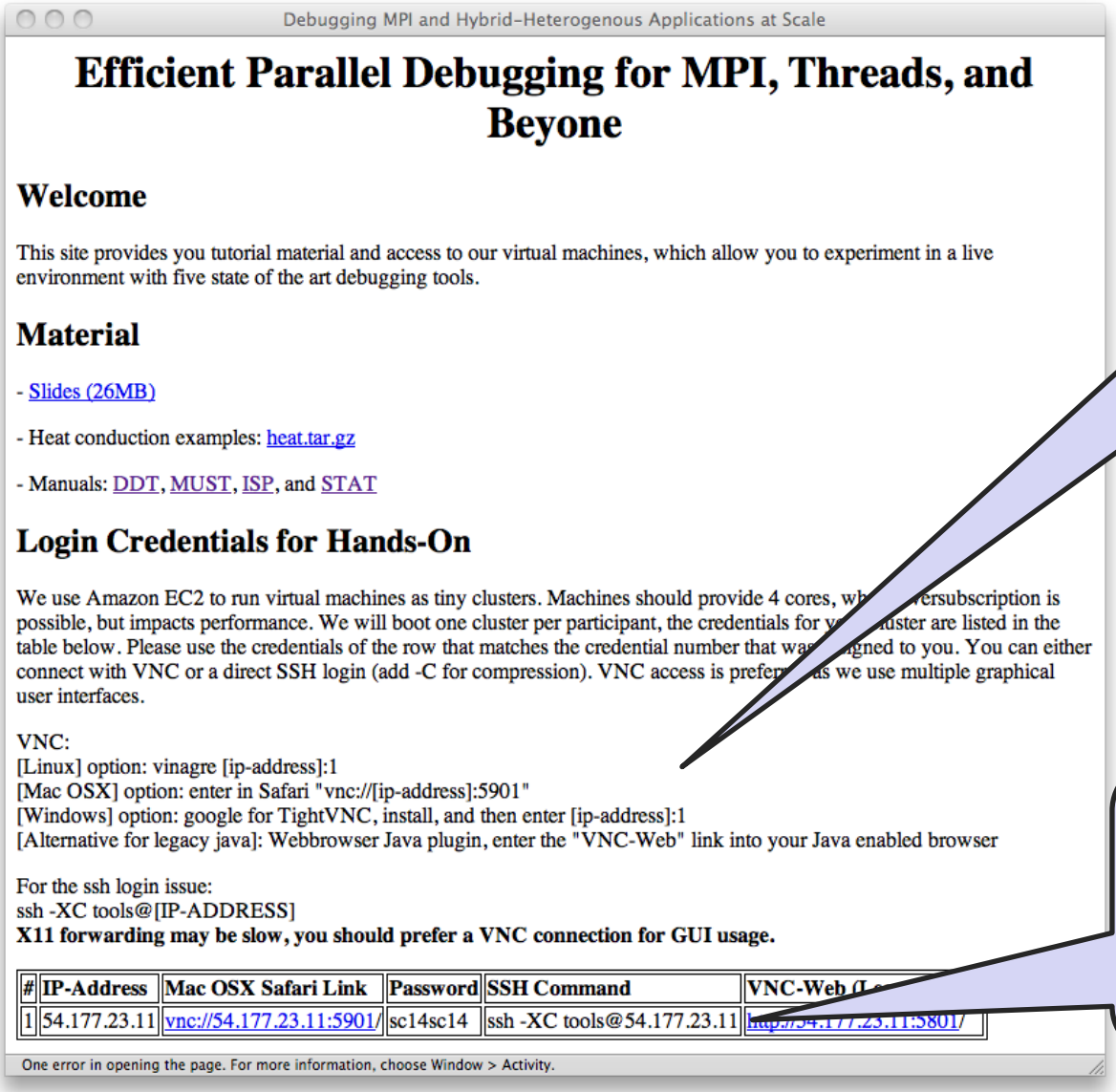
(1) Open the following webpage:

- **http://bit.ly/sc16-debug**
- (=>http://212.83.56.177/sc16-tutorial)

(2) Look up the IP address of your personal virtual machine (**Use the ID on the cheat sheet corner**)

(3) Connect (and test X11 Forwarding / VNC)

Hands-On I
Cloud Access – Credential website



Connection details (coming up in detail)

Credential table

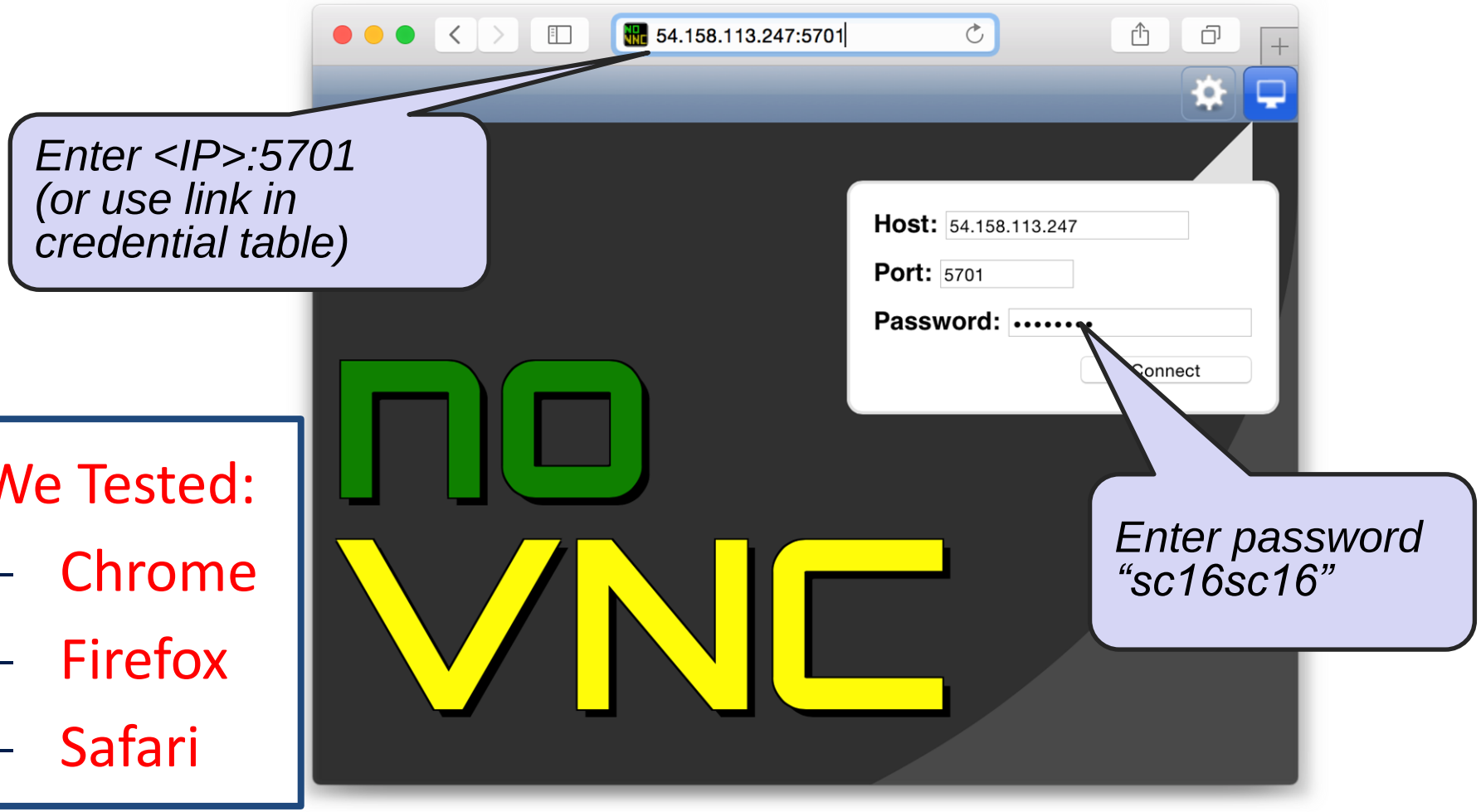
*Use **the row identified on the corner of your cheat sheet***

Hands-On I
Cloud Access – Login with VNC

- Preferred as we use multiple GUIs
- Linux option: “vinagre <ip>:1”
- Mac option: In Safari enter “vnc://<ip>:5901”
- Windows option: If you have a client, use it, e.g. TightVNC => “<ip>:1”
- Alternative for ALL platforms:
  - Enter the link of the “VNC-Web” column for your row into your favorite browser
    => NoVNC Browser-based VNC
- You enter a simple X11 environment (xfce)
- To open a shell (one available when you log in)
  - “Applications Menu” -> “Terminal Emulator”

Hands-On I
Cloud Access – Login with VNC (2)

- Browser-based VNC => „VNC-Web“ column



We Tested:

- Chrome
- Firefox
- Safari

Hands-On I
Cloud Access – Login with SSH

- Alternatively use a ssh login to your cluster
  - On windows “putty” is what gives you this choice
- Enable X11 forwarding and compression(!):
  - **ssh –XC tools@<IP>**
  - X11 forwarding won’t easily do on Windows!
- Password is in the credential table (**sc16sc16**)
- Graphical user interfaces (e.g. Allinea DDT, Intel Inspector XE) may be slow

Hands-On I
Cloud Access – VNC Screen Resolution

- The VNC connection uses a fixed (low) resolution
- [If needed] adjust as follows:
  (1) End your VNC connection
  (2) Login via SSH (not VNC!)
  (3) Issue the following command:
    - restart-vnc.sh <WIDTH>x<HEIGHT>
    - E.g.: restart-vnc.sh 1024x768
  (4) Login via VNC again
- Also use this if the VNC-Server crashes

Hands-On I
Cloud Access – Directory Structure

% cd $HOME
% ls

- manuals
  - Manuals for all tools
- hands-on-1/{c|f90}
  - The heat example for Hands-on I
- hands-on-2
  - {c|f90}: heat example for all tools
  - {must|isp|ddt|inspxe|archer|stat}-extra
    - Extra material for tool specific training
    - Look here if you are quick
  - NPB3.3-MPI
    - A benchmark to toy with the tools

Hands-On I
Heat Example – Build & Run

- Login to your virtual machine, then:

```
% cd $HOME/hands-on-1/{c|f90}
```

- Build and run as follows:

```
% mpi{cc/f90} -g heat[VERSION].{c/F90} –o heat.exe
% mpirun -np 4 heat.exe
```

Hands-On I
Heat Example – Output


```
% mpirun -np 4 heat.exe
intial grid:
=============================================================================
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 …
 .  1.2 1.7 1.9 2.0 1.9 1.7 1.2  .  .  .  .  .  .  .  .  .  .  .
 .  .  0.9 1.3 1.4 1.3 0.9  .  .  .  .  .  .  .  .  .  .  .  .
 …
grid after 16 iterations:
=============================================================================
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  .
 …
 0.0 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.0  .  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  .
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  .  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  .
= Energy Conservation Check =
 initial Energy:    113.484
   final Energy:    108.893
     Difference:     -4.590
```

Hands-On I
Heat Example – Versions for Hands-On I

- We have 4 different version of the example:
  - heatC-MPI-MUST.c / heatF-MPI-MUST.F90

    "Is known to exhaust MPI resources and let MPI implementations crash after long runs, are all these MPI implementations broken?"
  - heatC-MPI-DDT.c / heatF-MPI-DDT.F90

    "Runs through, usually; During termination it crashes with lots of error output, but we have a result, so who cares?"
  - heatC-omp-Archer.c / heatF-omp-Archer.F90

    "Well this example shows the importance of optimization: With -O0 it loses some energy, but once you enable optimization it works like a charm!"
  - heatC-MPI-STAT.c / heatF-MPI-STAT.F90

    "This one appears to hang, but probably the calculations are just taking their time."

Hands-On I
Heat Example – One Issue for Each Tool

- The name of the source files identify the tool to use (heatC-MPI-MUST.c => MUST)
- The cheat-sheet identifies the steps to apply the tools
- The tools identify the issues in the examples almost spot-on (little real debugging)
- **Goal**:
  - Give each tool a try, feel free to start in any order
  - Try to correct the issues!
  - We are happy to help, there are Lots! of tool experts in the room, just for you!

Hands-On I
Heat Example – MUST

- Build:

```
% mpicc -g heatC-MPI-MUST.c
```

C

```
% mpif90 -g heatF-MPI-MUST.F90
```

Fortran

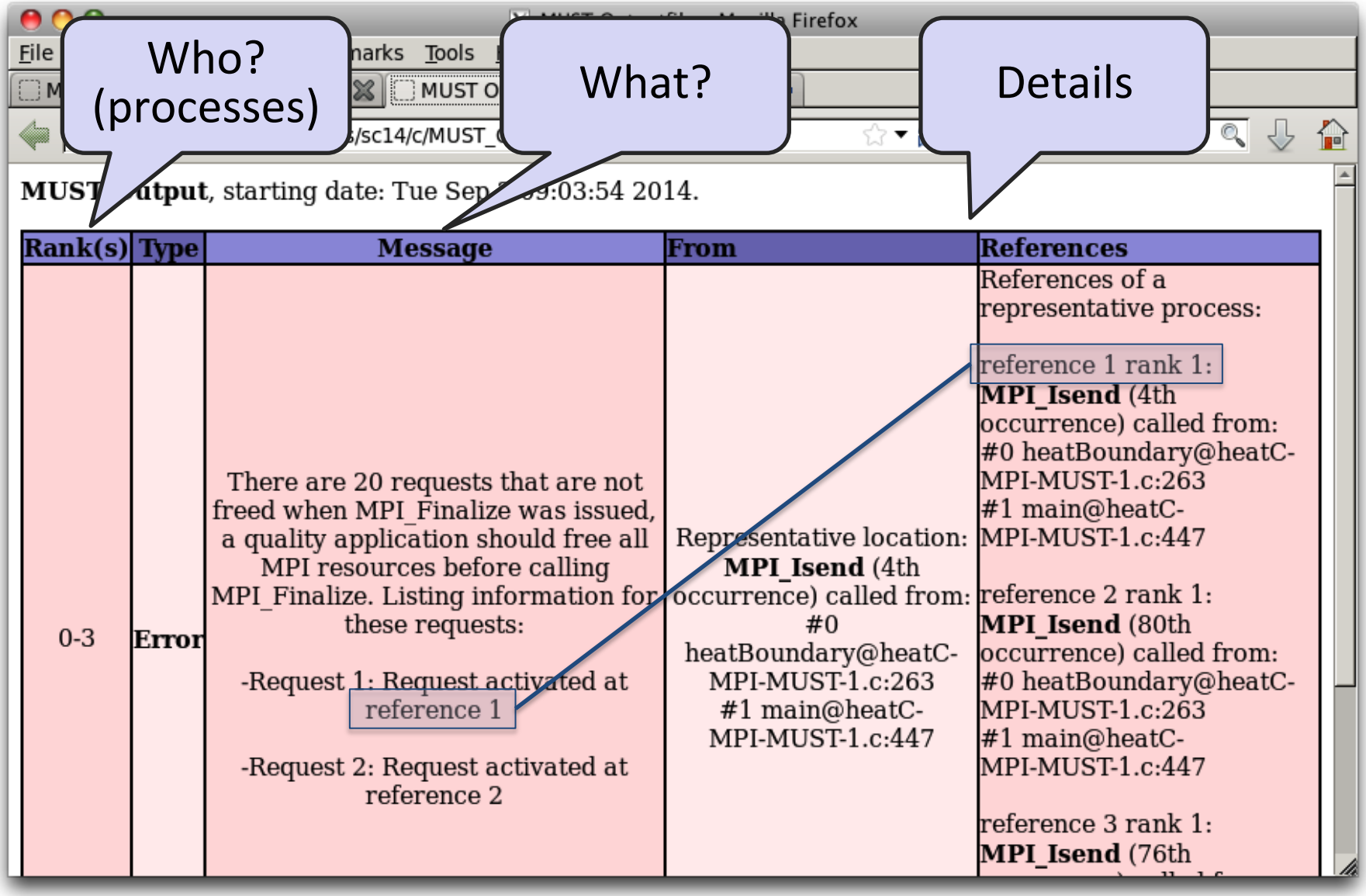
- Run:

```
% mustrun -np 4 a.out
```

- Investigate:

```
% firefox MUST_Output.html
```

Hands-On I
Heat Example – MUST, Report



Hands-On I
Heat Example – MUST, The Issue

- Source:

```
void heatBoundary(heatGrid* grid, dataMPI* mympi)
{
   MPI_Status statuses[4], status;
   MPI_Request requests[4];

   /*Send left column to left neighbor*/
   ...

   /*Send lower row to bottom neighbor*/
   MPI_Isend (&(grid->theta[mympi->start_x][mympi->start_y+mympi->num_cells_y-1]),
         1, mympi->rowtype, mympi->down, 123, mympi->cart, &(requests[3]));
   ...

   /*Complete the non-blocking sends*/
   MPI_Waitall (3, requests, statuses);
}
```

MUST: MPI requests of this call not completed (freed)

=>Stored in requests[3]

This completes:

requests[0]-requests[2]

=> Use 4 instead of 3

Hands-On I
Heat Example – Allinea DDT

- Build:

```
% mpicc -g heatC-MPI-DDT.c
```

C

```
% mpif90 -g heatF-MPI-DDT.F90
```

Fortran

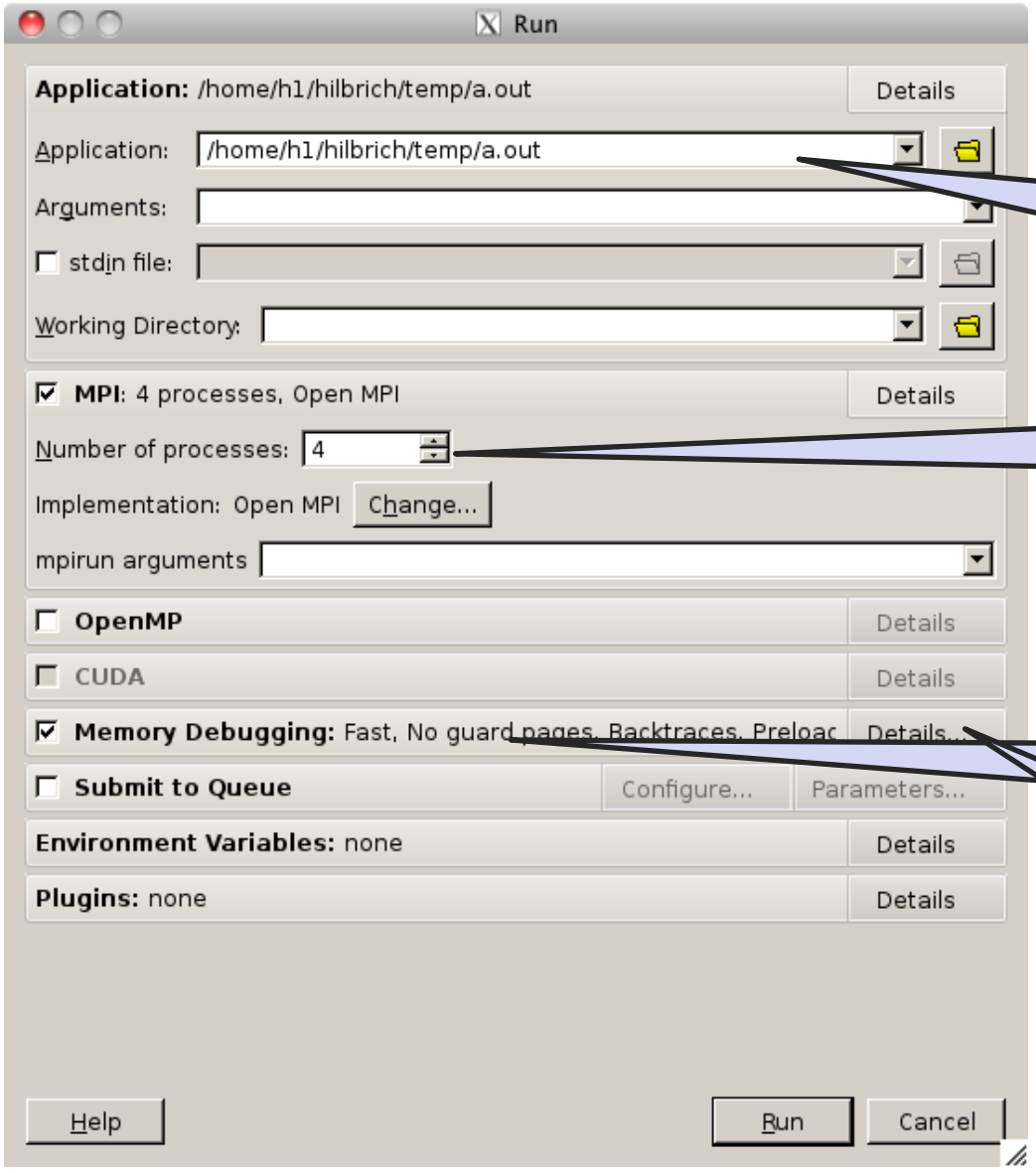
Note: Debugging the issue is more tricky in the Fortran version

- Run from within GUI:

```
% ddt a.out
```

- Enable:
  - Memory debugging
  - 4 processes

Hands-On I
Heat Example – DDT, GUI (1)



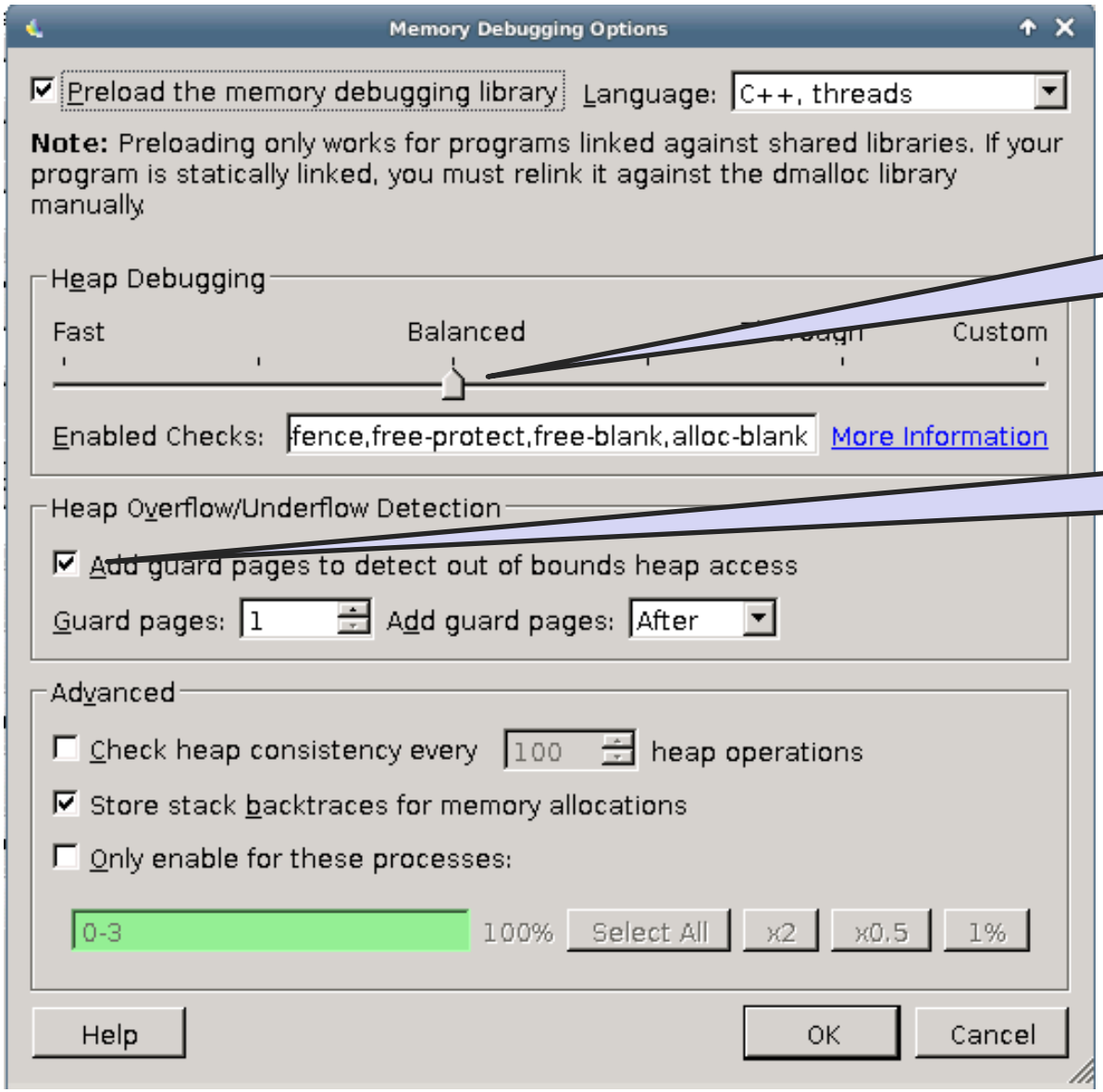
Binary to execute

Number of processes to use

Enable memory debugging

Configure memory debugging!

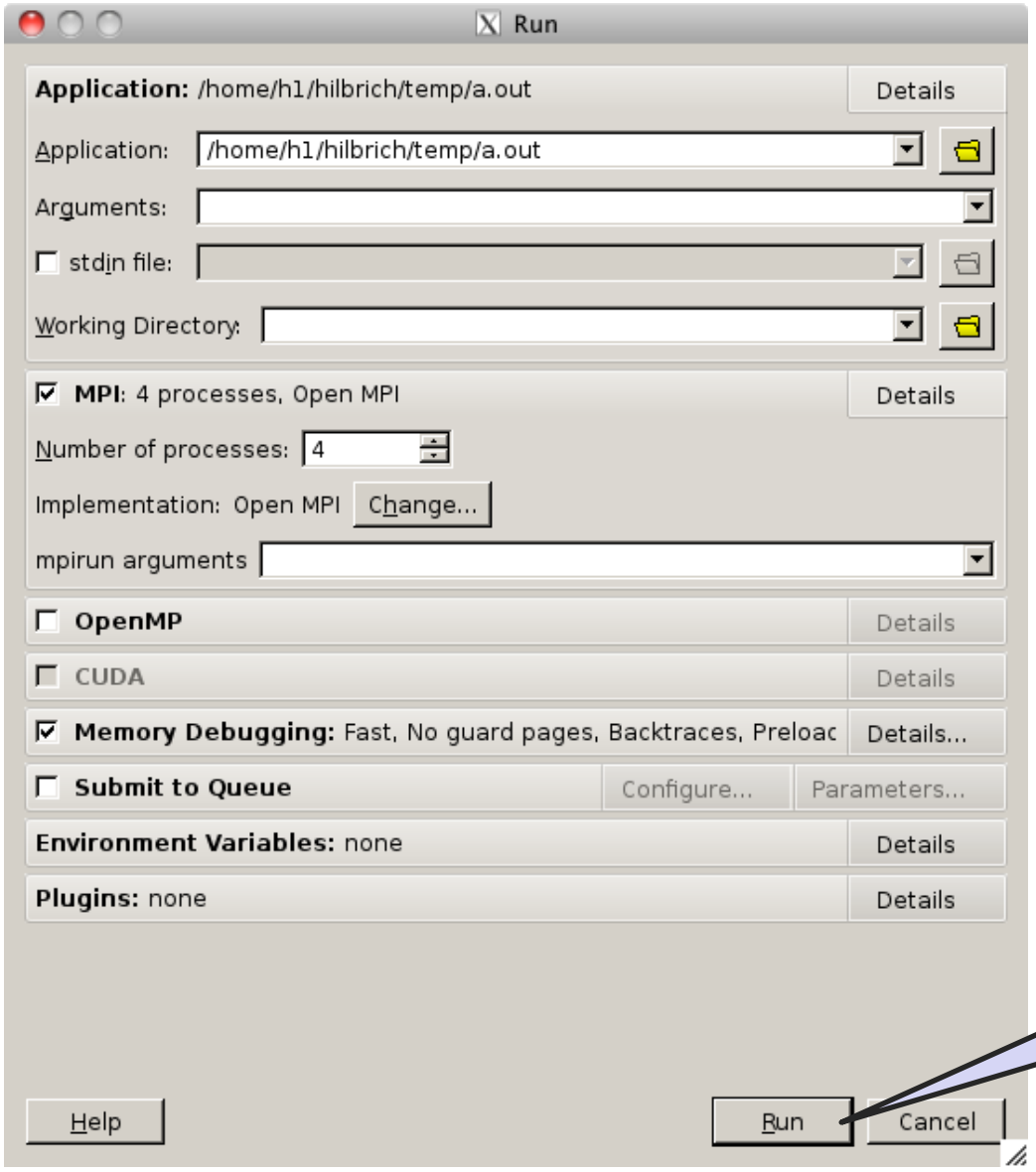
Hands-On I
Heat Example – DDT, GUI (2)



Select “Balanced”

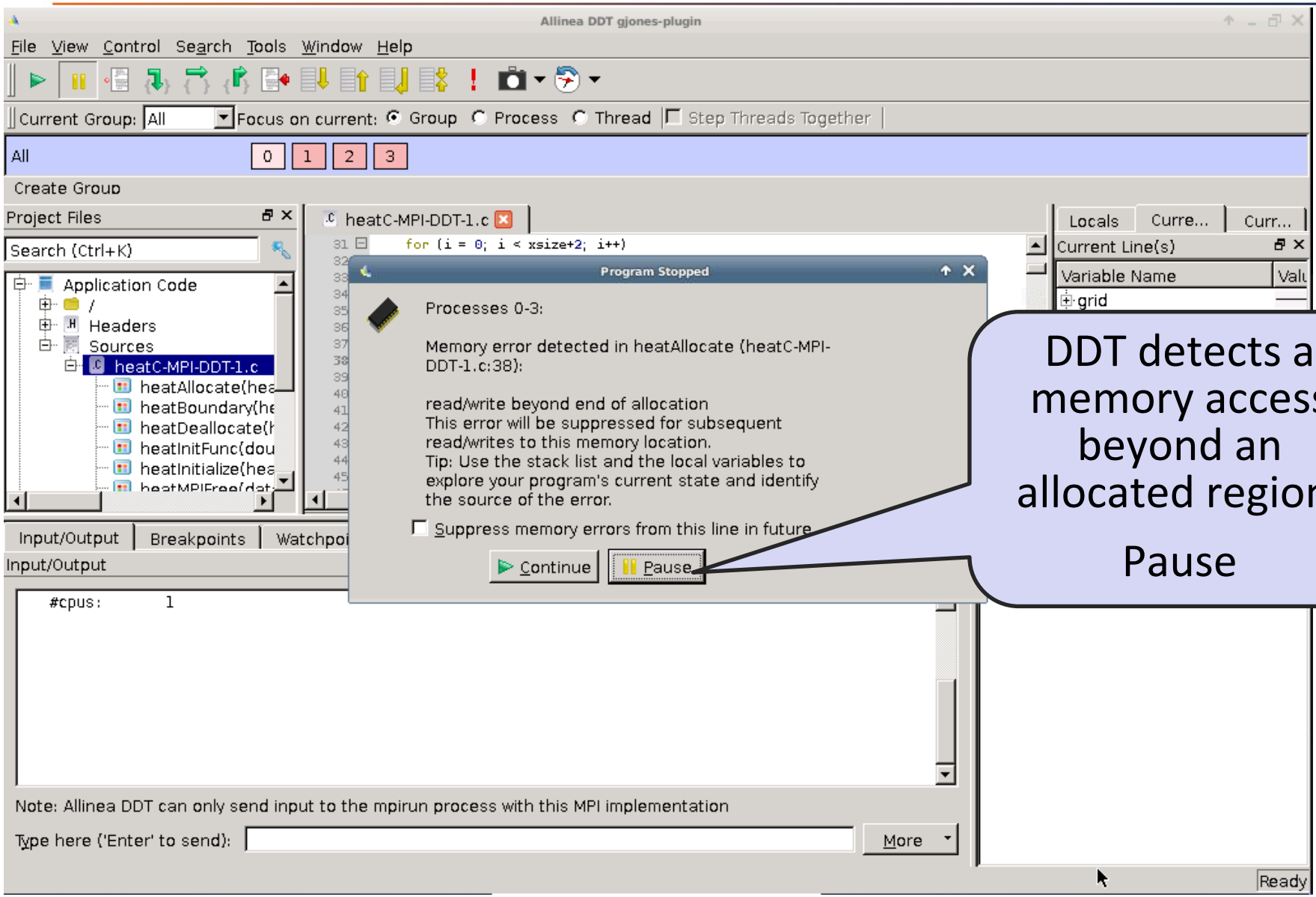
Enable guard pages!

Hands-On I
Heat Example – DDT, GUI (3)

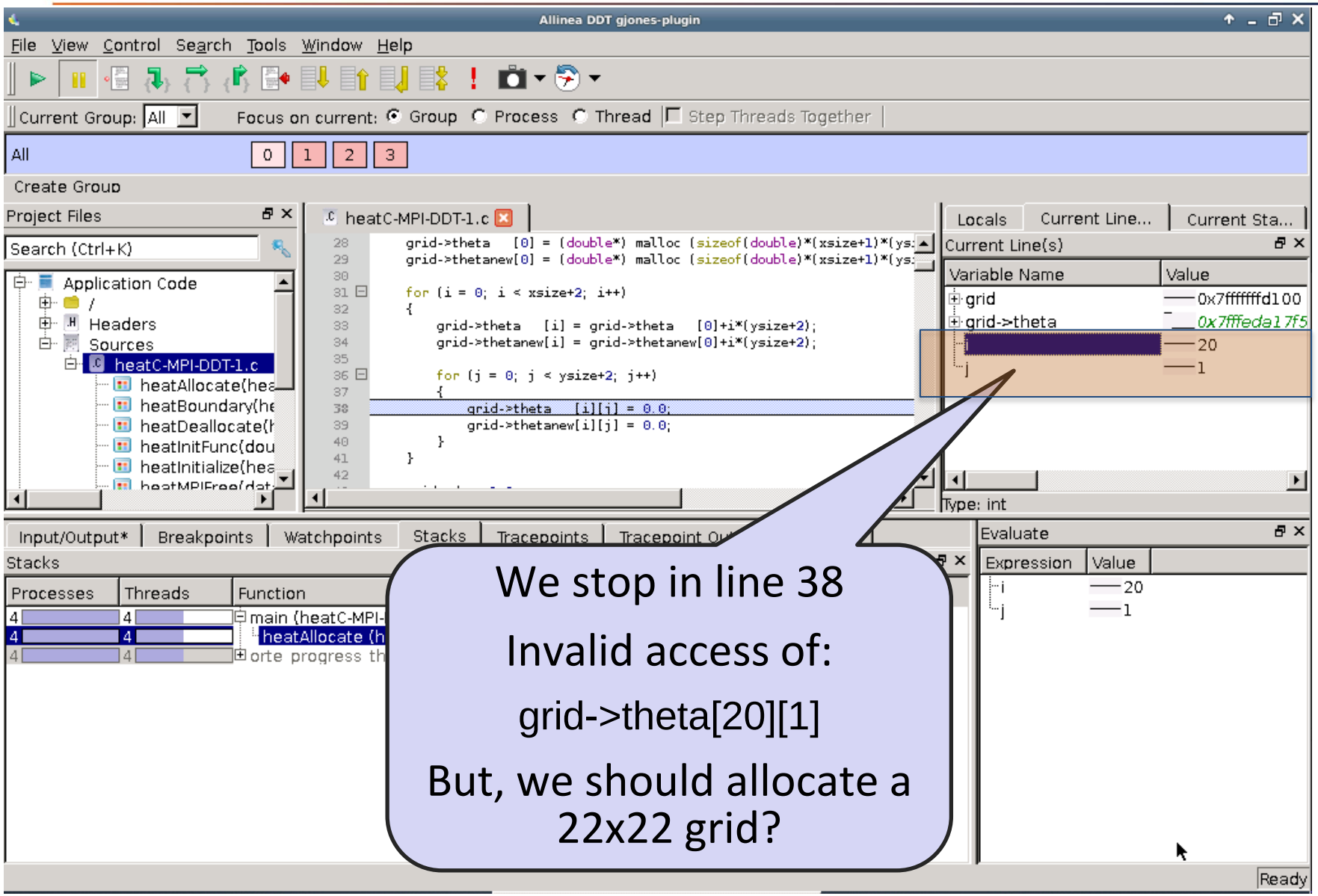


Run

Hands-On I
Heat Example – DDT, GUI (4)



Hands-On I
Heat Example – DDT, GUI (5)



We stop in line 38

Invalid access of:

grid->theta[20][1]

But, we should allocate a 22x22 grid?

Hands-On I
Heat Example – DDT, The Issue

- Source:

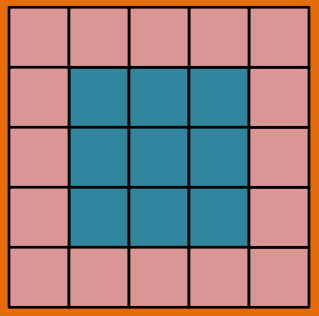
```
void heatAllocate(heatGrid *grid, int xsize, int ysize)
{
   int i,j;

   grid->xsize = xsize;
   grid->ysize = ysize;
   grid->theta    = (double**) malloc (sizeof(double*)*(xsize+1));
   grid->thetanew = (double**) malloc (sizeof(double*)*(xsize+1));
   grid->theta   [0] = (double*) malloc (sizeof(double)*(xsize+1)*(ysize+1));
   grid->thetanew[0] = (double*) malloc (sizeof(double)*(xsize+1)*(ysize+1));
   …
}
```

The grid must be sized as:

(xsize+2) * (ysize+2)

For xsize/size many cells + border around

=> Allocations must use (xsize+2) and (ysize+2) !



Hands-On I
Heat Example – Archer

- Build:

C

```
% clang-archer -g heatC-omp-Archer.c
```

Fortran

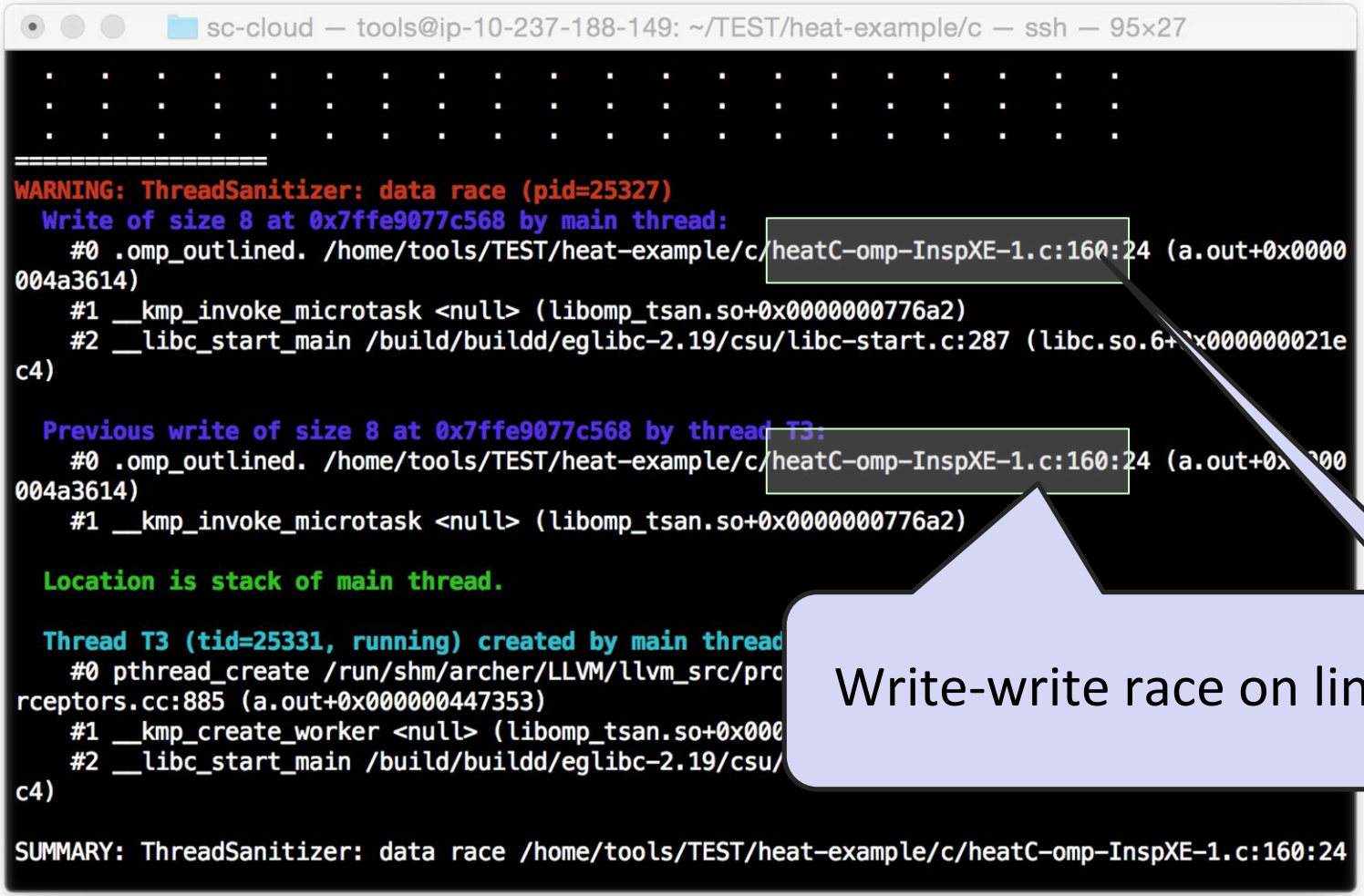
```
% gfortran -g -c -fopenmp -fsanitize=thread \
        heatF-omp-Archer.F90 -o heat.o
% gfortran -fsanitize=thread heat.o -lomp
```

- Run:

```
% export OMP_NUM_THREADS=4
% ./a.out
```

- In the output on the shell, analyze the Archer reports

Hands-On I
Heat Example – Archer, Result



- Correction as for InspectorXE

Hands-On I
Heat Example – Archer, The Issue

- Source:

```
void heatTimestep(heatGrid* grid, doub
{
   int x, y;
   double dtheta;

   /* calculate the time step: read fr... theta, write new timestep to thetanew */
#pragma omp parallel private(x, y)
   {
#pragma omp for
      for (x=1; x <= grid->xsize;x++)
      {
         for (y=1; y <= grid->ysize; y++)
         {
            dtheta = ...;
            ...
         }
      }
```

Solution:

⇒ Each threads needs its own dtheta

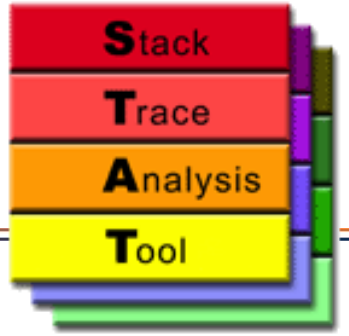
⇒ Use "**private(x,y,dtheta)**"

The race that ARCHER detects

⇒ dtheta is shared

⇒ All threads write to it without synchronization

Hands-On I
Heat Example – STAT



- Build:

```
% mpicc -g heatC-MPI-STAT.c
```

C

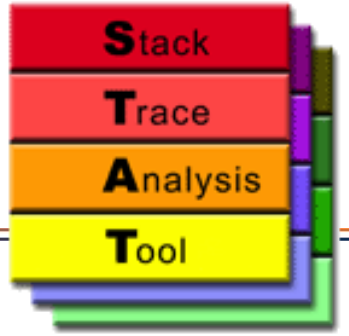
```
% mpif90 -g heatF-MPI-STAT.F90
```

Fortran

- Run (it will hang):

```
% mpirun –np 4 a.out
```

Hands-On I
Heat Example – STAT, Usage



- Second shell => Attach with STAT:

```
% ps -u tools
PID    TTY      TIME       CMD
...
6209   pts/0    00:00:00   mpirun
6222   ?        00:00:13   a.out
6223   ?        00:00:12   a.out
6224   ?        00:00:13   a.out
6225   ?        00:00:12   a.out
...
% stat-cl 6209
```

- Investigate the stack trace:

```
% stat-view stat_results/a.out.0000/03_a.out.0000.3D.dot
```

Hands-On I
Heat Example – STAT, Result

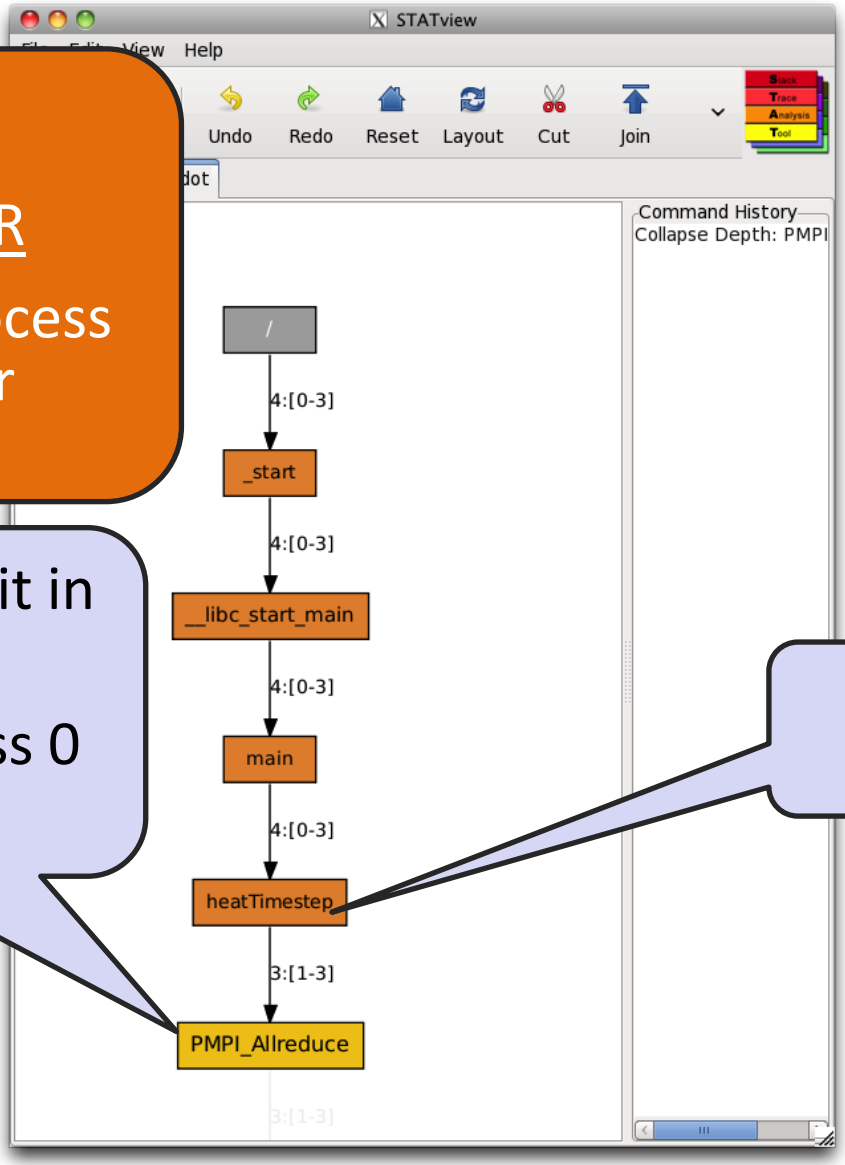
a) Investigate the source code of heatTimestep OR

b) Investigate process 0 in a debugger (gdb or DDT)

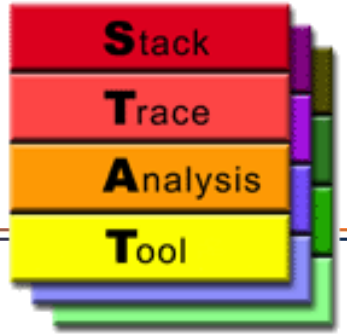
Processes 1-3 wait in an MPI_Allreduce

⇒ What is process 0 doing?

All processes in heatTimestep



Hands-On I
Heat Example – STAT, The Issue

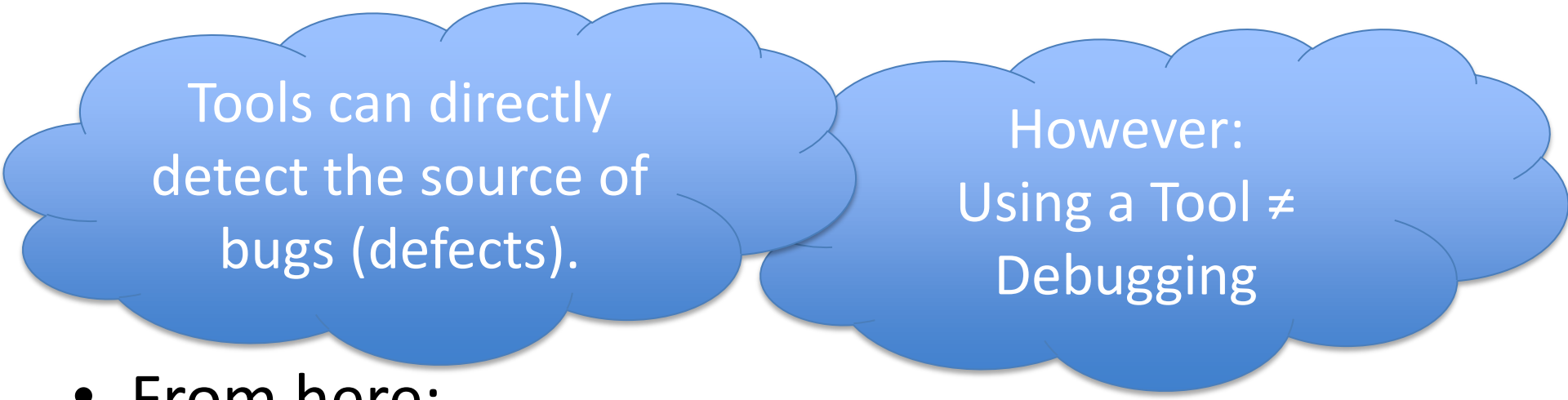


- Source:

```
void heatTimestep(heatGrid* grid, dataMPI* mympi, double dt, double* dthetamax)
{
   ...
   /* calculate the time step: read from theta, write new timestep to thetanew */
   /* Only calculate on a processes sub-grid */
   for (x=mympi->start_x; x < mympi->start_x + mympi->num_cells_x;x++)
   {
      for (y=mympi->start_y; y < mympi->start_y + mympi->num_cells_y; y++)
      {
         ...
      }

      if (mympi->rank == 0)
         x = mympi->start_x;
   }
...
}
```

A regression causes rank 0 to loop indefinitely (Fortran: it's a read)

Hands-On I Summary



- From here:
  - Learn about discipline, intuition, and the role of tools
  - Engage in systematic debugging
  - Understand MPI and OpenMP specific defects
  - Learn more about the individual tools
  - Finally, a bit on defects in heterogeneous codes

allinea

High performance tools to debug, profile, and analyze your applications

The Tao of Debugging

Discipline, Magic, Inspiration, Science

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5], Ganesh Gopalakrishnan[1], Mark O'Connor[3], and Joachim Protze[4]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Organization

- Introduction and Tools
- Hands-On I

Coffee Break

- **The Tao of Debugging**
- Systematic Debugging

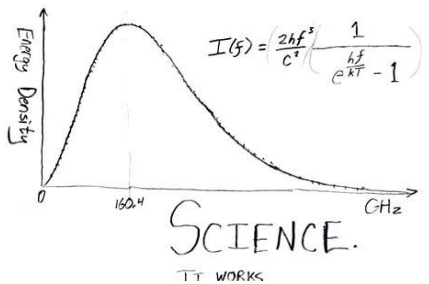
Lunch Break

- Defects in Practice
- MPI Applications
- Threaded Applications

Coffee Break

- Hands-On II
- Beyond MPI and Threads
- Discussion and Wrap-Up

The Tao of Debugging



Discipline Magic Inspiration Science

allinea

The Tao of Debugging

Bohrbug	Steady, dependable bug
Heisenbug	Vanishes when you try to debug (observe)
Mandelbug	Complexity and obscurity of the cause is so great that it appears chaotic
Schroedinbug	First occurs after someone reads the source file and deduces that it never worked, after which the program ceases to work

Debugging by Discipline

Three techniques, rigorously applied, will dramatically improve your life.

At least, when it's time to debug.

Discipline 1: Logbook

$ mkdir logs
$ **vim** logs/segfault-at-4096-procs

When running lu.E.4096 with the trace-4410.dat set, the job exited with: "An error occurred in MPI_Send [li346-209:25319] on communicator MPI_COMM_WORLD MPI_ERR_RANK: invalid rank".

To reproduce: mpiexec -n 4096 lu.W.4096 trace-4410.dat on supermuc. Seems to happen every time.

* Tried reading core file with gdb, "File truncated“ * Set ulimit -c unlimited and ran again: ...

Exactly what happened, including error messages

How to reproduce it

Investigation notes

Discipline 2: Test script

1. Compile with debug information: -g -O0
2. Submit the job with the correct inputs
3. Wait for it to run...
4. Check the output for errors

So much can - and will - go wrong!

Instead, spend 2 minutes adapting an existing test script that does all of this and prints PASS or FAIL:

```
$ logs/segfault-at-4096-procs.sh
Sep 27 15:29: Queued as job.43214
Sep 27 18:01: Running...
Sep 27 19:29: FAIL
```

Discipline 3: Regression Testing

Simple
- Sanity and performance checks
- Reliability is crucial – no false positives allowed

Regular
- Run on every code commit
- Speed is important – don't run entire cases

Auto
- Use source control hooks to submit test jobs
- OSS to view and manage runs (http://jenkins-ci.org)

Discipline 3: Regression Testing

DDT

- Prefix sanity tests with ddt --offline $REV.html …
- Integrate debug reports into Jenkins/CI system

MAP

- Prefix performance tests with: map --profile …
- MAP's editor highlights source lines changed

PR

- Generate HTML reports directly or from MAP files
- Integrate into Jenkins/CI & graph metrics over time

Summary: Discipline

Efficiency comes through preparation. Debugging a problem is much easier when you can:

1. Track what you've tried so far	**Logbook**
2. Reproduce bugs with a single command	**Test script**
3. Make and undo changes fearlessly	**Regression testing**

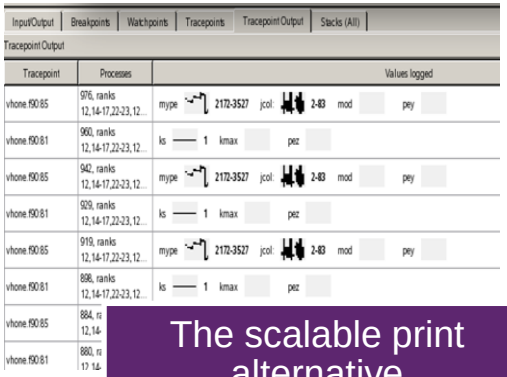
When these are combined, **magical things** can happen...

Debugging by Magic

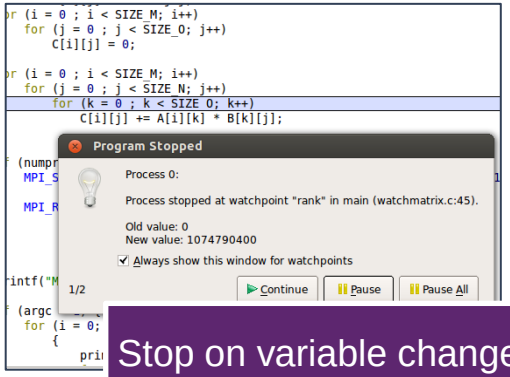
Any technology sufficiently advanced is indistinguishable from magic.

Unpredictable,
dangerous,
irresistible.

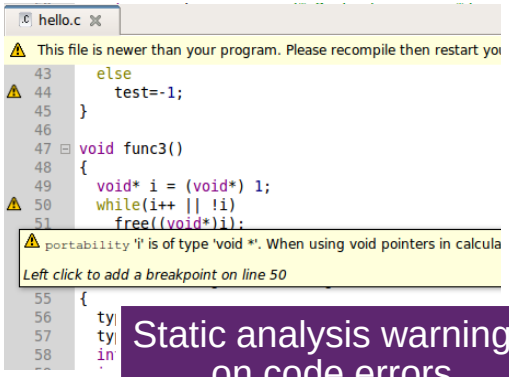
Favourite Magical Incantations



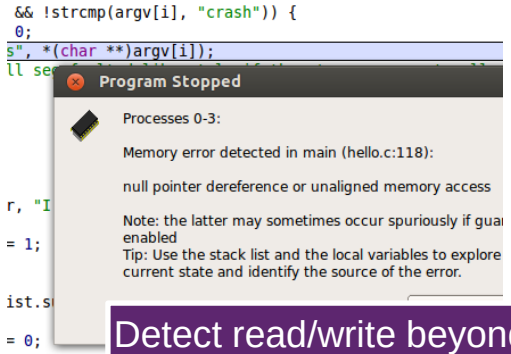
The scalable print alternative



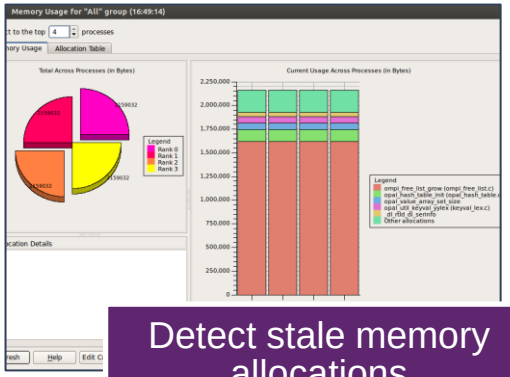
Stop on variable change



Static analysis warnings on code errors



Detect read/write beyond array bounds



Detect stale memory allocations

allinea FORGE

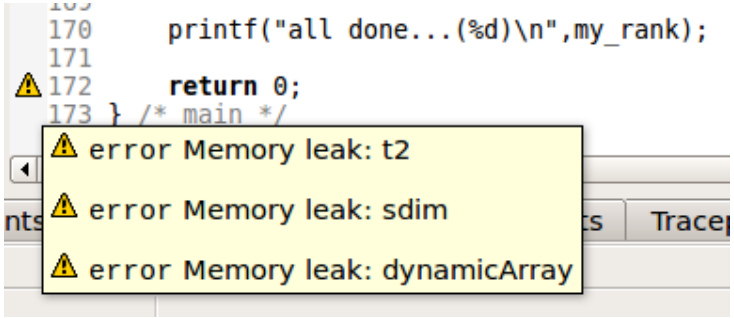
Protective Magic

Static analysis looks for common mistakes in code:

$ /path/to/ddt/libexec/cppcheck
$ /path/to/ddt/libexec/ftnchek (not a typo!)

Checking cstartmpi.c...
[cstartmpi.c:172] (error) Memory leak: t2

Also integrated into DDT's GUI and on by default:



Disciplined Magic

A magical servant!

$ hg bisect --bad

"The current version has a problem"

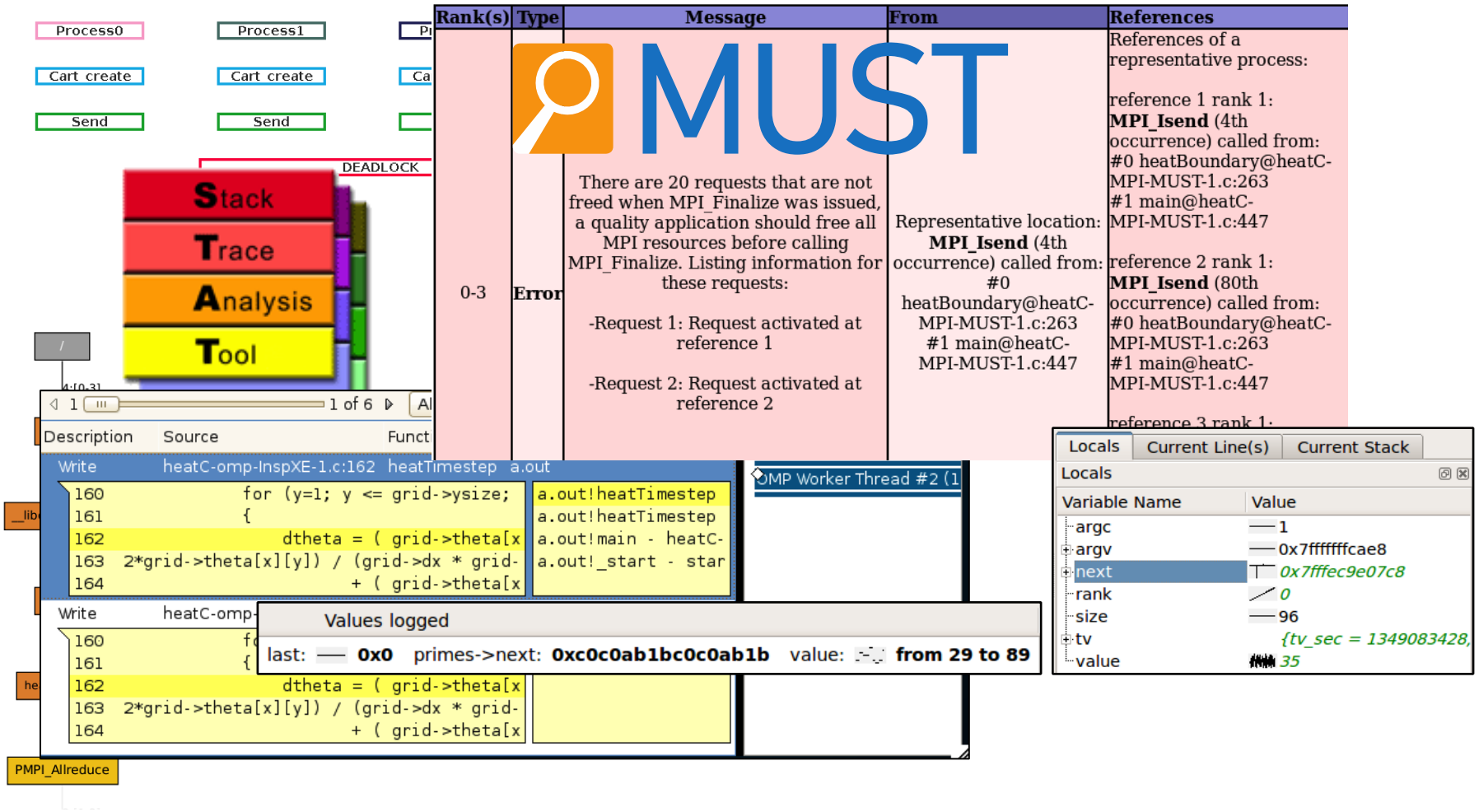
$ hg bisect --good 4

"It worked back in version 4"

$ hg bisect -c **logs/my-test.sh**

Master, you broke it when you committed: "Switched to single-precision matrices"

Advanced Magic



Summary: Magic

A good first step - can be a quick and easy way to fix:

Use source control and tests to create a magical servant:

```
$ hg bisect --bad
$ hg bisect --good 4
$ hg bisect -c logs/my-test.sh
$ hg log -pr <changeset id>
```

Bonus - static analysis:
cppcheck / ftncheck

Debugging by Inspiration

Look at the problem,
see the solution.

Vertrauen ist gut,
trust your instincts.

Kontrolle ist besser,
test if they're right

Inspiration: Test your instincts

"Ah, I bet that's because rl_x is negative, so then the y offset will be wrong, and ..."

Check to see if you're right **before** changing the code:

```
$ ddt -n 64 -offline log.html -trace-at blas.c:412,rl_x myprog.exe
```

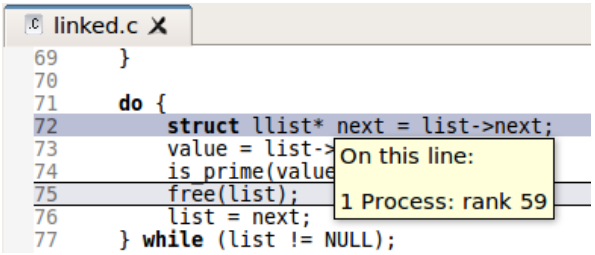
If your instinct is right, great! If it's wrong, **log it**:

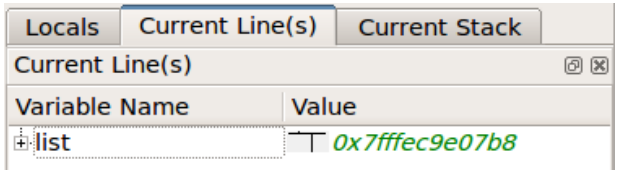
```
$ cat >> logs/incorrect-y-merge
        Hypothesis: Negative rl_x in blas.c:412 corrupts the y offset
        Observation: according to -trace-at, rl_x = 4.113 in blas.c:412
```

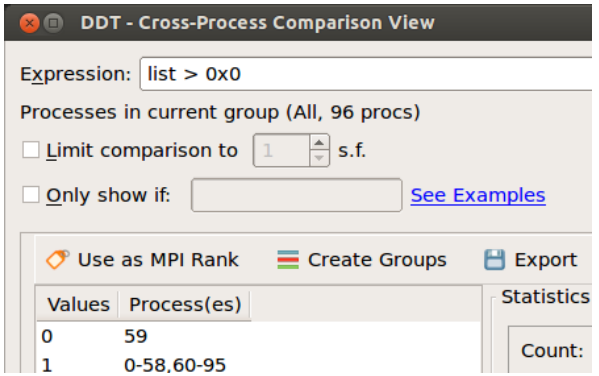
Inspiration: Just look at the problem

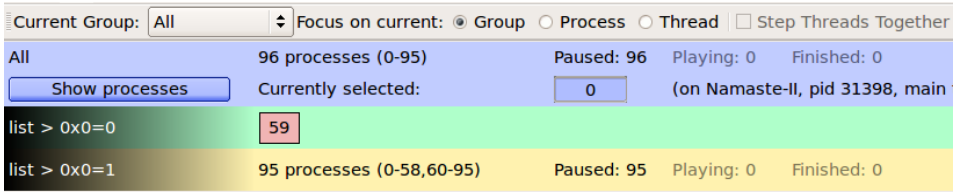
Explore the data and program flow in an **interactive** session:

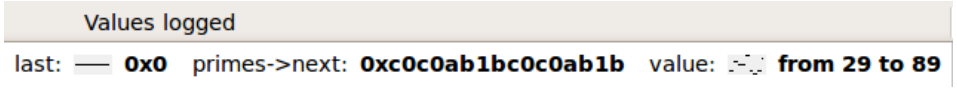
$ ddt -n 192 -start myprog.exe arg1 arg2

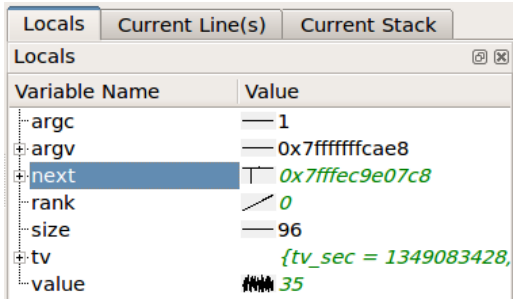












Alternative sources of inspiration

1. Explain the problem to a **rubber duck**.

2. Search your **logbooks**:

$ grep -ir blas4a logs/*
segfault-at-4096-procs: Hypothesis: Crash is in **blas4a**/blas4b
segfault-at-4096-procs: Observation: rank 9 at **blas4a**.c:145
segfault-at-4096-procs: Fix: Switched to ISend in **blas4a** and deadlock-with-openmpi: so the send buffers in **blas4a** overran.

Summary: Inspiration

When you have a sense for what the problem is:

Test it: $ ddt -offline log.html -trace-at mmult.c:412,rx,ry,rz

Log it: $ cat >> logs/short-problem-name

Suspect rx is out of bounds in mmult.c:412.

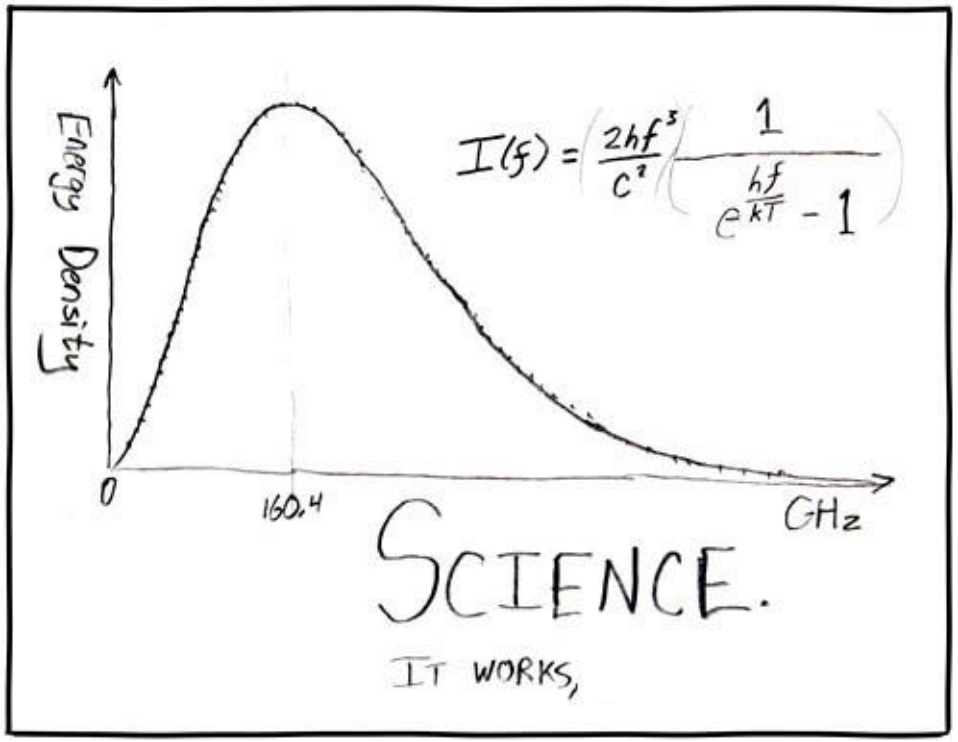
Testing with -trace-at mmult.c:412,rx,ry,rz showed...

Search your logbooks: $ grep -ri "out of bounds" logs/*

Talk to a rubber duck.

Tip - set a **time limit** for debugging by inspiration.
After 15 minutes, try **science**.

Debugging by Science



1. Hypothesis
2. Prediction
3. Experiment
4. Observation
5. Conclusion

There is a **reason**

and you **will** find it!

Debugging by Science

Image: xkcd CC-BY

Systematic Debugging

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], Joachim Protze[4] and Simone Atzeni[1]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Organization

- Introduction and Tools
- Hands-On I

Coffee Break

- The Tao of Debugging
- **Systematic Debugging**

Lunch Break

- Defects in Practice
- MPI Applications
- Threaded Applications

Coffee Break

- Hands-On II
- Beyond MPI and Threads
- Discussion and Wrap-Up

Systematic Debugging
Contents

1) Nomenclature
2) Basic Debugging Process
3) TRAFFIC Debugging Process

Systematic Debugging
Bug: disambiguation

- “Bug” is abstract and ambiguous, what is it:
  - Broken code
  - Broken state
  - Broken output
- Better distinguish:
  - Defect
  - Infection
  - Failure

Systematic Debugging
Defect

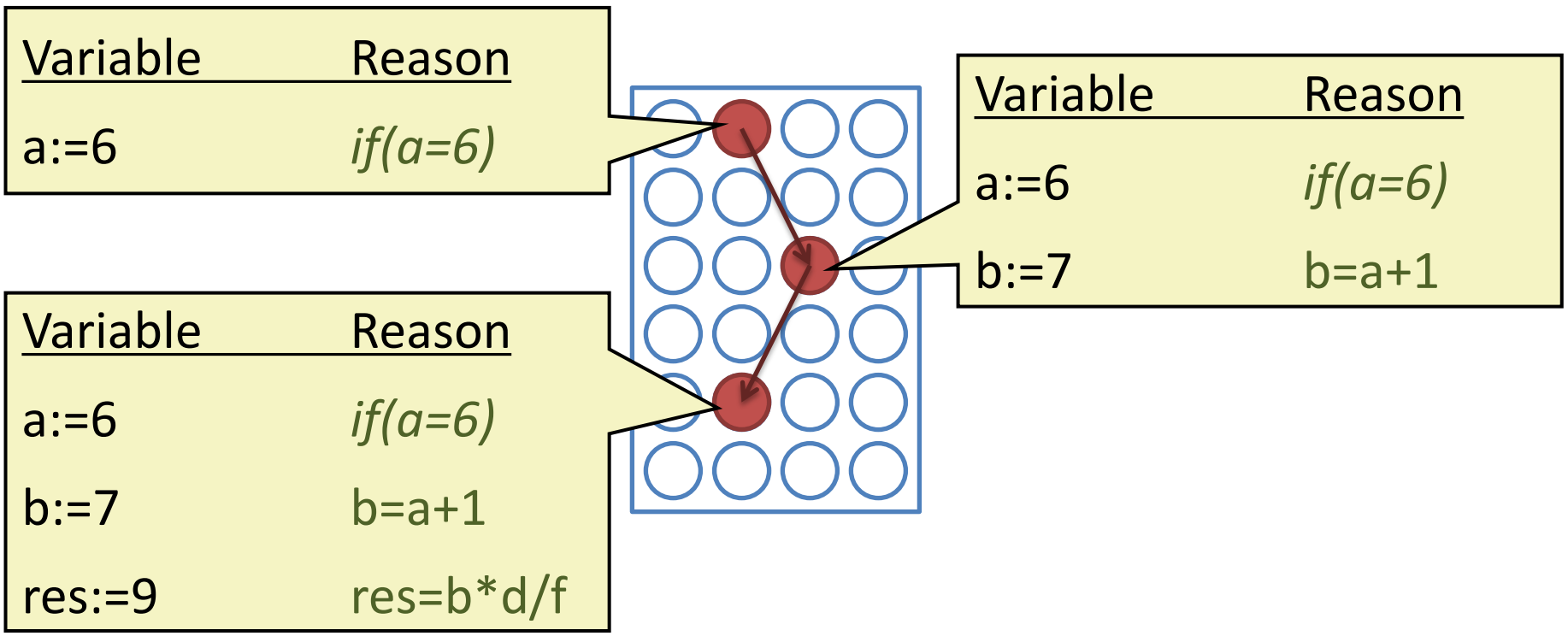
- Defect is a faulty piece of code
- Example?

```
int a;              // a has any value
if (a=5)            // is always true
      a++;          // a is 6 now
```

- Debugging is the process of identifying defects

Systematic Debugging
Infection

- The defect infects the program flow/state
- Infection propagates and results in failure



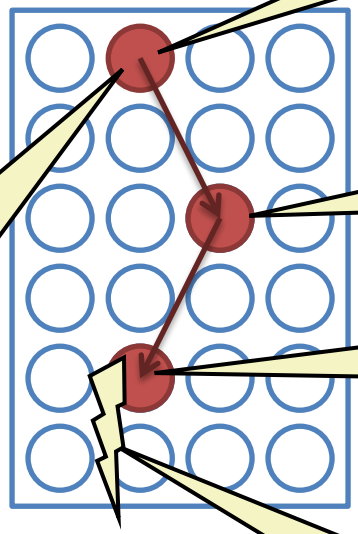
Systematic Debugging
Failure

- Failure: observable manifestation of a defect
- Observation types:
  - Faulty results
  - Crash
  - Admin calls you
  - Hang
- Example:

```
for (a=0; a<10; a++){       // a can be 0,…,10
      if (a=5)              // is always true
            a++;            // a is 6 now
      // some real code here
}
```

Systematic Debugging
Basic Debugging Process

1) Observe the failure
2) Trace infection-chain to defect
3) Identify the defect
4) Fix the defect

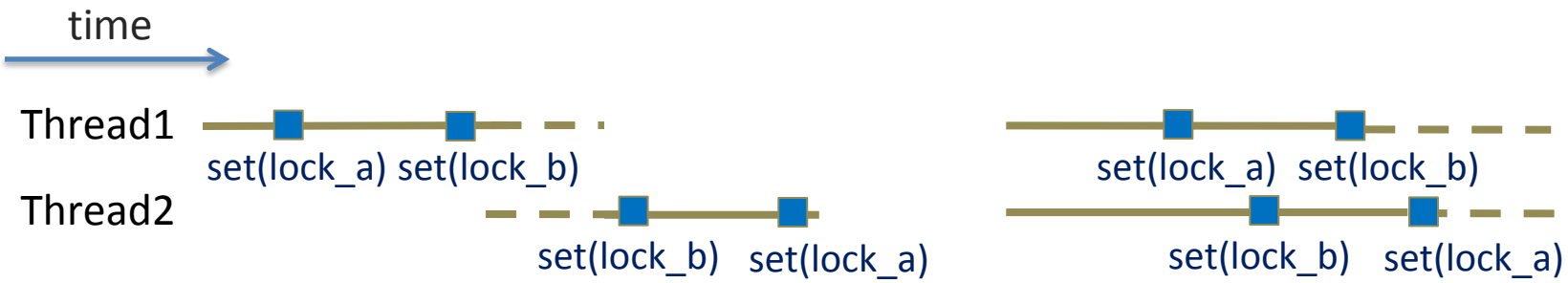


2)	*if(a=6)* => Why is ***a*** off!
2)	b=a+1 => Why is ***b*** off!
2)	res=b*d/f => Why is ***res*** off!
3)	*if(a=6)* => Oh the statement incorrectly sets ***a***
4)	=> It hast to be "if(a==6)"
1)	printf ("%d",res); => Huh, the print is off!

Systematic Debugging
Parallel Debugging Process

- Basically the same:
  - Trace the infection back from failure to defect
- But:
  - Infection might propagate across parallel units (threads/processes)

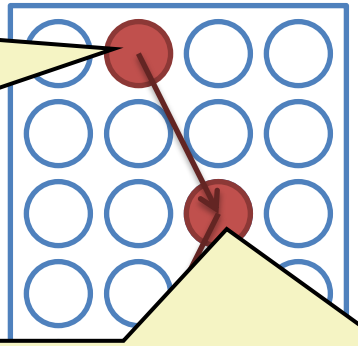
    ⇒Observe multiple/all processes/threads
  - Infection might be timing dependent



Systematic Debugging
Infection Propagation with MPI

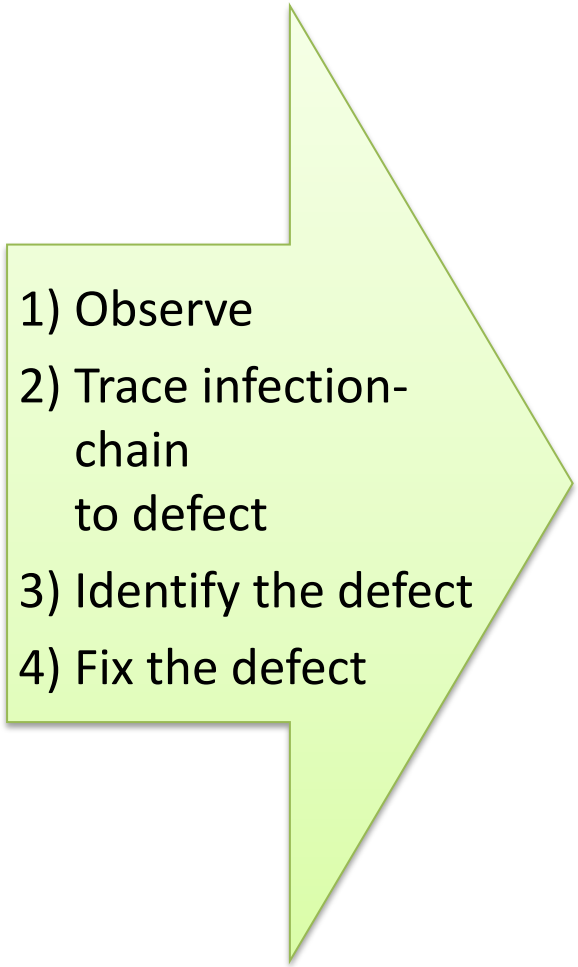
- The defect infects the program flow/state
- Infection propagates and results in failure

Rank	Variable	Reason
0	a:=6	*if(a=6)*



Rank	Variable	Reason
0	a:=6	*if(a=6)*
1	b:=6	*MPI_Recv(&b,…)*

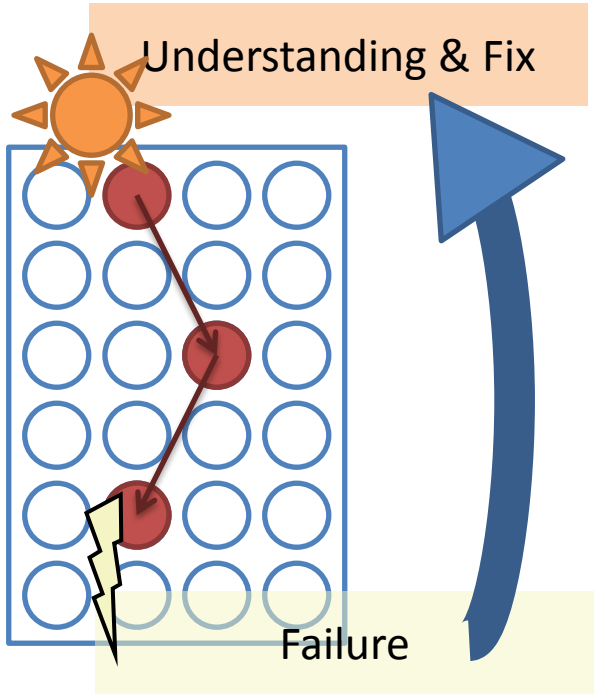
Systematic Debugging
Basic Debugging => TRAFFIC



1) **T**rack the problem
2) **R**eproduce the failure
3) **A**utomate and simplify
4) **F**ind possible infection origins
5) **F**ocus on most likely origins
6) **I**solate the infection chain
7) **C**orrect the defect

Systematic Debugging
When to follow TRAFFIC

- Task:



- Options:

Intuition	Tools	TRAFFIC
„This type of failure typically results from uninitialized memory; Did we initiallize that array in the new function?“	• “It’s a crash from within MPI, lets try MUST” • MUST catches a usage error • Investigate fix	• Follow the steps …

Can be very fast, but you can also get lost (or be very ineffective)
⇒ **Set a time limit**

Systematic Debugging
TRAFFIC: (1) – Track the problem

- Track failures to document code issues to:
  - Document ongoing debugging efforts
  - To synchronize in a team
  - Determine when a release is ready
  - Detect issues resulting from same defect
  - Get statistics on critical code parts
- E.g. Bug Tracker:
  - Bugzilla
  - trac
  - Excel sheet
  - A sheet of paper

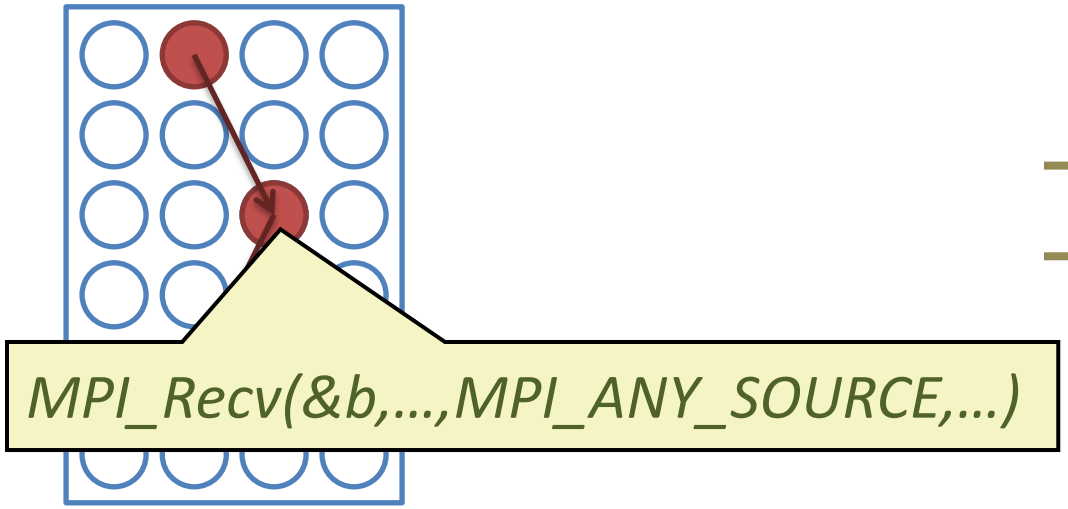
Systematic Debugging
TRAFFIC: (2+3) – Reproduce and Automate

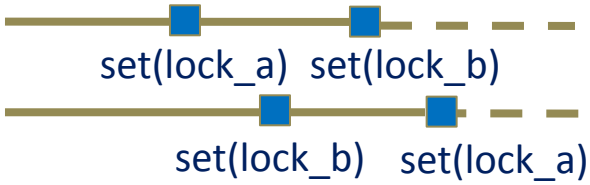
- Identify the input parameters that cause a Failure
  - E.g., input file, number of processes, seed for random number generator
- Automate the execution
  - Create a script that use the input and checks for expected output
- Simplify the execution
  - Reduce number of processes, reduce runtime, involved code (modules/libraries/…)
  - Reduce number of compiler warnings

$\Rightarrow$Use as test case in regression testing

Systematic Debugging
TRAFFIC: (2+3) – Reproduce & Automate, Parallelism

- Reproducing can be challenging





- Techniques:

Reproduce failed run	Use Tools	Scale up!
MPI/OpenMP: Research approaches available, e.g.: – MPIWiz – Retrospect – ReMPI	Intel InspectorXE, Archer, and ISP take care of analyzing all/some potential executions	Chance of failure to occur can increase with scale • Can suffice to efficiently debug an issue • Not practical for unit tests

Systematic Debugging
TRAFFIC: (4) – Find possible infection origins

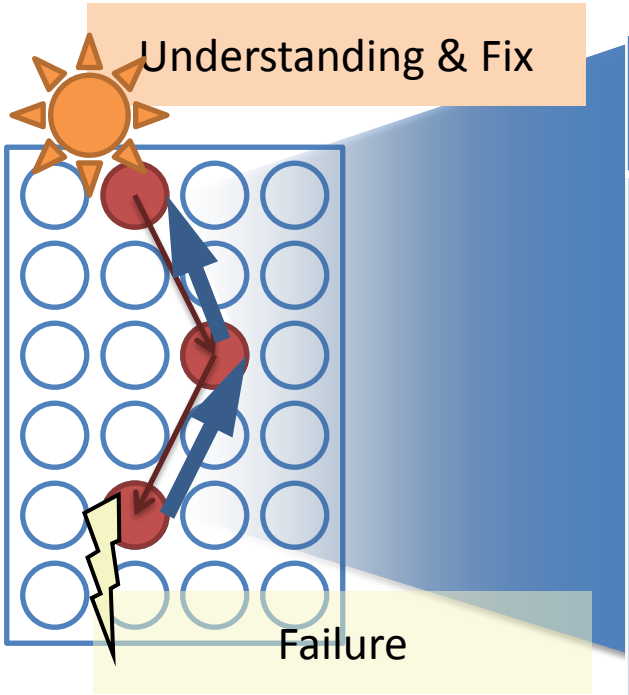
- In theory:
  - "Just follow the infection chain"
  - In practice: too long chain, hard to follow
  - Thus: try to find a *good* starting point in the chain
- Create hypotheses for infection origin:
  - From version tracking (recent modifications)
  - Reading the source code
  - Watching program output
  - Observe important/influential state with debugger
  - Intuition

Systematic Debugging
TRAFFIC: (5) – Focus on the most likely origin

- Use *asserts* to ensure expected states, failed asserts simplify origin detection dramatically
- Intuition or bad smells can guide
- Code coverage:
  - Only executed code can cause the problem
  - Compare with successful execution
- Most recent change (version tracking)
- Earlier infection sources

Systematic Debugging
TRAFFIC: (6) – Isolate the infection chain: Options

- Follow the infection chain:



- Options:

Ad-hoc (intuition)	Tools	Scientific Method
Your favorite simple solution: • Printf • Debugger • Just looking at the code • …	A helpful tool: • Allinea DDT memory debugging • ISP non-determinism coverage • …	• Slow but thorough (Hard to get lost, hard to fail)

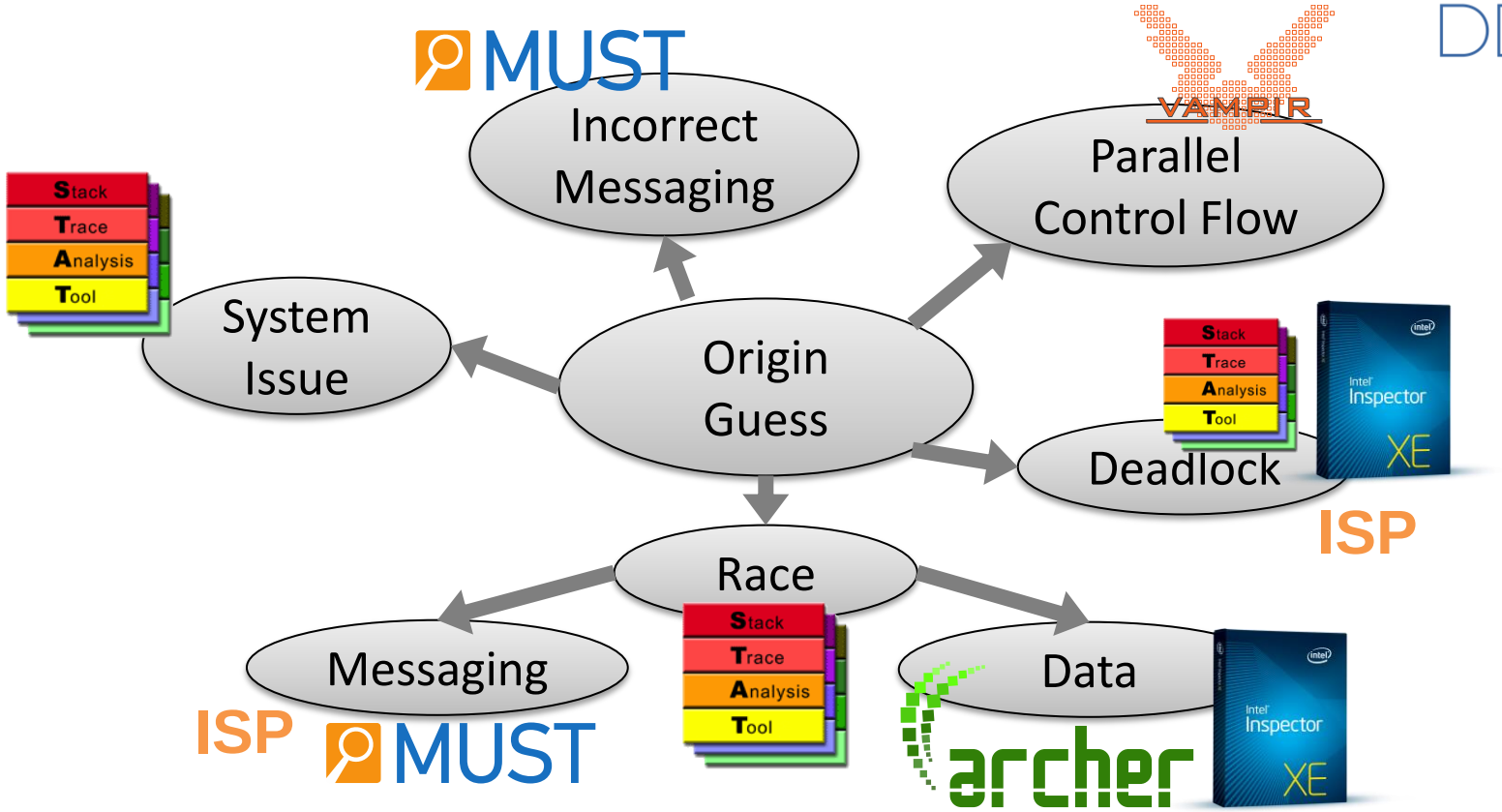
Set a timeout!
Examples: 10 min, 1 hour, or till lunch

Systematic Debugging
TRAFFIC: (6) – Isolate the infection chain: Ad-hoc

- Set a timeout to identify the defect that causes your failure ad-hoc, e.g., 1 hour (depends on complexity)
- Do you know this situation:
  - "Ah in timestep 77, on process 8, variable k has a value of 22."
  - "Why was I looking at this? What was the consequence of that?"
  - "What did I intend to try next, what did I already do?"
- For a complex defect, ad-hoc "trying around" can get you lost (=> very inefficient)

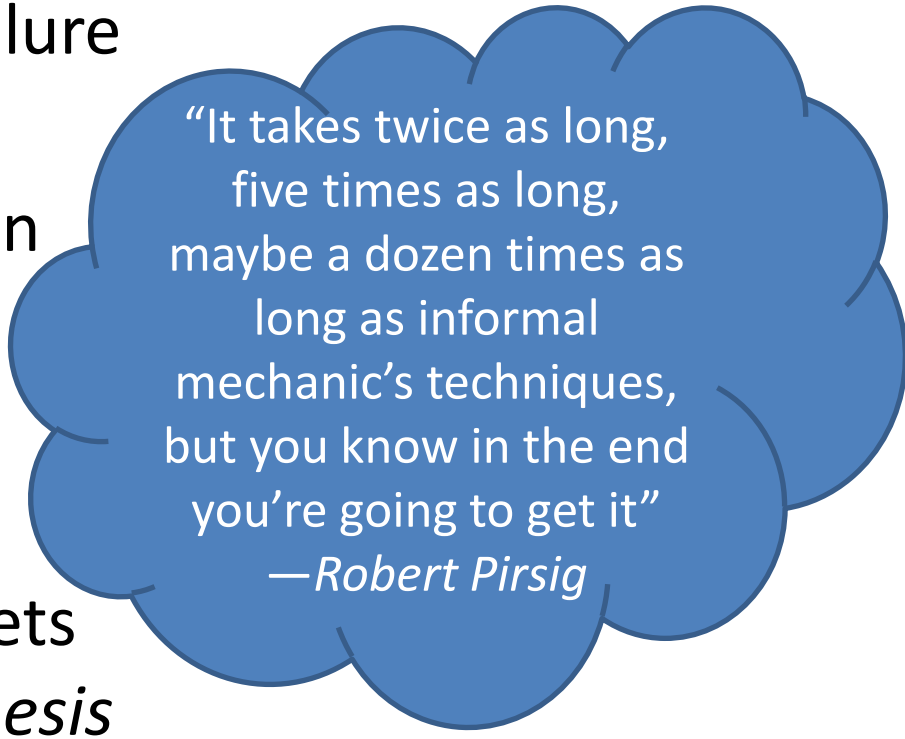
Systematic Debugging
TRAFFIC: (6) – Isolate the infection chain: Tools

- Failure type and infection origin imply towards a tool
- For parallelism related issues:
  - Almost all: allinea DDT



Systematic Debugging
TRAFFIC: (6) – Isolate the infection: Scientific Method

- Scientific method:
  - Create a *hypothesis* on failure origin
  - Make *predictions* based on *hypothesis*
  - Test the *hypothesis* by experiments
  - If *experiment* doesn't meets *predictions,* refine *hypothesis*
- IMPORTANT: Track the process, e.g., a log book



Systematic Debugging
TRAFFIC: (6) – Chain; Scientific method example

```
for (a=0; a<10; a++){      // a can be 0,…,10
      if (a=5)             // is always true
            a++;           // a is 6 now
      // some real code here
}
```

- Hypothesis:
  - The if statement lets the loop skip value a=5
- Prediction (of a in "some real code"):
  - a ε {0,1,2,3,4,6,7,8,9}
- Experiment:
  - Observe the value of a, e.g., with a debugger

  $\Rightarrow a$ = 5 (Always)
- New hypothesis:
  - The if statement incorrectly sets the value of a

Systematic Debugging
TRAFFIC: (7) – Correct the code

- Fix the defect:

```
for (a=0; a<10; a++){       // a can be 0,…,10
      if (a==5)             //Special case for 5
            a++;            //Skip one iteration
      // some real code here
}
```

- Verify the success by using the test case
- Prevent regression by
  - Including the test case to your test suite
  - Run the test suite regularly

Systematic Debugging
Further reading

- This session is inspired by:
  "Why Programs Fail:
  A Guide to Systematic Debugging"
  by Andreas Zeller
  - There is a free online course available:
    https://www.udacity.com/course/software-debugging--cs259

Outlook

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], Joachim Protze[4] and Simone Atzeni[1]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

After Lunch

- Introduction and Tools
- Hands-On I

Coffee Break

- The Tao of Debugging
- Systematic Debugging

Lunch Break

- **Defects in Practice**
- **MPI Applications**
- **Threaded Applications**

Coffee Break

- **Hands-On II**
- **Beyond MPI and Threads**
- **Discussion and Wrap-Up**

Learn how our tools solve problems in practice

Understand common pitfalls in MPI and OpenMP

Debug the "tricky" heat example or Play with the tools

Learn about debugging other parallel paradigms

Defects in Practice

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], and Joachim Protze[4]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Organization

- Introduction and Tools
- Hands-On I

Coffee Break

- The Tao of Debugging
- Systematic Debugging

Lunch Break

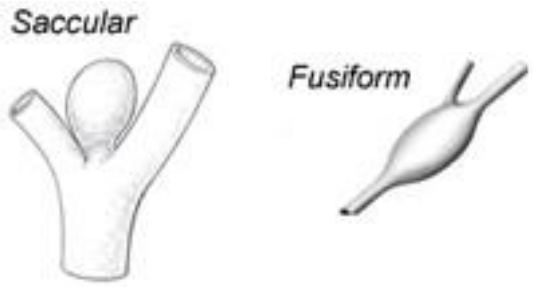
- **Defects in Practice**
- MPI Applications
- Threaded Applications

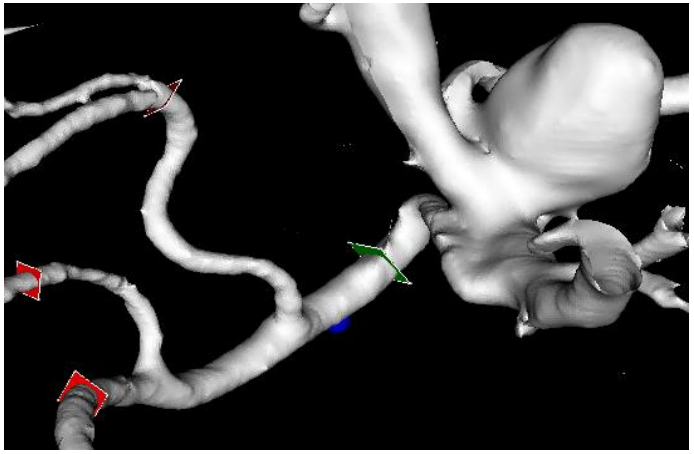
Coffee Break

- Hands-On II
- Beyond MPI and Threads
- Discussion and Wrap-Up

Defects in Practice
Story 1 – HPC could be brain surgery

- Brain aneurysms
  - 2-5% of population – most are undiagnosed
  - 30,000 rupture in US each year – 40% fatal
  - Early discovery and treatment increases survival rates
- Neurosurgery as HPC
  - MRI provides the blood vessel structure
  - Intra-cranial blood flow and pressures is just complex CFD
  - Full brain 3D model is 2-10GB geometry





Defects in Practice
Story 1 – Impact of Petascale and Beyond

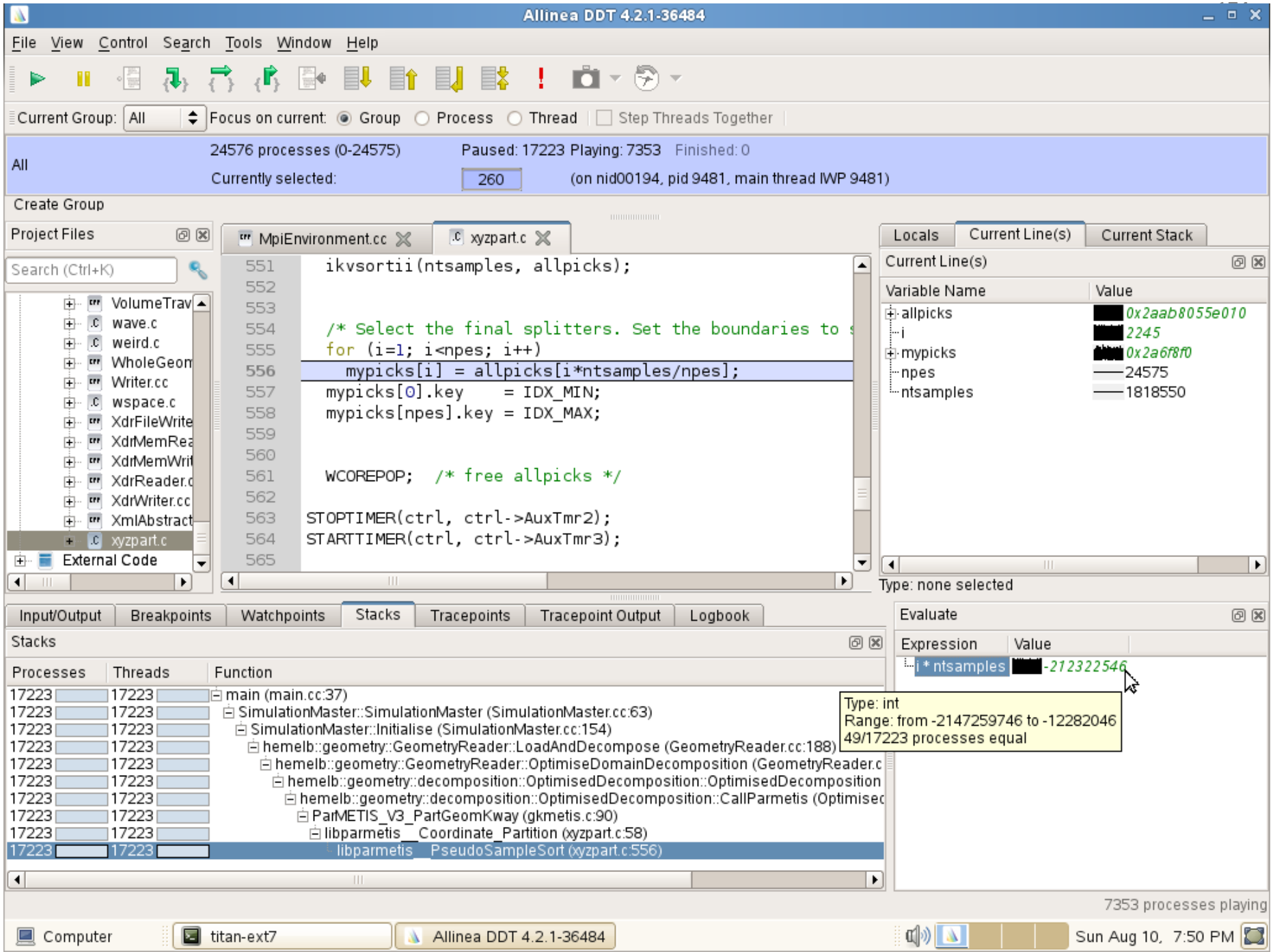
- Individualized HPC
  - Patient's MRI scan enables surgical decision: whether to operate, how to operate, …
  - Circle of Willis requires super-Petascale ~~machine~~ software
  - Need answer in minutes or hours
- Machines can do 20 PetaFLOPs
  - Super-Petascale will be affordable soon
  - Software has to scale

Defects in Practice
Story 1 – Real scaling challenge

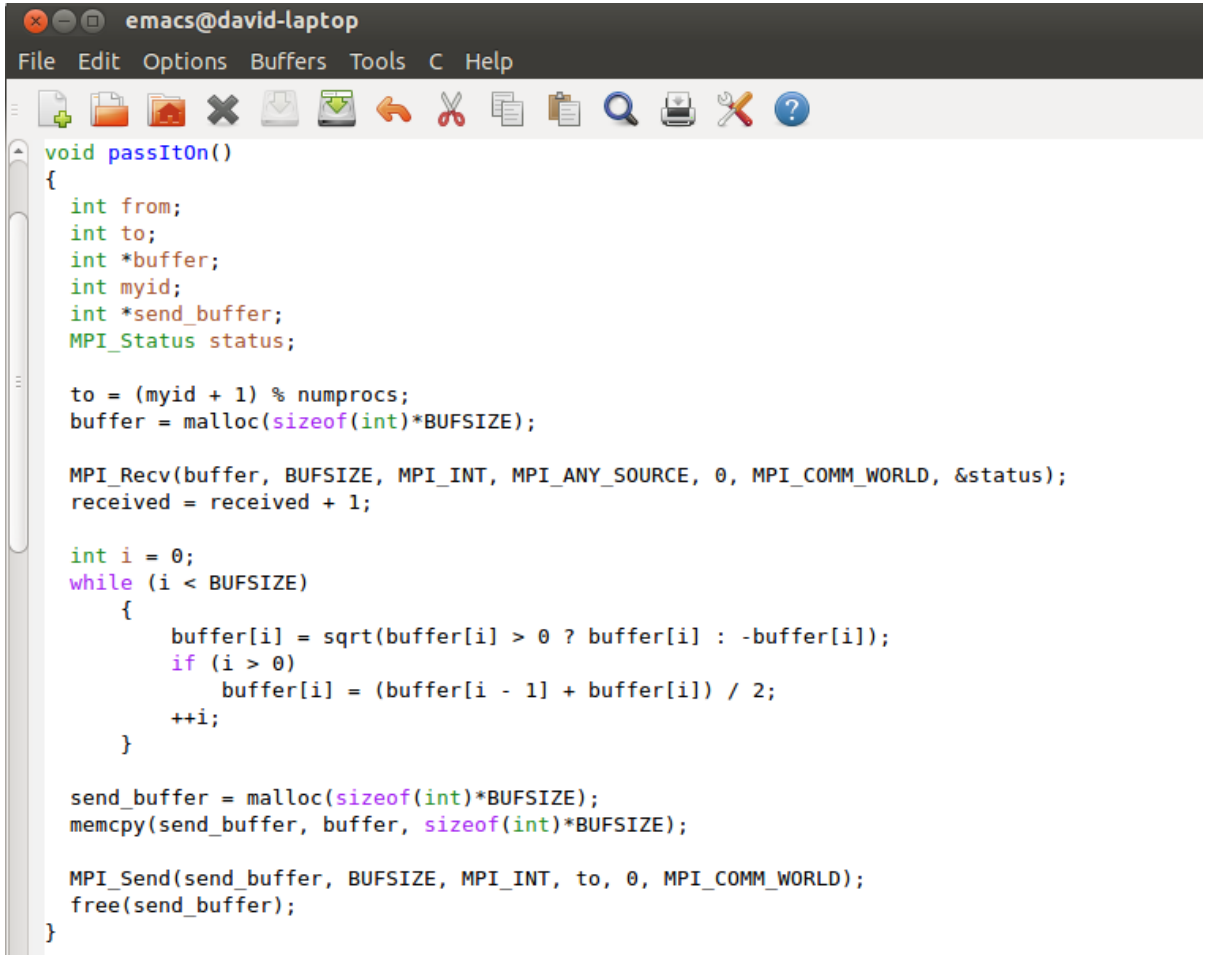
- Crashes at 49,152 cores on Cray XC30
  - Error message “Terminated”. Thanks.

- Now what?
  - Try other (inferior?) partitioner?
  - Invest weeks in bug fix by trial and error?
  - Write own partitioning library?

- Why use a debugger?
  - It’s about time


Allinea DDT 4.2.1-36484
File View Control Search Tools Window Help
Current Group: All Focus on current: Group Process Thread Step Threads Together
All
24576 processes (0-24575) Paused: 17223 Playing: 7353 Finished: 0
Currently selected: 260 (on nid00194, pid 9481, main thread IWP 9481)
Create Group
Project Files
Search (Ctrl+K)
VolumeTrav
wave.c
weird.c
WholeGeom
Writer.cc
wspace.c
XdrFileWrite
XdrMemRea
XdrMemWrit
XdrReader.c
XdrWriter.cc
XmlAbstract
xyzpart.c
External Code
MpiEnvironment.cc
xyzpart.c
551 ikvsortii(ntsamples, allpicks);
552
553
554 /* Select the final splitters. Set the boundaries to
555 for (i=1; i<npes; i++)
556 mypicks[i] = allpicks[i*ntsamples/npes];
557 mypicks[0].key = IDX_MIN;
558 mypicks[npes].key = IDX_MAX;
559
560
561 WCOREPOP; /* free allpicks */
562
563 STOPTIMER(ctrl, ctrl->AuxTmr2);
564 STARTTIMER(ctrl, ctrl->AuxTmr3);
565
Locals Current Line(s) Current Stack
Current Line(s)
Variable Name Value
allpicks 0x2aab8055e010
i 2245
mypicks 0x2a6f8f0
npes 24575
ntsamples 1818550
Type: none selected
Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook
Stacks
Processes Threads Function
17223 17223 main (main.cc:37)
17223 17223 SimulationMaster::SimulationMaster (SimulationMaster.cc:63)
17223 17223 SimulationMaster::Initialise (SimulationMaster.cc:154)
17223 17223 hemelb::geometry::GeometryReader::LoadAndDecompose (GeometryReader.cc:188)
17223 17223 hemelb::geometry::GeometryReader::OptimiseDomainDecomposition (GeometryReader.c
17223 17223 hemelb::geometry::decomposition::OptimisedDecomposition::OptimisedDecomposition
17223 17223 hemelb::geometry::decomposition::OptimisedDecomposition::CallParmetis (Optimised
17223 17223 ParMETIS_V3_PartGeomKway (gkmetis.c:90)
17223 17223 libparmetis__Coordinate_Partition (xyzpart.c:58)
17223 17223 libparmetis__PseudoSampleSort (xyzpart.c:556)
Evaluate
Expression Value
ntsamples -212322546
Type: int
Range: from -2147259746 to -12282046
49/17223 processes equal
7353 processes playing
Computer titan-ext7 Allinea DDT 4.2.1-36484 Sun Aug 10, 7:50 PM

Defects in Practice
Story 2 – Another real example

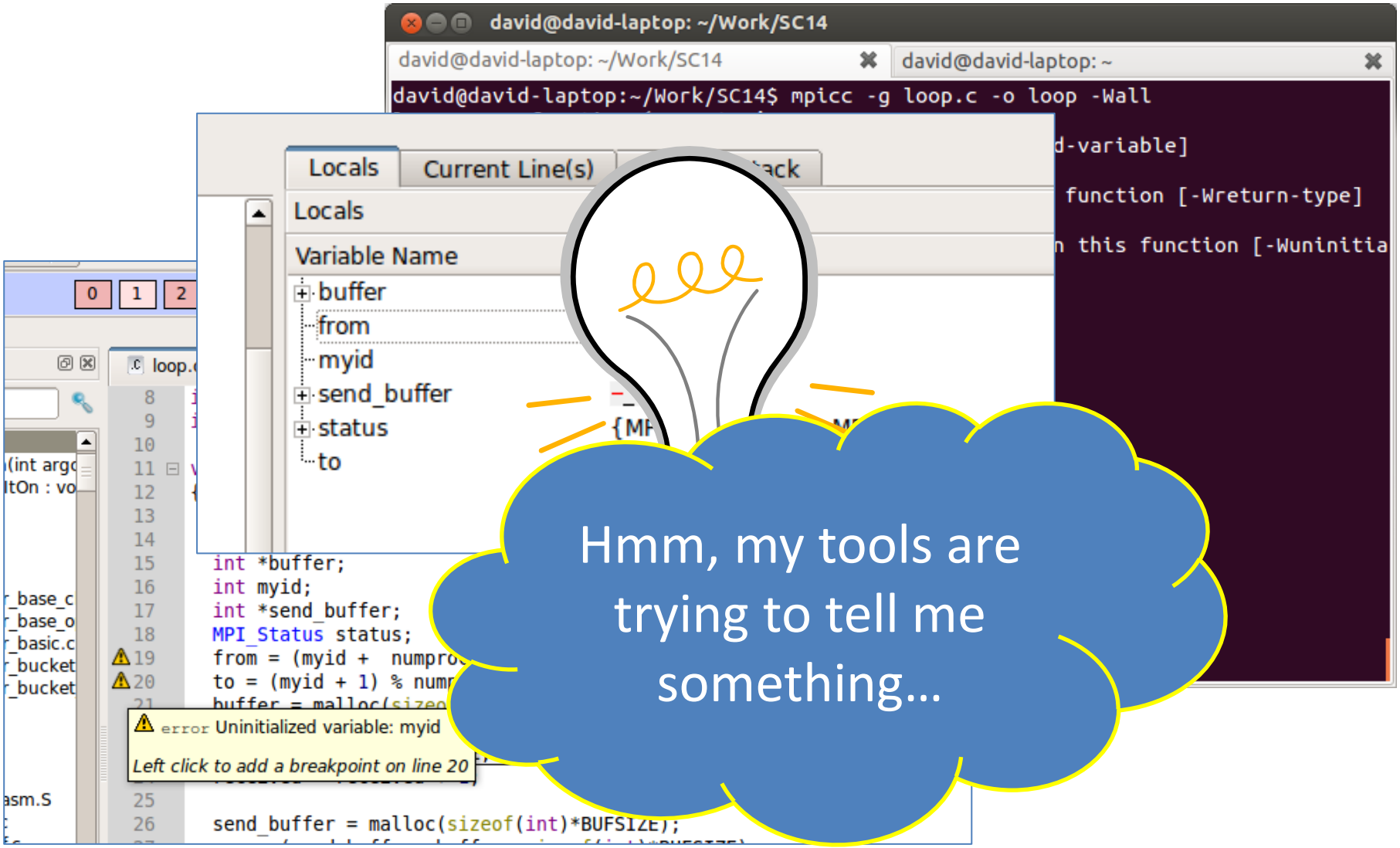



```c
void passItOn()
{
  int from;
  int to;
  int *buffer;
  int myid;
  int *send_buffer;
  MPI_Status status;

  to = (myid + 1) % numprocs;
  buffer = malloc(sizeof(int)*BUFSIZE);

  MPI_Recv(buffer, BUFSIZE, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);
  received = received + 1;

  int i = 0;
  while (i < BUFSIZE)
      {
          buffer[i] = sqrt(buffer[i] > 0 ? buffer[i] : -buffer[i]);
          if (i > 0)
              buffer[i] = (buffer[i - 1] + buffer[i]) / 2;
          ++i;
      }

  send_buffer = malloc(sizeof(int)*BUFSIZE);
  memcpy(send_buffer, buffer, sizeof(int)*BUFSIZE);

  MPI_Send(send_buffer, BUFSIZE, MPI_INT, to, 0, MPI_COMM_WORLD);
  free(send_buffer);
}
```

Defects in Practice
Story 2 – Two developers, one winner

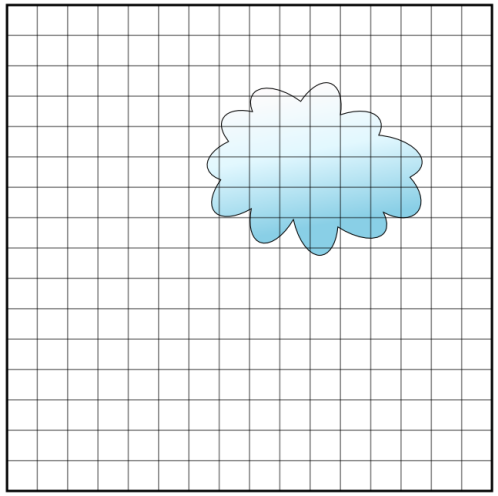
- Listen to the warnings… code always looks right
  - Head-scratching printf user – 2 hours 50 – no result
  - Tool user – 10 minutes – problem solved

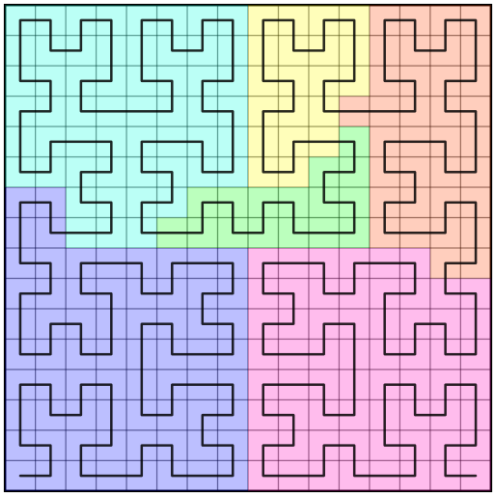
Defects in Practice
Story 2 – Listen to your inner voices… (or tools)



Defects in Practice
Story 3 – Control Flows at Scale

Dynamic load balancing Benchmark FD4 (*Development Version*):





Starting at 256 processes (BG/Q):

```
partbench: /.../AsyncLongGatherT.h:239: CCMI::Executor::Composite*
...
Assertion `co->getFlags() & EarlyArrival' failed.
2013-03-11 12:36:45.181 (WARN ) [0x400011c8b30]
:262848:ibm.runjob.client.Job: abnormal termination by signal 6 from
rank 127
```

Defects in Practice
Story 3 – MUST could Pinpoint

Rank 255 is in MPI_Bcast, others in MPI_Scatter

MUST Output, starting date: Fri Mar 15 12:57:54 2013.

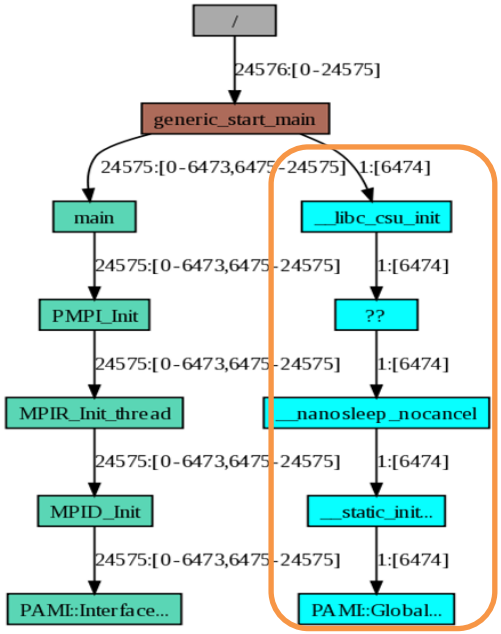
Rank	Type	Message	From	References
131	**Error**	A collective mismatch occured (The application executes two different collective calls on the same communicator)! The collective operation that does not matches this operation was executed at reference 1. (Information on communicator: Communicator created at reference 2 size=128) Note that collective matching was disabled as a result, collectives won't be analysed for their correctness or blocking state anymore. You should solve this issue and rerun your application with MUST.	**MPI_Scatter** called from: #0 MAIN__@partbench.F90:401 #1 main@partbench.F90:43	reference 1 rank 255: **MPI_Bcast** #0 __fd4_part_sfcd_mod_MOD_fd4_ #1 __fd4_balance_mod_MOD_fd4 #2 MAIN__@partbench.F90:357 #3 main@partbench.F90:43 reference 2 rank 131: **MPI_Comm** #0 __fd4_part_sfcd_mod_MOD_fd4_ #1 __fd4_balance_mod_MOD_fd4 #2 __fd4_util_mod_MOD_fd4_uti #3 MAIN__@partbench.F90:248 #4 main@partbench.F90:43

Root cause:

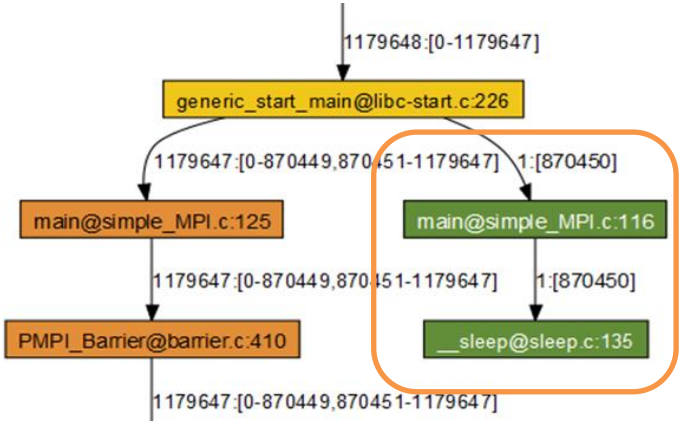
```
if (CONDITION)
then MPI_Scatter (...)
fi
```

Processes 0-254: TRUE
Process 255: FALSE

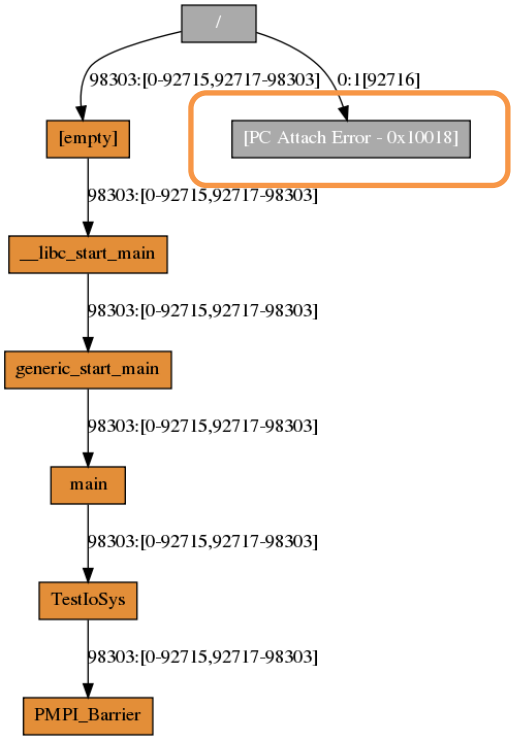
Defects in Practice
Stories 4-6 – STAT for bugs on Sequoia



A crash during init. due to a bad CN.



A hang due to a decrementor on a core running super slow at *over one million processes.*

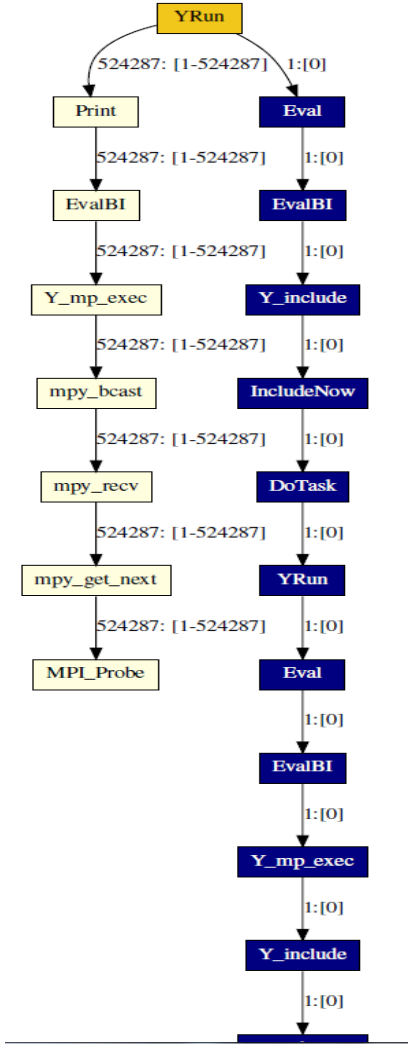


Hang due to a bug in LUSTRE using the full 96 BGQ racks.

92716: **bgqio199**

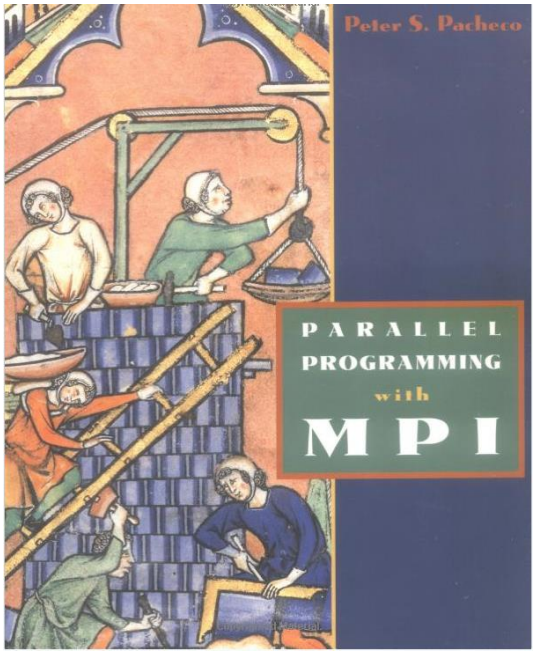
From Dong Ahn, LLNL-PRES-655004

Defects in Practice
Story 7 – STAT and Non-Determinism



- A hang only appeared only when pf3d is scaled to *half a million processes*.
- User refused to debug for *6 months*…
- Incorrect message mismatches due to *non-deterministic communication patterns*.
- Non-deterministic concurrency errors are increasingly common and painful.
- Demand for scalable—yet effective—techniques and tools for this class of errors.

Defects in Practice
Story 8 – Sometimes its in the Textbooks



Chapter 7: MPI_Comm_create example

ISP:

Rank 0: Resource leaks detected.
Rank 1: Resource leaks detected.
IS P detected no deadlocks!
Total Interleavings: 1

Defects in Practice
Story 9 – Pitfalls making serial code thread-safe

```
static double farg1,farg2;
#define FMAX(a,b) (farg1=(a),farg2=(b),farg1>farg2?farg1:farg2)
```

What could possibly go wrong?

To avoid side effects, the arguments are copied to temporary storage

Double checked scoping of variables: everything seems to be fine

```
1621:   T = FMAX(0.1111*foo*bar[x],THRESH);
```

What could possibly go wrong?

Archer flags a write-write race in line 1621

Debugging

"Is it hard? Not if you have the right attitudes. It's having the right attitudes that's hard."

Robert Pirsig

- *Learn what you need to know before you need to put it into practice.*
- *Having the right approach will become an instinct*

MPI Applications

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], and Joachim Protze[4]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Organization

- Introduction and Tools
- Hands-On I

Coffee Break

- The Tao of Debugging
- Systematic Debugging

Lunch Break

- Defects in Practice
- **MPI Applications**
- Threaded Applications

Coffee Break

- Hands-On II
- Beyond MPI and Threads
- Discussion and Wrap-Up

MPI Applications
Contents

- Common Types of MPI Errors
- Syntax Errors
- Semantic Errors
- MUST Details

MPI Applications
Common MPI Error Classes

- Common syntactic errors:
  - Incorrect arguments
  - Resource usage
  - Lost/Dropped Requests
  - Buffer usage
  - Type-matching
  - Deadlocks

Tool to use:
MUST,
ISP,
STAT,
(Allinea DDT)

- Semantic errors that are correct in terms of MPI standard, but do not match the programmers intent:
  - Displacement/Size/Count errors

Tool to use:
Allinea DDT, (STAT)

MPI Applications
Syntactic – Incorrect Arguments

- Incorrect arguments manifest during:
  - Compilation (Type mismatch)
  - Runtime (Crash in MPI or unexpected behavior)
  - During porting (Only manifests for some MPIs/systems)
- Example (C):

Fortran datatype in C

```
MPI_Send(
    buf,
    count,
    MPI_INTEGER,
    target,
    tag,
    MPI_COMM_WORLD);
```

MPI Applications
Syntactic – Resource Tracking Errors

- Many MPI features require resource allocations
  - Communicators
  - Data types
  - Requests
  - Groups, error handlers, reduction operations

- Simple “MPI_Op leak” example:

```
MPI_Op_create (..., &op);
MPI_Finalize ();
```

“MPI_Op_free (&op);” missing

MPI Applications
Syntactic – Dropped and Lost Requests

- Two resource errors with message requests
  - Leaked by creator (i.e., never completed)
  - Never matched by src/dest (dropped request)
- Simple “lost request” example:

```
MPI_Irecv (..., &req);
MPI_Irecv (..., &req);
MPI_Wait (&req,…)
```

Overwrites first request, its handle is now lost (and can thus not be completed)

MPI Applications
Syntactic – Buffer Usage Errors

- Complications:
  - Memory regions passed to MPI must not overlap (except send-send)
  - Derived datatypes can span non-contiguous regions
  - Collectives can both send and receive
- Example:

```
MPI_Isend (&(buf[0])/*buf*/, 5/*count*/, MPI_INT,...);
MPI_Irecv (&(buf[4])/*buf*/, 5/*count*/, MPI_INT,...);
```

Overlaps element buf[4] from the MPI_Isend call!

MPI Applications
Semantic – Example Displacement Errors

- Description:
  - Erroneous sizes, counts, or displacements
    - Example: During datatype construction or communication
    - Often “off-by-one” errors
- Example (C):

Stride must be 10 for a column

```
/* Create datatype to send [illegible] of 10x10 array */
MPI_Type_vector (
    10,                         /* #blocks */
    1,                          /* #elements per block */
    9,                          /* #stride */
    MPI_INT,                    /* old type */
    &newType                    /* new type */ );
```

MPI Applications
Complex MPI Errors

- MPI type matching
- MUST datatype positions
- MPI deadlocks
- ISP Replay

⇒Tools simplify removal of complex MPI datatype and deadlock issues

⇒ISP and MUST use advanced output here

MPI Applications
MPI Type Matching

- Three kinds of MPI type matching
  - Send buffer type and MPI send data type
  - MPI send type and MPI receive type
  - MPI receive type and receive buffer type
- Similar requirements for collective operations
- Buffer type <=> MPI type matching
  - Requires compiler support
  - MPI_BOTTOM, MPI_LB & MPI_UB complicates
  - Not provided by our tools

MPI Applications
Basic MPI Type Matching Example

- MPI standard provides support for heterogeneity
  - Endian-ness
  - Data formats
  - Limitations
- Simple example code:

```
Task 0                          Task 1
MPI_Send(1, MPI_INT)MPI_Recv(8, MPI_BYTE)
```

- Do the types match?
  - **Buffer type <=> MPI type: Yes**
  - **MPI send type <=> MPI receive type?**
    - **NO!**
    - **Common misconception**

MPI Applications
Derived MPI Type Matching Example

- Consider MPI derived types corresponding to:
  - T1: struct {double, char}
  - T2: struct {double, char, double}
- Do these types match?

- **Example 1:**

```
Task 0                    Task 1
MPI_Send(1, T1)    MPI_Recv(1, T2)
```

- **Yes: MPI supports partial receives**
  - **Allows efficient algorithms**
  - **double <=> double; char <=> char**

MPI Applications
Derived MPI Type Matching Example

- Consider MPI derived types corresponding to:
  - T1: struct {double, char}
  - T2: struct {double, char, double}
- Do these types match?

- **Example 2:**

```
Task 0                    Task 1
MPI_Send(1, T2)    MPI_Recv(2, T1)
```

- **Yes:**
  - **double <=> double**
  - **char <=> char**
  - **double <=> double**

MPI Applications
Derived MPI Type Matching Example

- Consider MPI derived types corresponding to:
  - T1: struct {double, char}
  - T2: struct {double, char, double}
- Do these types match?

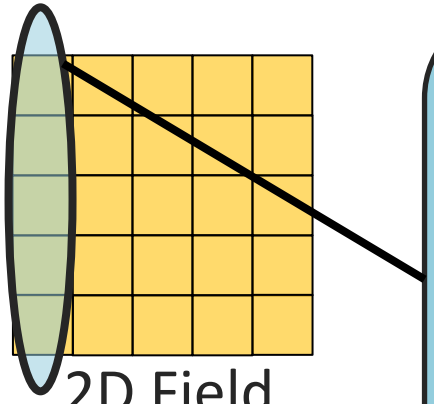
- **Example 3:**

```
Task 0                  Task 1
MPI_Send(2, T1)    MPI_Recv(2, T2)
MPI_Send(2, T2)    MPI_Recv(4, T1)
```

- **No! What happens? Nothing good!**

MPI Applications
MUST Datatype Positions

- Derived datatypes use constructors, example:



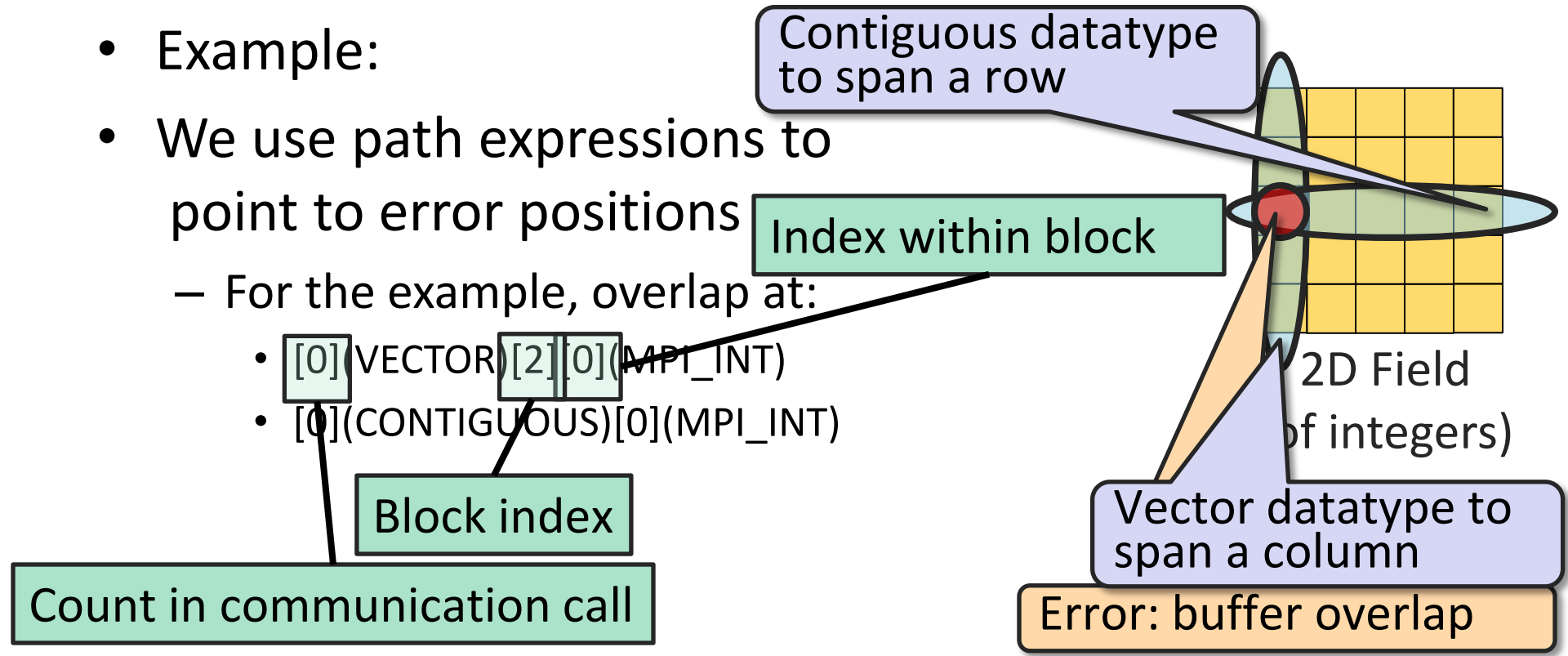
2D Field
(of integers)

```
MPI_Type_vector (
  NumRows          /*count*/,
  1                /*blocklength*/,
  NumColumns       /*stride*/,
  MPI_INT          /*oldtype*/,
  &newType);
```

- Errors that involve datatypes can be complex:
  - Need to be detected correctly
  - Need to be visualized

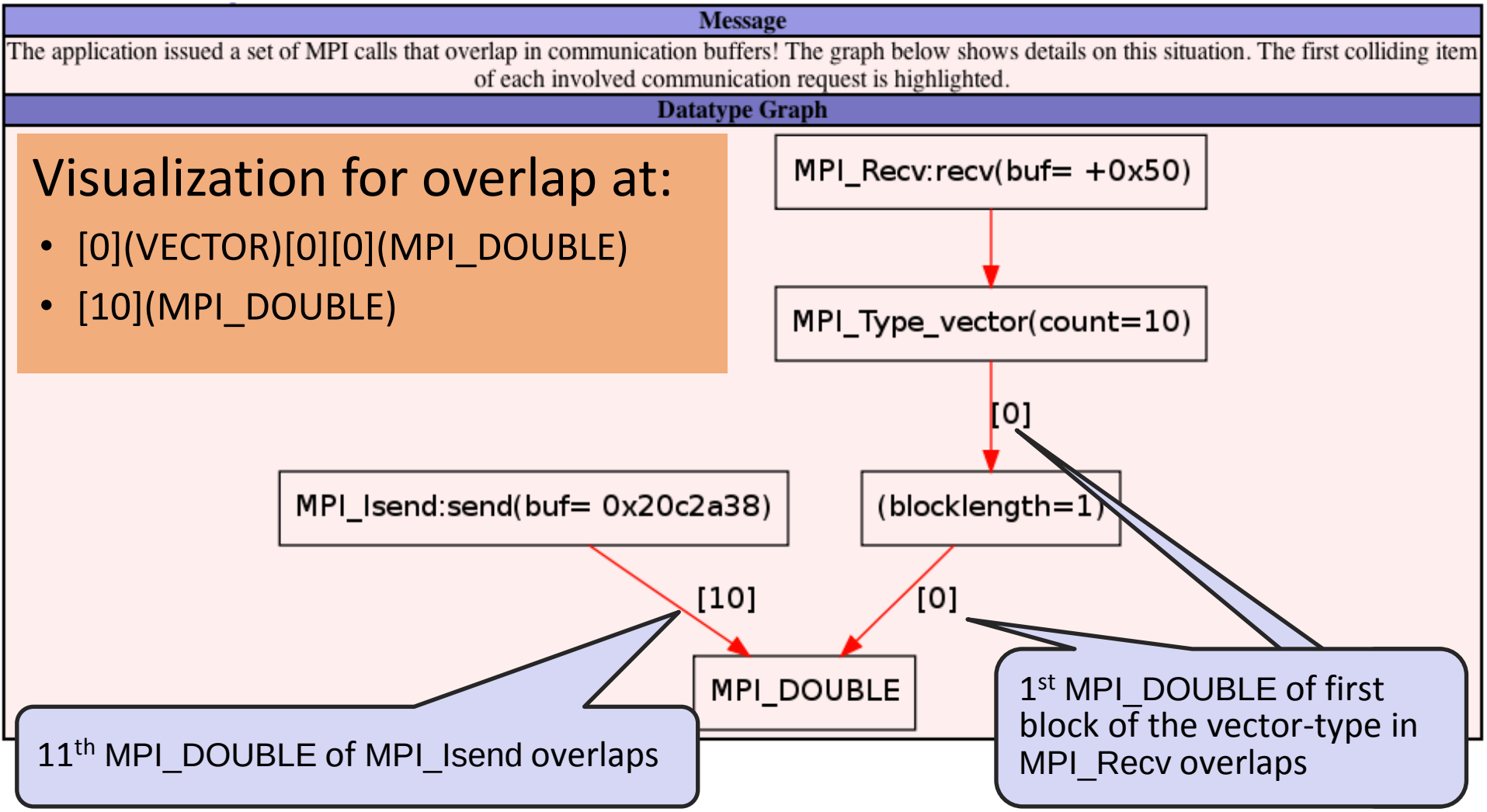
MPI Applications
MUST Datatype Positions (2)

- How to point to an error in a derived datatype?
  - Derived types can span wide areas of memory
  - Understanding errors requires precise knowledge
  - E.g., not sufficient: Type X overlaps with type Y
- Example:
- We use path expressions to point to error positions
  - For the example, overlap at:
    - [0](VECTOR)[2][0](MPI_INT)
    - [0](CONTIGUOUS)[0](MPI_INT)



MPI Applications
MUST Datatype Positions (3)

Syntax Errors | Semantic Errors | Datatype Issues | Deadlocks | Non-Determinism



MPI Applications
Basic MPI Deadlocks

- Unsafe or erroneous MPI programming practices
- Code results depend on:
  - MPI implementation limitations
  - User input parameters
- Classic example code:

```
Task 0          Task 1
MPI_Send        MPI_Send
MPI_Recv        MPI_Recv
```

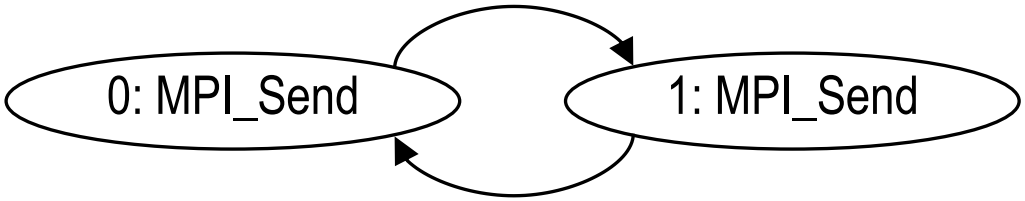
- Assume application uses “Thread funneled”

MPI Applications
Deadlock Visualization

- How to visualize/analyze deadlocks?
  - Common approach waiting-for graphs (WFGs)
  - One node for each rank
  - Rank X waits for rank Y => node X has an arc to node Y
- Consider the previous example:

```
Task 0          Task 1
MPI_Send        MPI_Send
MPI_Recv        MPI_Recv
```

- Visualization:



- Deadlock criterion: cycle (For simple cases)

MPI Applications
Deadlocks with MPI Collectives

- Erroneous MPI programming practice
- Simple example code:

Tasks 0, 1, & 2	Task 3
MPI_Bcast	MPI_Barrier
MPI_Barrier	MPI_Bcast

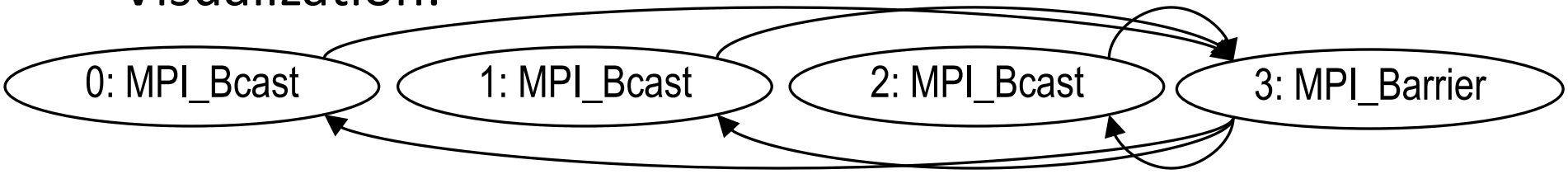
- Possible code results:
  - Deadlock
  - Correct message matching
  - Incorrect message matching
  - Mysterious error messages
- “Wait-for” every task in communicator

MPI Applications
Deadlock Visualization – Collectives

- What about collectives?
  - Rank calling collective waits for all tasks to issue a matching call
  - One arc to each task that did not call a matching call
- Consider the previous example:

Tasks 0, 1, & 2	Task 3
MPI_Bcast	MPI_Barrier
MPI_Barrier	MPI_Bcast

- Visualization:



- Deadlock criterion: still a cycle

MPI Applications
Deadlocks with Non-Determinism

- What about “MPI_ANY_SOURCE”, “any”, and “some”?
  - MPI_Waitany/Waitsome and wild-card source receives (MPI_ANY_SOURCE) have special semantics
  - Wait for at least one of a set of ranks but not necessarily all
  - Different from “waits for all” semantic
- Example:

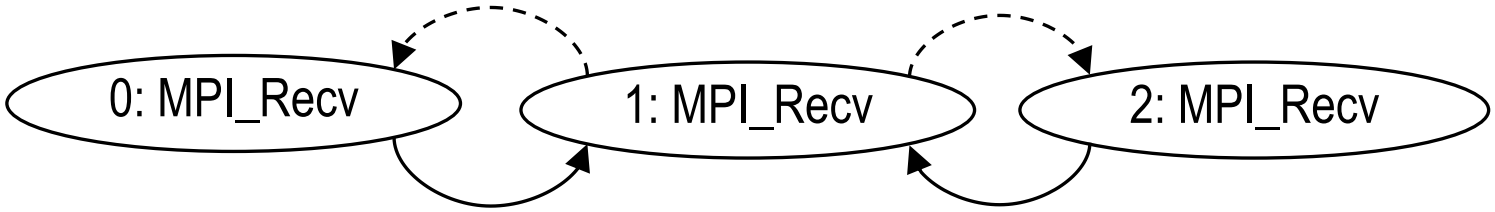
```
Tasks 0           Task 1                Task 2
MPI_Recv(from:1) MPI_Recv(from:ANY) MPI_Recv(from:1)
```

- What happens:
  - No call can progress, Deadlock
  - 0 waits for 1; 1 waits for either 0 or 1; 2 waits for 1

MPI Applications
Deadlock Visualization – Non-Determinism

- How to visualize the “any/some” semantics?
  - “Waits for all of” wait type => “AND” semantic
  - “Waits for any of” wait type => “OR” semantic
  - Each type gets one type of arc
    - AND: solid arcs
    - OR: Dashed arcs
- MUST provides a visualization and a criterion:

```
Tasks 0                Task 1                    Task 2
MPI_Recv(from:1) MPI_Recv(from:ANY) MPI_Recv(from:1)
```



MPI Applications
Schedule Dependent Deadlocks

- What about more complex interleavings?
  - Non-deterministic applications
  - Interleaving determines what calls match or are issued
  - Can manifest as bugs that only occur “sometimes”
- Example:

```
Tasks 0            Task 1                  Task 2
                   MPI_Recv(from:ANY)
MPI_Send(to:1)     MPI_Recv(from:2)        MPI_Send(to:1)
MPI_Barrier        MPI_Barrier             MPI_Barrier
```

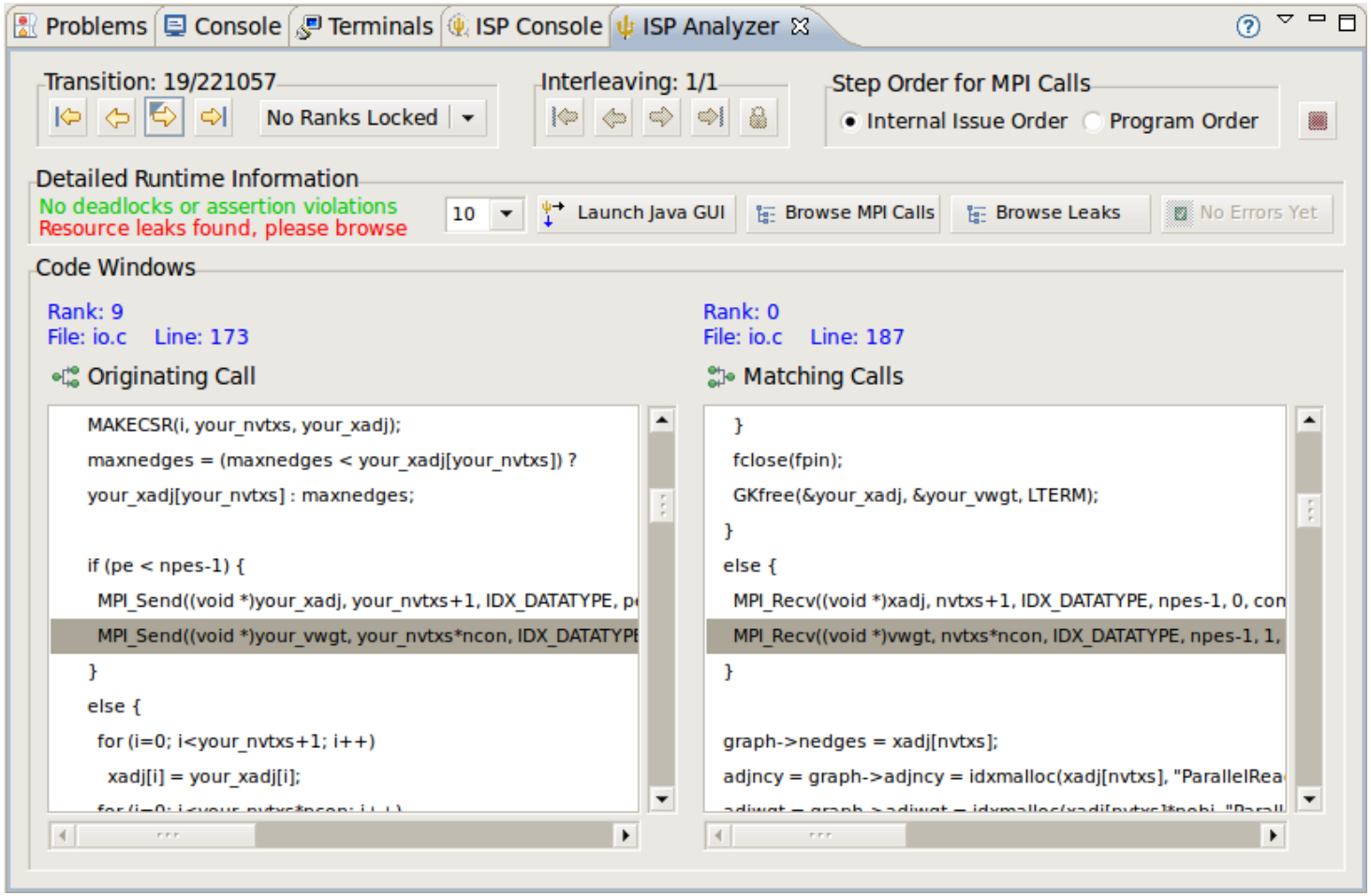
- What happens:
  - Case A:
    - Recv (from:ANY) matches send from task 0
    - All calls complete
  - Case B:
    - Recv (from:ANY) matches send from task 1
    - Deadlock

MPI Applications
Complementary Tool – ISP (recap)

- ISP (In-situ Partial Order) detects MPI usage errors
- Replay based deadlock detection:
  - Forces non-determinism coverage
  - Eliminates redundant tests

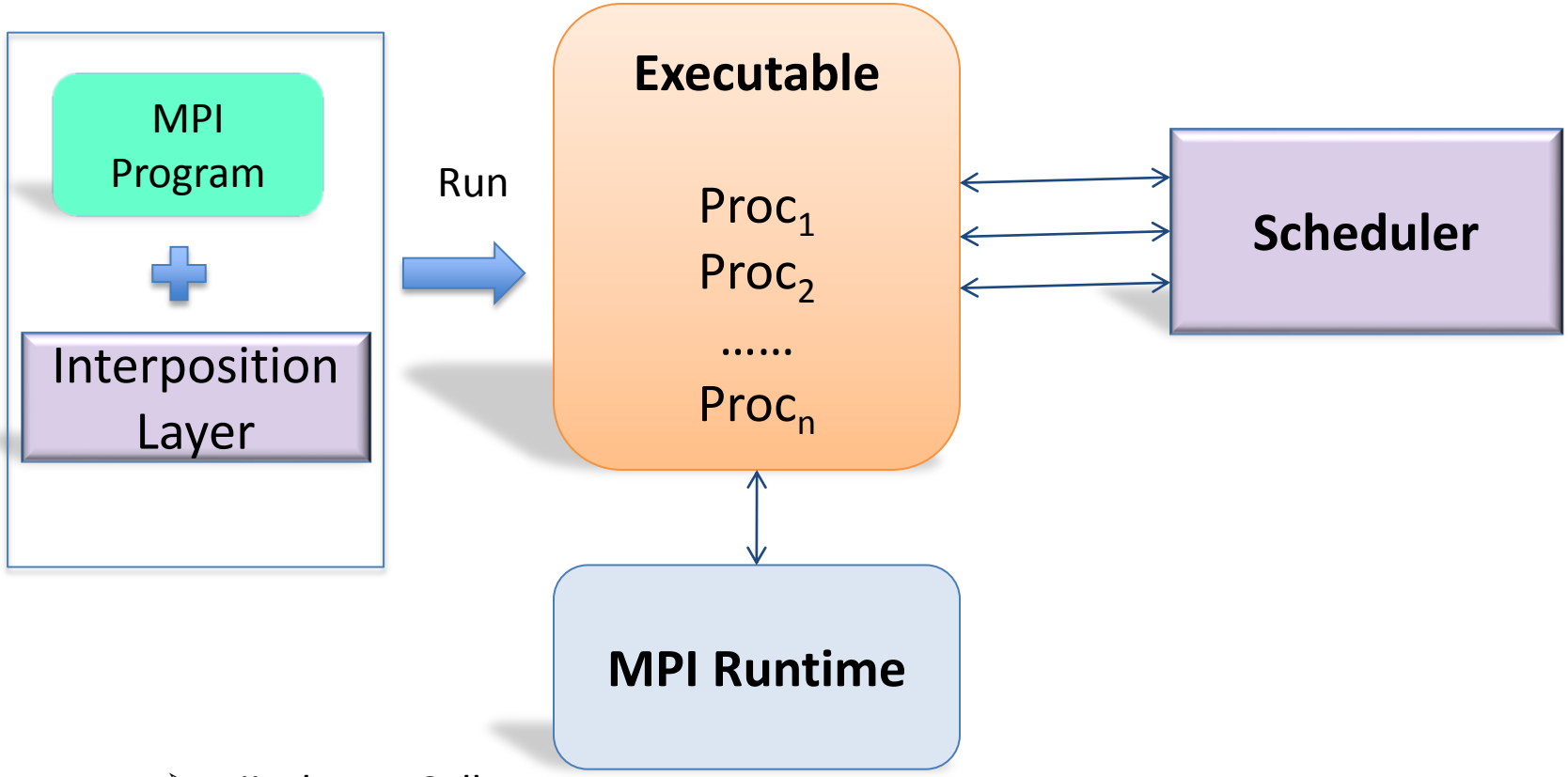
  $\Rightarrow$Detects rare deadlocks (Even ones never manifested)
- Reports errors with:
  - Java based GUI or
  - GEM an Eclipse PTP component
- DAMPI: distributed and scalable version of ISP

MPI Applications
Complementary Tool – ISP, GUI Example (GEM)



MPI Applications
ISP Detects Schedule Dependent Deadlocks

Syntax Errors | Semantic Errors | Datatype Issues | Deadlocks | Non-Determinism



- Hijack MPI Calls
- Scheduler decides how they are sent to the MPI runtime
- Scheduler plays out only the RELEVANT interleavings

Background II: ISP – Meeting all these goals is difficult!

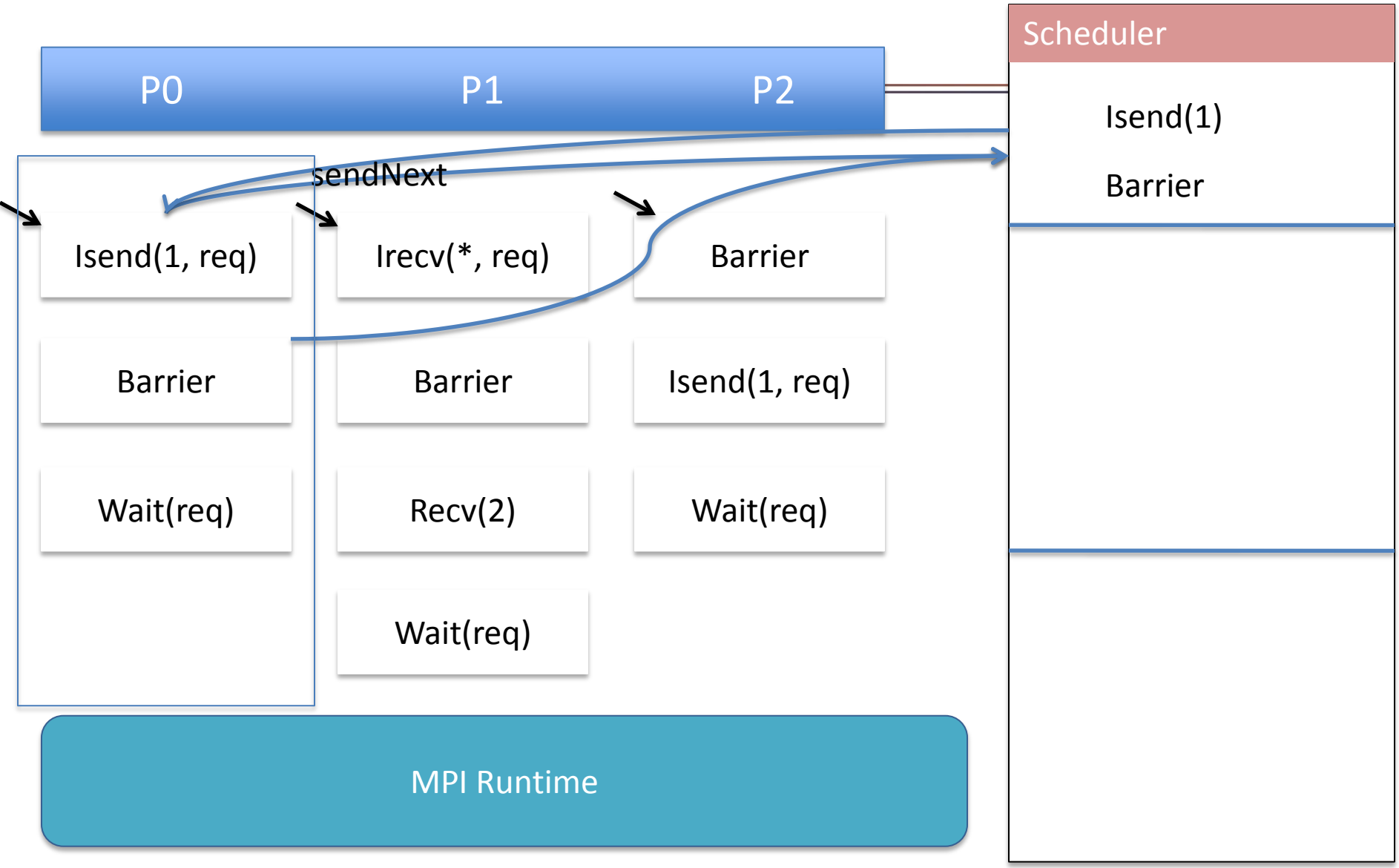
Syntax Errors | Semantic Errors | Datatype Issues | Deadlocks | Non-Determinism

- ISP automatically replays all possible interleavings of an application
- Tracks happens-before relation to determine all possible matches of each MPI calls
- Each process coordinates with a central scheduler
- Particular interleavings are enforced by rewriting MPI calls

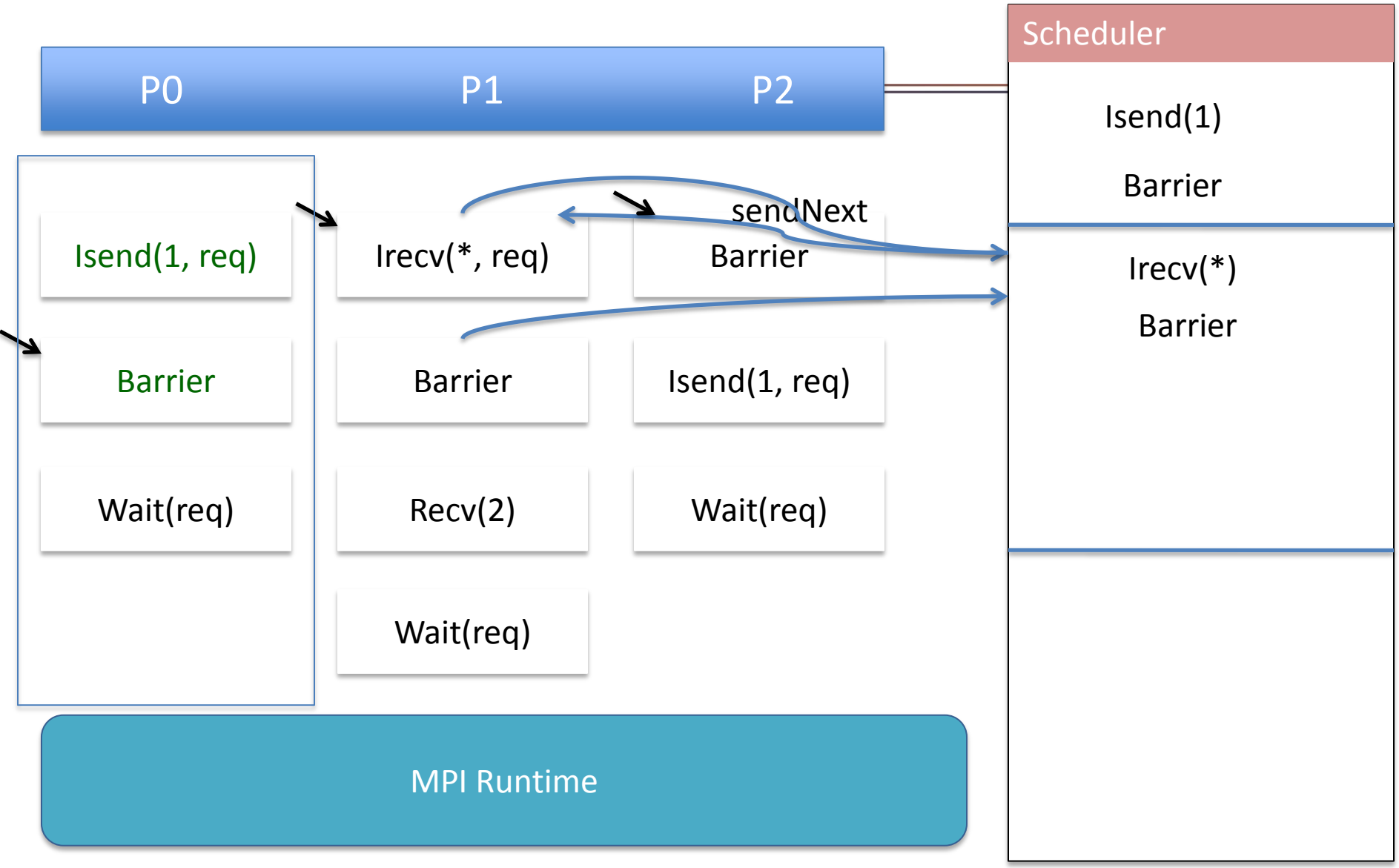
MPI Applications
ISP – Scheduler, Example Code

Process P0	Process P1	Process P2
Isend(1, req) ;	Irecv(*, req) ;	Barrier ;
Barrier ;	Barrier ;	Isend(1, req) ;
Wait(req) ;	Recv(2) ;	Wait(req) ;
	Wait(req) ;	

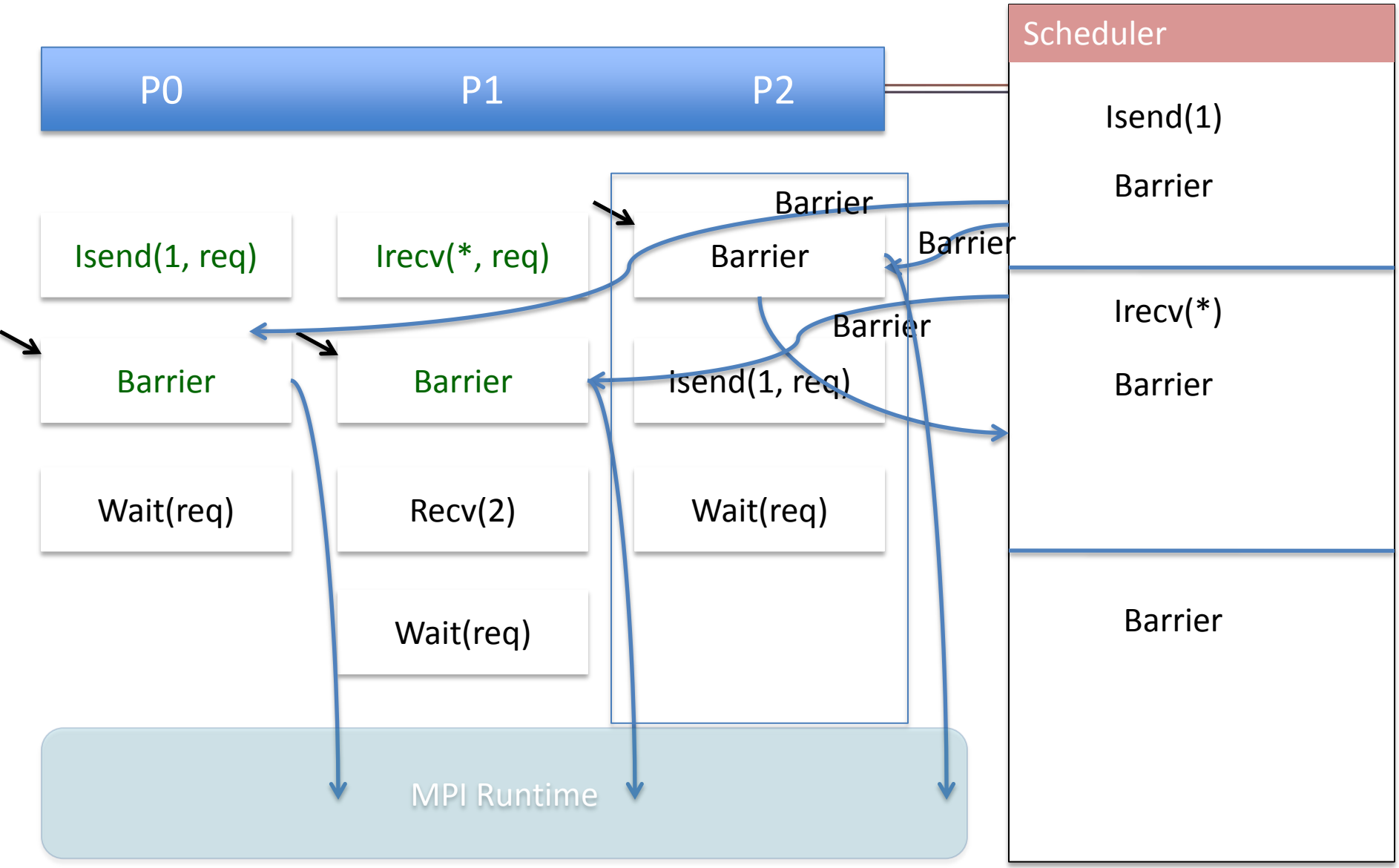
MPI Applications: ISP Scheduler Actions (animation)



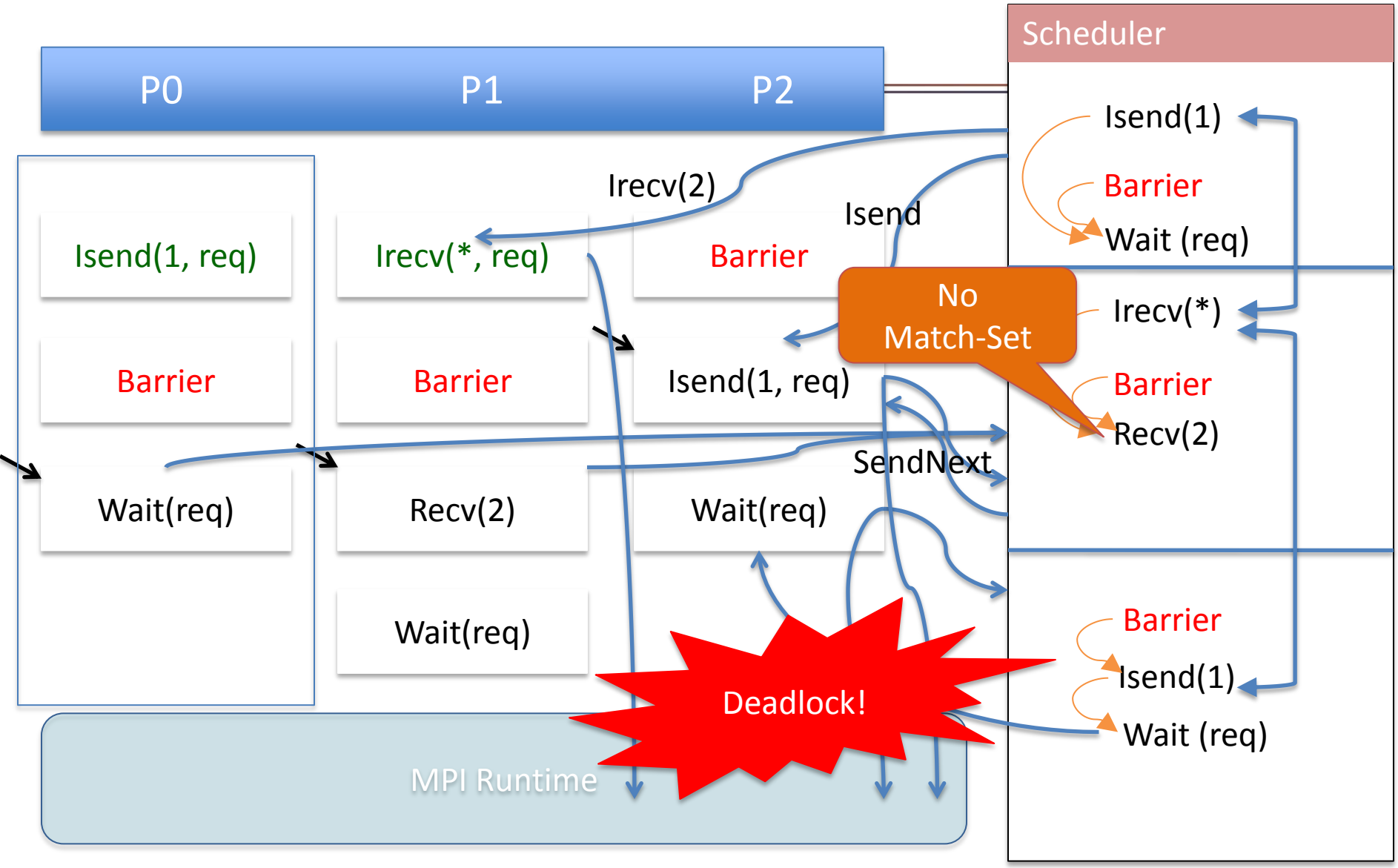
MPI Applications: ISP Scheduler Actions (animation)



MPI Applications: ISP Scheduler Actions (animation)



MPI Applications: ISP Scheduler Actions (animation)

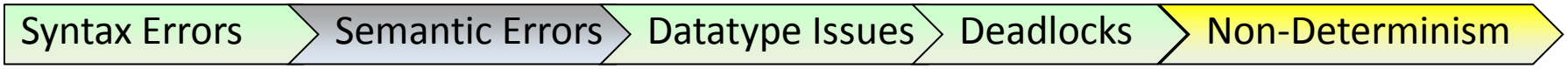


MPI Applications
Basic Usage of ISP

- Compile + link application with compiler wrapper
  - Replace “mpicc” with “**ispcc**”
- Replace “mpiexec” with “**isp**” (add “**-l trace.out**”)
  - E.g.: isp –l trace.out –n 4 myApp.exe
- Inspect deadlocks with “**ispUI trace.out**”

MPI Applications
Tool Who-is-Who

- MUST:



- ISP:

- STAT:

- DDT:

Threaded Applications

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], Joachim Protze[4] and Simone Atzeni[1]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Booth #2143

Booth #1508

Booth #3624

Booth #1030

Organization

- Introduction and Tools
- Hands-On I

Coffee Break

- The Tao of Debugging
- Systematic Debugging

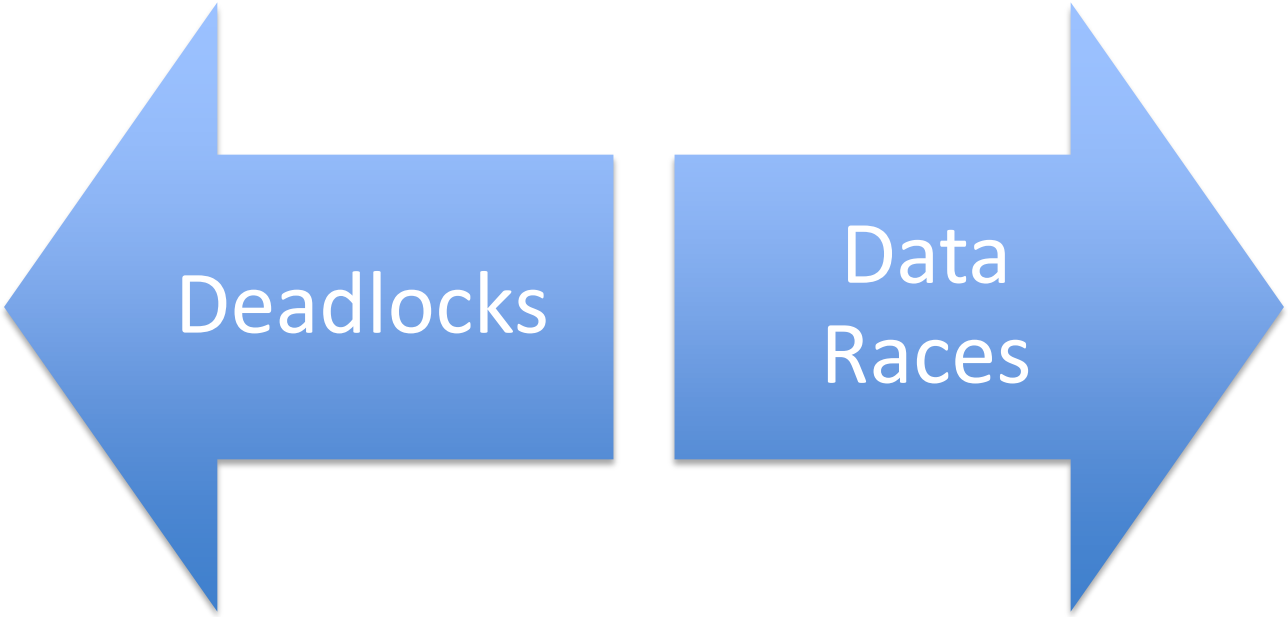
Lunch Break

- Defects in Practice
- MPI Applications
- **Threaded Applications**

Coffee Break

- Hands-On II
- Beyond MPI and Threads
- Discussion and Wrap-Up

Threaded Applications (Focus: OpenMP)
Threaded Defects

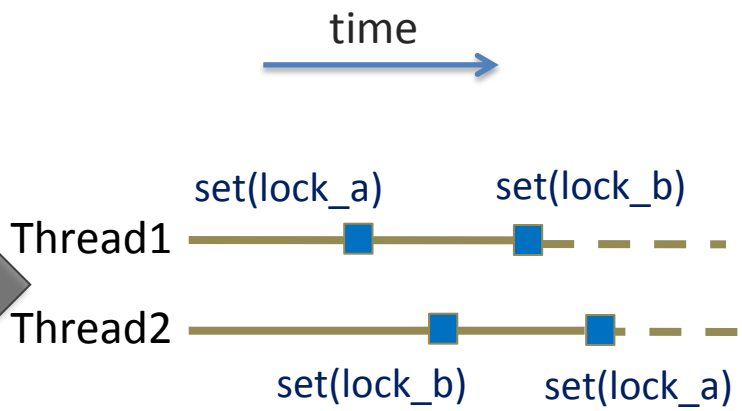


Threaded Applications (Focus: OpenMP)
Threaded Defects – Deadlock

A circular wait condition exists in the system that causes two or more parallel units to wait indefinitely

```
#pragma omp parallel sections
{
    #pragma omp section
    {
        omp_set_lock(&lock_a);
        omp_set_lock(&lock_b);
        omp_unset_lock(&lock_b);
        omp_unset_lock(&lock_a);
    }
    #pragma omp section
    {
        omp_set_lock(&lock_a);
        omp_set_lock(&lock_b);
        omp_unset_lock(&lock_b);
        omp_unset_lock(&lock_a);
    }
}
```

Deadlocking Execution Order



- Thread 1 waits for lock_b owned by thread 2
- Thread 2 waits for lock_a, owned by Thread 1.
- Neither thread can free a lock and both threads wait indefinitely.

Threaded Applications (Focus: OpenMP)
Threaded Defects – Data Race

Program behavior dependent on execution order of threads/processes

```
int x,y;
#pragma omp parallel
{
        x = omp_get_thread_num ();
        #pragma omp barrier
        #pragma omp master
                printf (“Master is:%d” ,x);
}
```

A write-write race on x

```
int x,y;
#pragma omp parallel
{
        #pragma omp master
                sleep(5);
        x = omp_get_thread_num ();
        #pragma omp barrier
        #pragma omp master
                printf (“Master is:%d” ,x);
}
```

If the master thread is intended to write x, it will usually do so, due to the sleep; But sometimes it may not …

Threaded Applications (Focus: OpenMP) Definitions

- **Deadlock**
  - Two or more threads are waiting for each other to unlock a synchronized block of code, where each thread relies on another to do the unlocking
  - Program hangs
  - May be non-deterministic
- **Data race**
  - Two threads access the same shared variable
    - at least one thread modifies the variable
    - the accesses are concurrent, i.e. unsynchronized
  - Leads to non-deterministic behavior
  - Hard to find with traditional debugging tools

Threaded Applications (Focus: OpenMP) Why is finding data races important?

Data races affect several fields:

- **File systems**
  - Two or more programs may "collide" in their attempts to modify or access a file, which could result in data corruption
- **Security**
  - Security vulnerability because of changes in a system between the checking of a condition (such as a security credential) and the use of the results of that check
- **Life-critical systems**
  - Flaws in the Therac-25 radiation therapy machine in 1985 (death of three patients and injuries to several more)
  - North American Blackout of 2003, the Energy Management System provided by GE Energy did not raise the alarm delaying the awareness of some problem

Threaded Applications
Threaded Defects – Data Race 1

Based on slides from Rolf Rabenseifner and Matthias Lieber

– Detected data race: write at line 71, read at line 72

```
68  #pragma omp parallel for
69    for (i=1; i<N; i++)
70    {
71      x = sqrt(b[i]) - 1;
72      a[i] = x*x + 2*x + 1;
73    } /* end of omp parallel for */
```

– What's wrong? How to solve?

Solution: (see appendix)

Threaded Applications
Threaded Defects – Data Race 2

Based on slides from Rolf Rabenseifner and Matthias Lieber

– Detected data race: write at line 49, read at line 54

```
43    a[0] = 0;
44  #pragma omp parallel
45    {
46  #pragma omp for nowait
47      for (i=1; i<N; i++)
48      {
49        a[i] = 3.0*i*(i+1);
50      } /* end of omp for nowait */
51  #pragma omp for
52      for (i=1; i<N; i++)
53      {
54        b[i] = a[i] - a[i-1];
55      } /* end of omp for */
56    } /* end of omp parallel */
```

– What's wrong? How to solve?

Solution: (see appendix)

Threaded Applications
Threaded Defects – Data Race 3

Based on slides from Rolf Rabenseifner and Matthias Lieber

– Detected data race: write at line 29, read at line 30

```
25    a[0] = 0;
26  #pragma omp parallel for
27    for (i=1; i<N; i++)
28    {
29      a[i] = 2.0*i*(i-1);
30      b[i] = a[i] - a[i-1];
31    } /* end of omp parallel for */
```

– What's wrong? How to solve?

Solution:

(see appendix)

Threaded Applications
Threaded Defects – Data Race 4

Based on slides from Rolf Rabenseifner and Matthias Lieber

– **Not detected by Intel Inspector XE!**

```
84    f = 2;
85  #pragma omp parallel for private(f,x)
86    for (i=1; i<N; i++)
87    {
88      x = f * b[i];
89      a[i] = x - 7;
90    } /* end of omp parallel for */
91    a[0] = x;
```

– What's wrong? How to solve?

Solution:

(see appendix)

– Predecessor Intel Thread Checker detected this error

Threaded Applications
Threaded Defects – Data Race 5

Based on slides from Rolf Rabenseifner and Matthias Lieber

– Detected data race: write at line 105, read at line 105

```
101   sum = 0;
102 #pragma omp parallel for
103   for (i=1; i<N; i++)
104   {
105     sum = sum + b[i];
106   } /* end of omp parallel for */
```

– What's wrong? How to solve?

Solution: (see appendix)

Threaded Applications
Threaded Defects – Solutions (for print version)

Based on slides from Rolf Rabenseifner and Matthias Lieber

Example 3: Add **private(x)** at line 68

Example 2: Remove **nowait** at line 46

Example 1:
Two separate loops (will cause bad cache reuse)
or
Recompute **a[i-1]**: **b[i] = a[i] - 2.0*(i-1)*(i-2);**

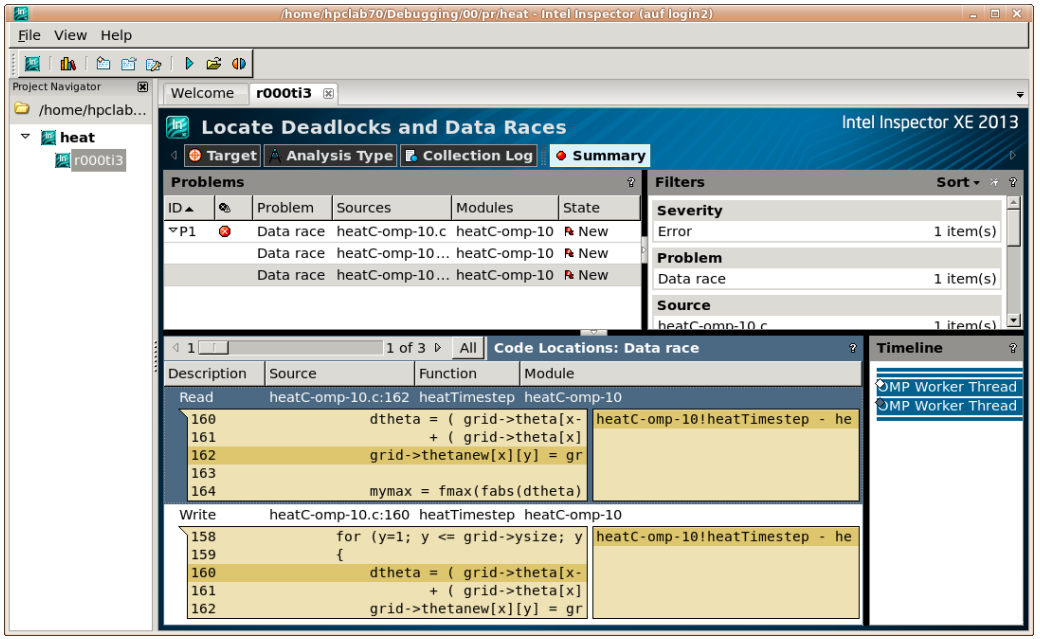
Example 4: use **firstprivate(f) lastprivate(x)** instead
private(f,x) at line 85

Example 5: Add **reduction(+:sum)** at line 102

Threaded Applications
Intel Inspector XE

Based on slides from Rolf Rabenseifner and Matthias Lieber

- Error checking tool for
  - Memory errors
  - **Threading errors** (OpenMP, Pthreads, Intel TBB)
- Available for Linux and Windows
- Supports C, C++, C#, F#, Fortran
- Graphical user interface
- Successor of Intel Thread Checker
- More info: http://software.intel.com/en-us/intel-inspector-xe



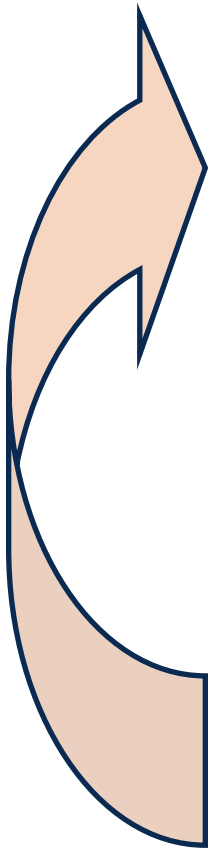
Threaded Applications Inspector XE – Limitations

Based on slides from Rolf Rabenseifner and Matthias Lieber

- Runtime check
  - Error detection only in software branches that are executed
- Very high memory and runtime overhead
  - Roughly 10x - 100x
  - Use small number of iterations
  - Use small data set
  - But large and complex enough that all software branches are executed

Threaded Applications Inspector XE – Usage

Based on slides from Rolf Rabenseifner and Matthias Lieber



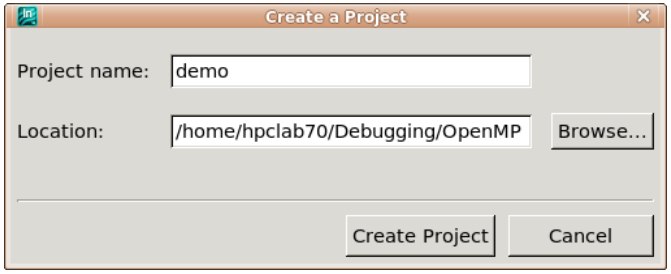
- Compile the program with the **–g** compiler flag
  - **`icc -g -openmp myprog.c -o myprog`**
- Run the program under control of Intel Inspector XE
  - **`export OMP_NUM_THREADS=...`**
    `inspxe-gui`
  - Detects problems only in software branches that are executed
- Understand and correct the threading errors detected
- Edit the source code
- Repeat until no errors reported

Note: Intel recommends compiling with **–O0** to prevent optimizations interfering with debug information, but also recommends to check the optimized production binary.

Threaded Applications Inspector XE – Create a Project

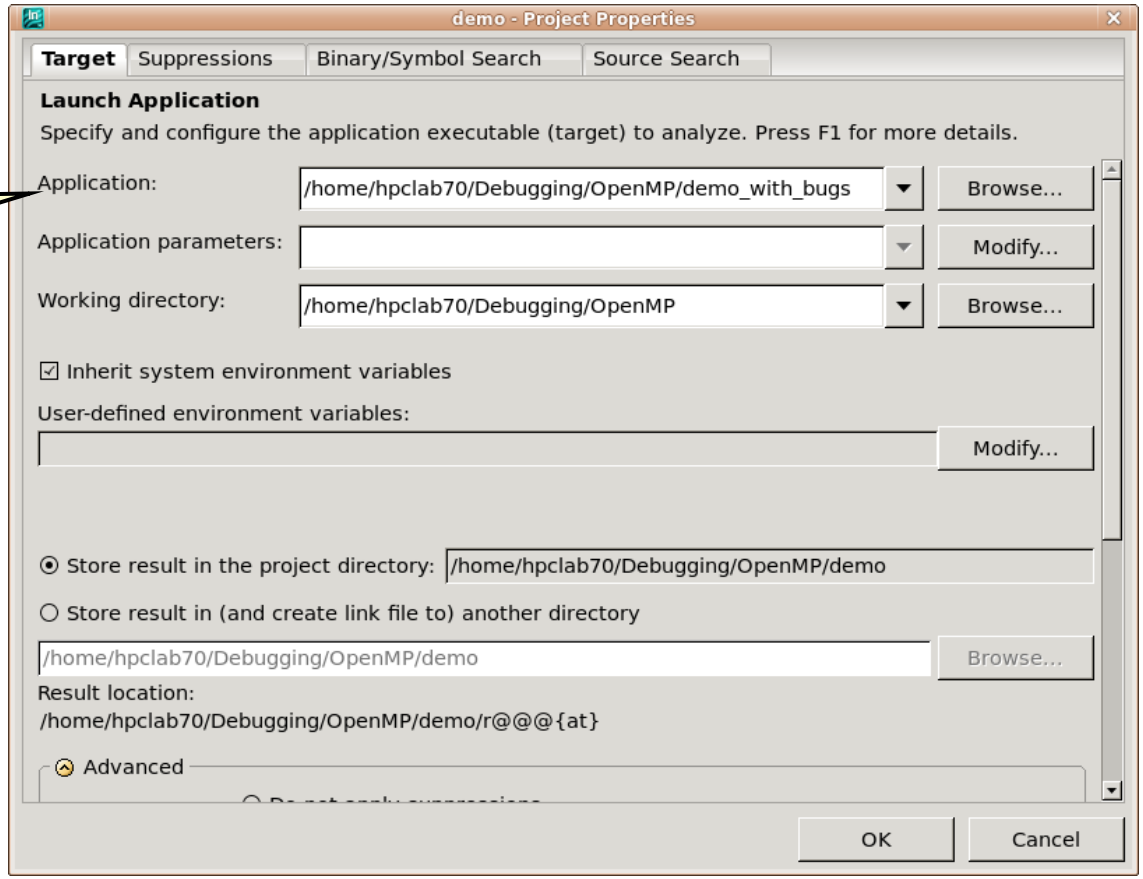
Based on slides from Rolf Rabenseifner and Matthias Lieber

Click „New Project“ in the start screen



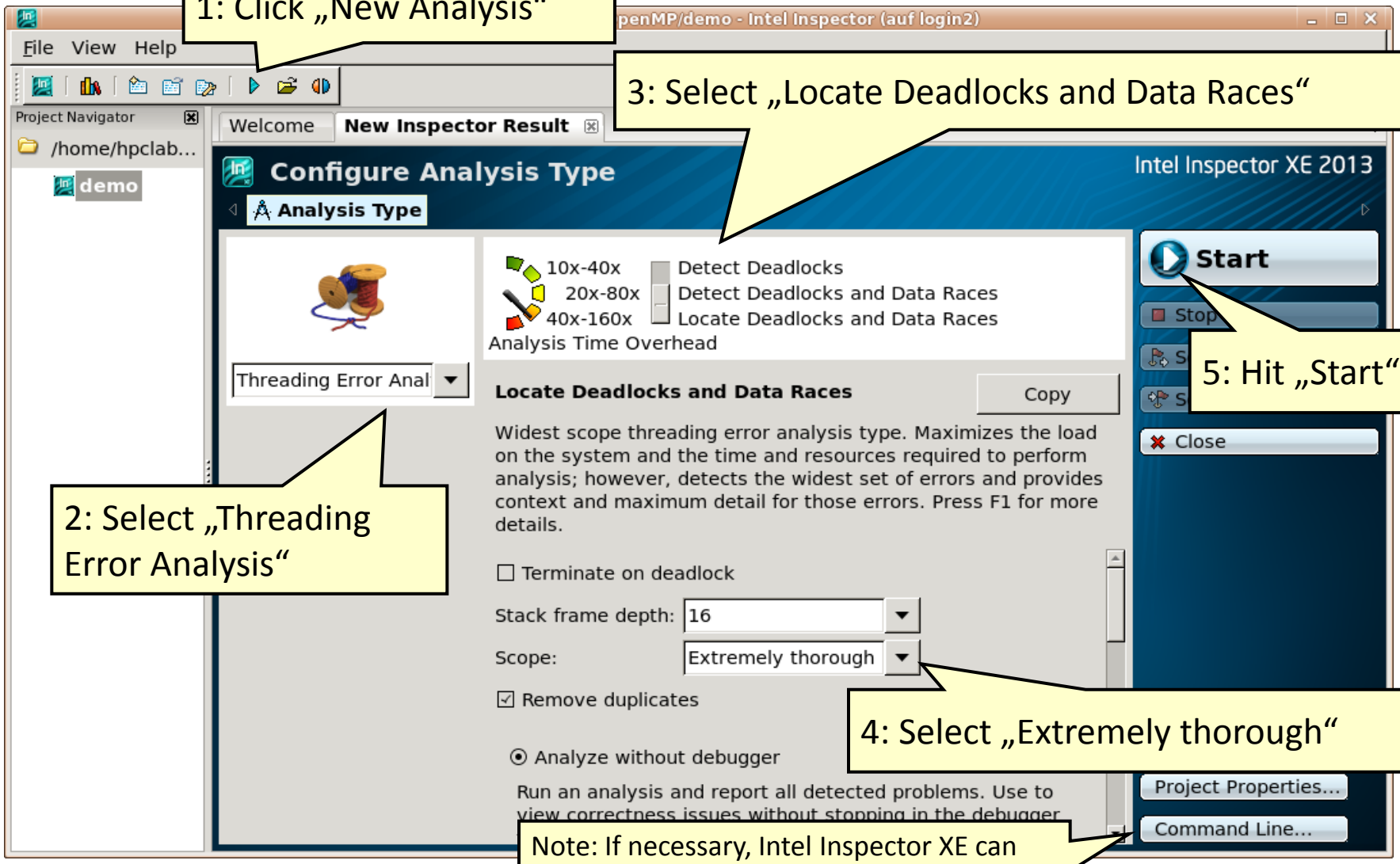
Enter name and location for project directory

Enter path to executable



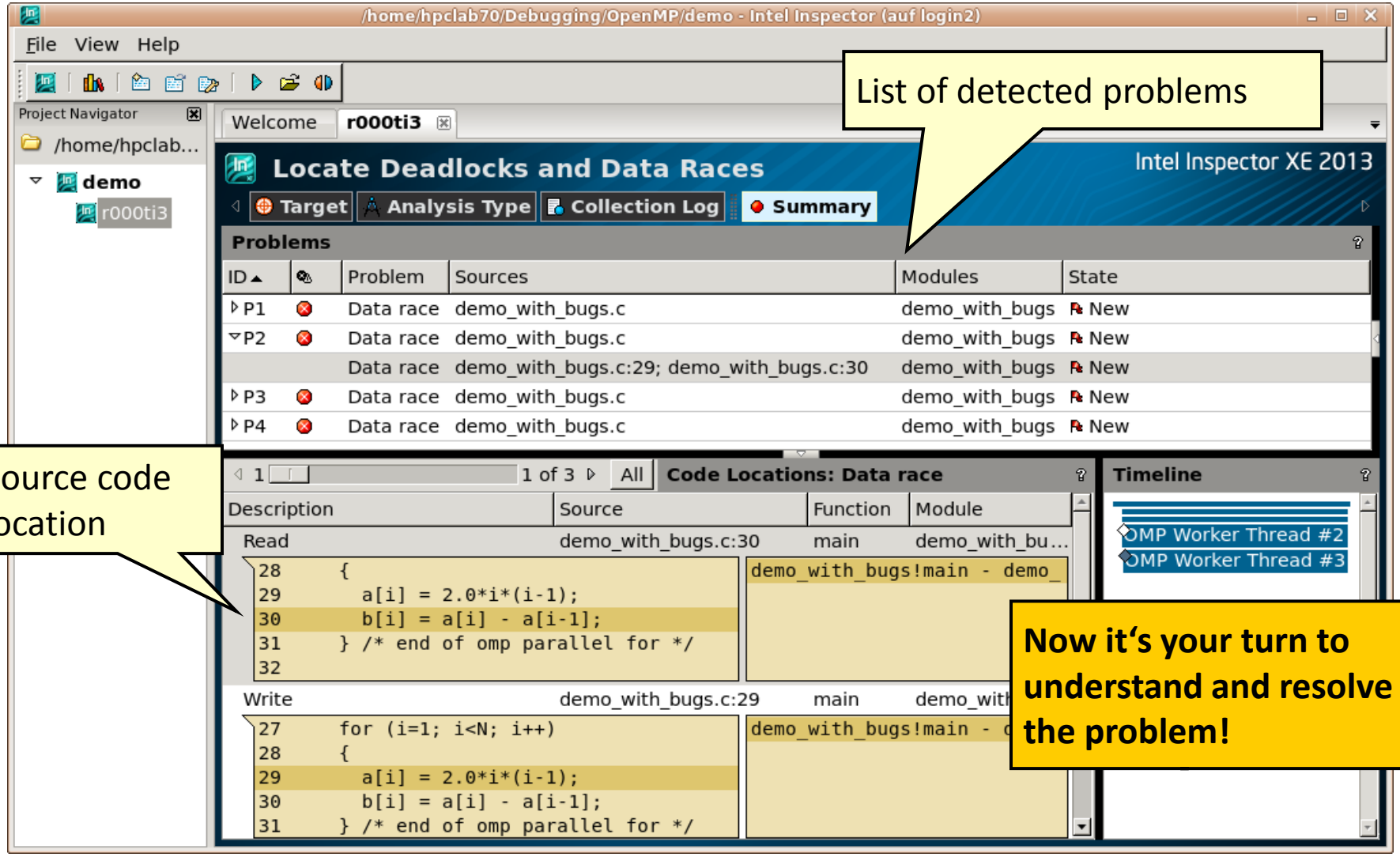
Threaded Applications
Inspector XE – Configure Analysis

Based on slides from Rolf Rabenseifner and Matthias Lieber



Threaded Applications
Inspector XE – Result Summary

Based on slides from Rolf Rabenseifner and Matthias Lieber



Threaded Applications
Archer

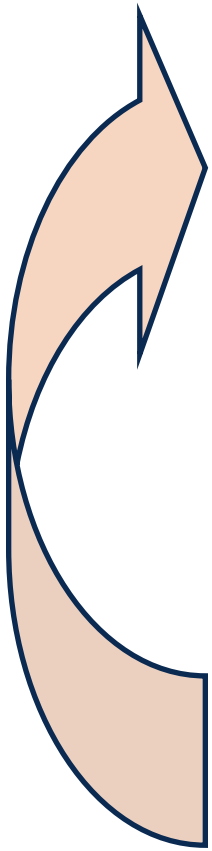
- Error checking tool for
  - Memory errors
  - **Threading errors** (OpenMP, Pthreads)

- Based on LLVM/Clang and ThreadSanitizer (runtime check)
- Available for Linux, Windows and Mac
- Supports C, C++
- More info: https://github.com/PRUNER/archer

Threaded Applications
Archer – Limitations

- Static Analysis
  - Only for OpenMP programs
  - Exclude race free regions and sequential code from runtime analysis to reduce overhead
- Runtime check
  - Error detection only in software branches that are executed
- Low runtime overhead
  - Roughly 2x - 20x
  - Detect races in large OpenMP applications
  - No false positives
- Compiler instrumentation
  - Slower compilation process (apply different passes on the source code to identify race free regions of code, instruments only the rest), faster and more precise at runtime

Threaded Applications
Archer – Usage



- Compile the program with the **–g** compiler flag
  - **`clang-archer myprog.c –o myprog`**
- Run the program under control of Archer Runtime
  - **`export OMP_NUM_THREADS=...`**
    `./myprog`
  - Detects problems only in software branches that are executed
- Understand and correct the threading errors detected
- Edit the source code
- Repeat until no errors reported

Threaded Applications
Archer – Result Summary


```
1  #include <stdio.h>
2
3  int main(int argc, char **argv) {
4      int a = 0;
5      #pragma omp parallel
6      {
7          if (a < 100) {
8              #pragma omp critical
9              a++;
11         }
12     }
13 }
```

WARNING: ThreadSanitizer: data race

Read of size 4 at 0x7fffffffdcdc by thread T2:

#0 .omp_outlined. race.c:7 (race+0x0000004a6dce)

#1 __kmp_invoke_microtask <null> (libomp_tsan.so)

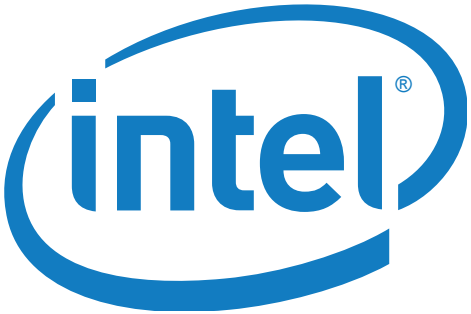
Previous write of size 4 at 0x7fffffffdcdc by main thread:

#0 .omp_outlined. race.c:9 (race+0x0000004a6e2c)

#1 __kmp_invoke_microtask <null> (libomp_tsan.so)

Threaded Applications (Focus: OpenMP)
How to choose the data race detector tool?

- **Intel Inspector XE**
  - **No source code available (only binary)**
  - **Limited memory**
  - **Fortran code**
  - **Intel-specific (SSA-AVX)**
  - **Graphic User-Interface**



- **Archer**
  - **Source code available**
  - **Limited runtime**
  - **No false alarm**
  - **Portability (Intel and PPC)**
  - **Target specific regions of code**

Threaded Applications
Data-race detector limitations

Based on slides from Rolf Rabenseifner and Matthias Lieber

- *"Do not guarantee software tools will detect or report every memory and threading error in an application.*
  - *Not all logic errors are detectable.*
  - *Heuristics used to eliminate false positives may hide real issues.*
  - *Highly correlated events are grouped into a single problem"* (Intel Inspector XE 2013 Release Notes)
- Example of non-detected conceptual error:

```
sum=0;
#pragma omp parallel for private(tmp)
for (i=0; i<100; i++)
{
#pragma omp critical
    tmp = sum;
    tmp = tmp + 1;
#pragma omp critical
    sum   = tmp;
}
printf("sum = %d\n",sum);
```

Non-determinism

```
% icc pseudo-race.c
% ./a.out
sum = 100
% icc -openmp pseudo-race.c
% ./a.out
sum = 60
% ./a.out
sum = 36
% ...
```

Conceptual error not detected

Threaded Applications Summary

- Deadlocks:
  - Avoid locks when possible
  - Prefer critical/master/…
- Races:
  - Often hard to detect, often “sometimes”
  - (Fortran) consider: default(private)
- Use tools to detect defects as early as possible:
  - During development + unit testing
  - Example tools Intel Inspector and Archer

Hands-On II

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], and Joachim Protze[4]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Booth #2143

Booth #1508

Booth #3624

Booth #1030

Organization

- Introduction and Tools
- Hands-On I

Coffee Break

- The Tao of Debugging
- Systematic Debugging

Lunch Break

- Defects in Practice
- MPI Applications
- Threaded Applications

Coffee Break

- **Hands-On II**
- Beyond MPI and Threads
- Discussion and Wrap-Up

Hands-On II Overview

Use the opportunity to try some more debugging and to explore the tools:

a) Debug heat conduction example with multiple issues
   - $HOME/hands-on-2/{c|f90}/
     heat{C/F}-MPI-multi.{c|F90}

   Apply: DDT, MUST, ISP, STAT
   (Challenging; Extensive slide set for step-by-step solution in the slides) OR

b) Learn more about your favorite tool:
   - $HOME/{must|isp|ddt|inspxe|stat|archer}-extra
   - Numbered by suggest order OR

c) If you have a small (open source) code, put it on your virtual machine and try the tools

Hands-On II
(Revisit) Heat Conduction – Build & Run

- Login to your virtual machine, switch directory

```
% cd $HOME/hands-on-2/{c|f90}
```

- Build and run as follows:

```
% mpicc -g heat{C|F}-MPI-multi.{c|F90} –o heat.exe
% mpirun -np 4 heat.exe
```

Hands-On II
Heat Conduction – Initial Output


```
$ mpirun -np 4 heat.exe
intial grid:
================================================================================
 .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 …
 .  1.2 1.7 1.9 2.0 1.9 1.7 1.2  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 .  .  0.9 1.3 1.4 1.3 0.9  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 …

grid after 16 iterations:
================================================================================
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  .
 …
 0.0 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.0  .  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  .
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  .  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  .

= Energy Conservation Check =
 initial Energy:    113.484
   final Energy:    108.893
     Difference:     -4.590
```

Hands-On II
Heat Conduction – Initial Output (2)

- It runs!
- But: It loses energy during the time steps
  - This should not happen with cyclic boundaries
- And as we introduced, there might be further “hidden” errors
- So lets apply our correctness and debugging tools …

Hands-On II
Heat Conduction – The Strategy

- Approach:
  - Let MUST have a look first to identify MPI related issues
  - Let ISP have a look afterwards to investigate non-determinism issues (C only)
  - Use DDT to investigate details around any issues that MUST and ISP find
  - If the application still fails try to isolate any remaining defects with the help of DDT

Hands-On II
Heat Conduction – The Strategy

- Approach:
  - **Let MUST have a look first to identify MPI related issues**
  - Let ISP have a look afterwards to investigate non-determinism issues (C only)
  - Use DDT to investigate details around any issues that MUST and ISP find
  - If the application still fails try to isolate any remaining defects with the help of DDT

Hands-On II
Heat Conduction – Step 1: MUST – Run

- Build:

```
% mpicc -g heatC-MPI-multi.c
```
C

```
% mpif90 -g heatF-MPI-multi.F90
```
Fortran

- Run:

```
% mustrun -np 4 a.out
```

- Investigate:

```
% firefox MUST_Output.html
```

Hands-On Part 2 – Heat Conduction

Step 1: MUST – Analyze Error

First Error: Overlap of communication buffers

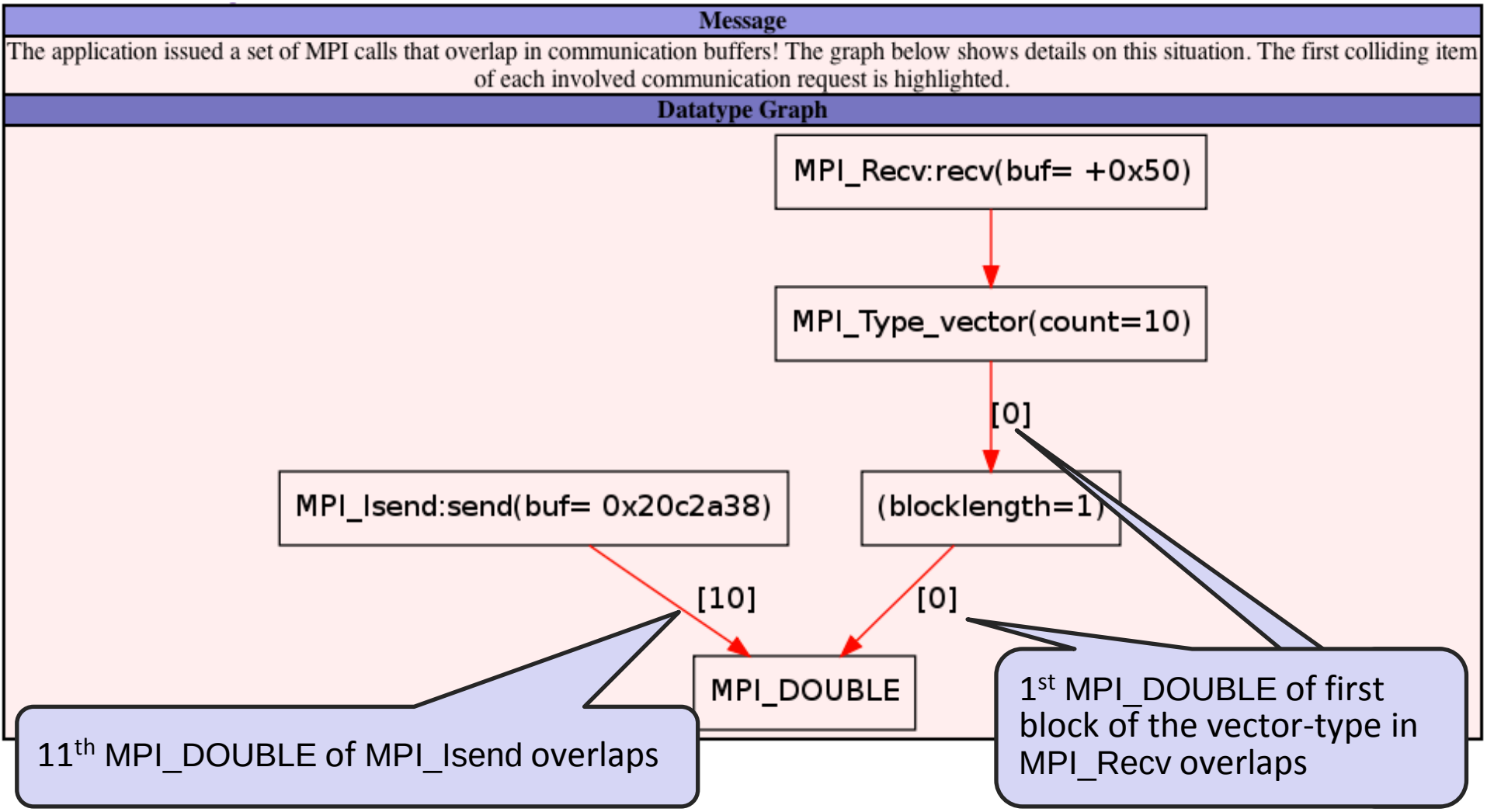
MPI_Isend in line 267

Rank(s)	Type	Message	From	References
1	Error	The memory regions to be transfered by this receive operation overlap with regions spanned by a pending non-blocking operation! (Information on the request associated with the other communication: Request activated at reference 1) (Information on the datatype associated with the other communication: MPI_DOUBLE) The other communication overlaps with this communication at position:[10](MPI_DOUBLE) (Information on the datatype associated with this communication: Datatype created at reference 2 is for C, commited at reference 3, based on the following type(s): { MPI_DOUBLE}) This communication overlaps with the other communication at position:(vector)[0][0](MPI_DOUBLE) A graphical representation of this situation is available in a detailed overlap view (MUST_Output-files/MUST_Overlap_1_0.html).	Representative location: **MPI_Recv** (3rd occurrence) called from: #0 heatBoundary@heatC-MPI-multi.c:284 #1 main@heatC-MPI-multi.c:472	References of a represe[illegible] process: reference 1 rank 1: **MPI_Isend** (1st occurrence) called from: #0 heatBoundary@heatC-MPI-multi.c:267 #1 main@heatC-MPI-multi.c:472 reference 2 rank 1: **MPI_Type_vector** (1st occurrence) called from: #0 heatMPISetup@heatC-MPI-multi.c:353 #1 main@heatC-MPI-multi.c:455 reference 3 rank 1: **MPI_Type_commit** (1st occurrence) called from: #0 heatMPISetup@heatC-MPI-multi.c:359 #1 main@heatC-MPI-multi.c:455

MPI_Recv in line 284

Click on link to visualize!

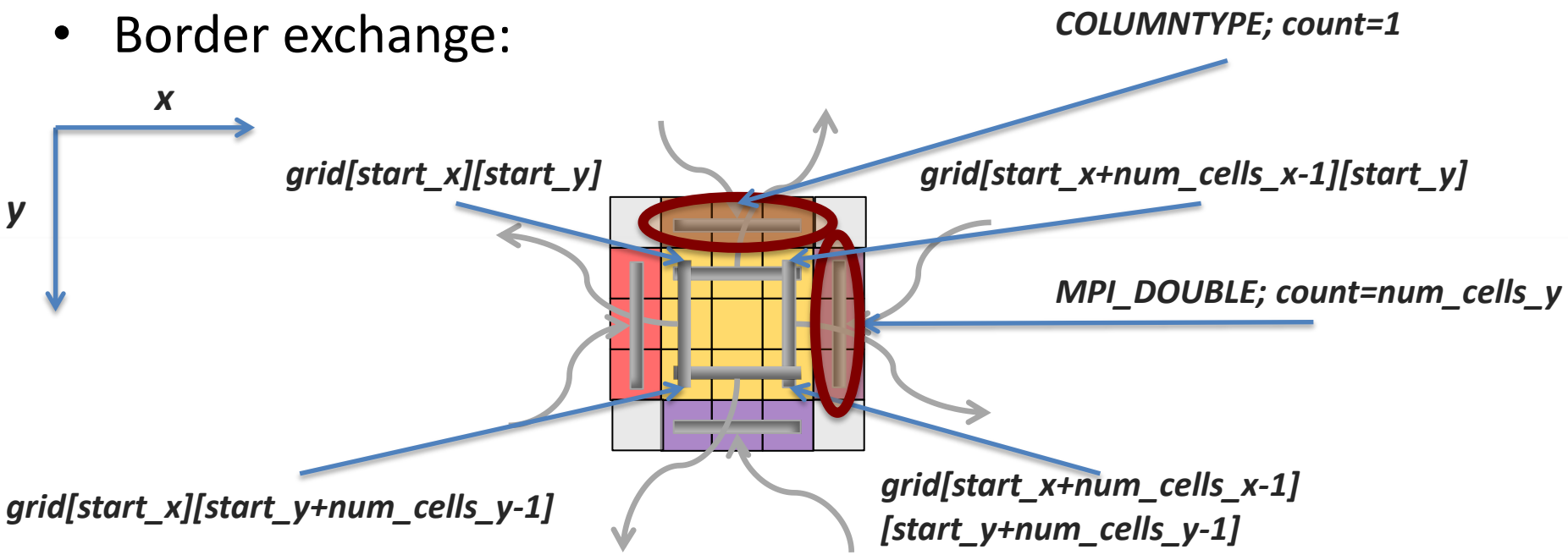
Hands-On Part 2 – Heat Conduction
Step 1: MUST – Analyze Error



Hands-On II
Heat Conduction – Step 1: MUST – Border Exchange

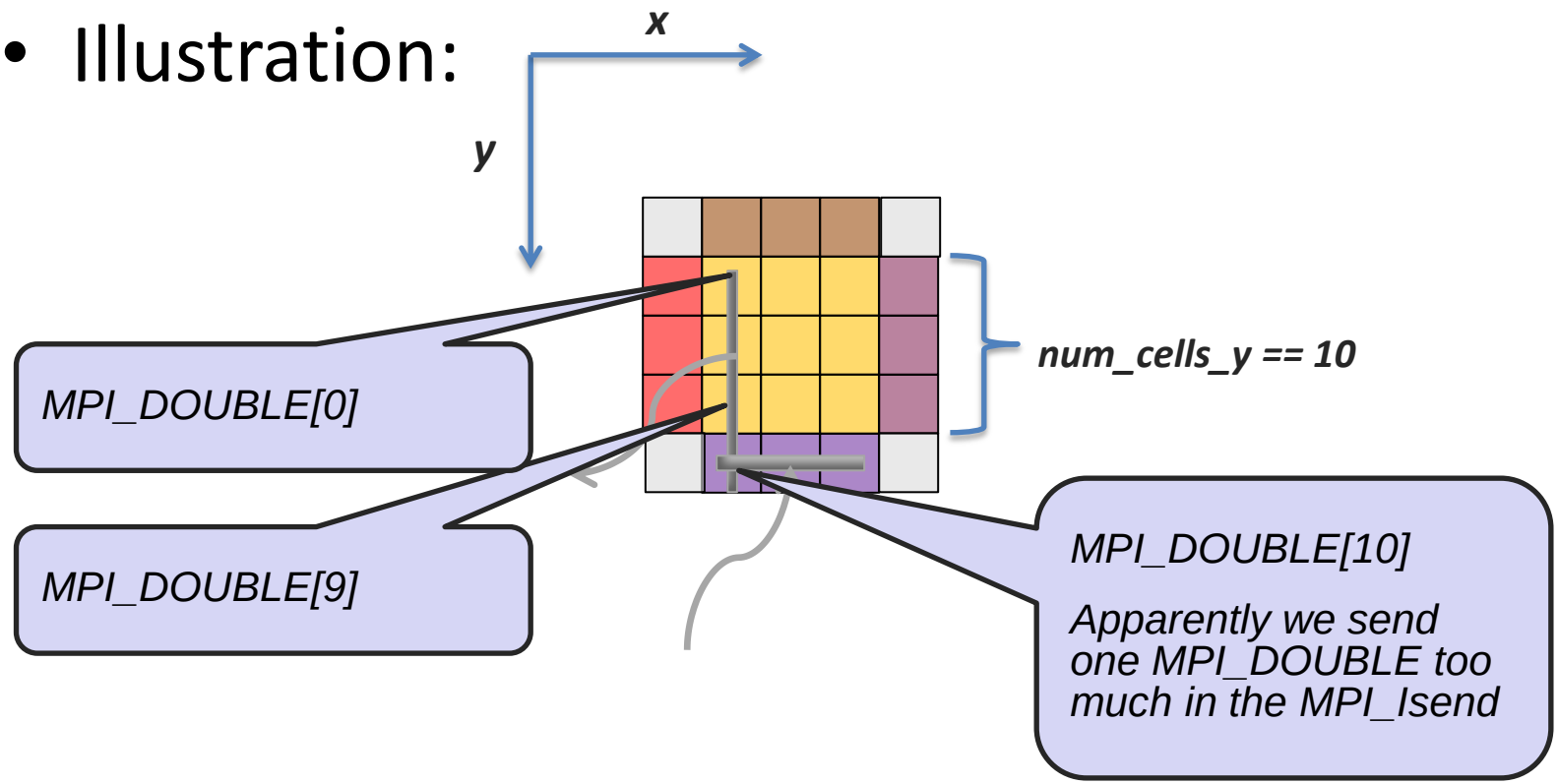
- Each task calculates on a part of the grid:
  - Grid: grid->theta[x][y] ; x and y are coordinates
  - Task calculates on:
    - X: mympi->start_x to mympi->start_x+mymmpi->num_cells_x -1
    - Y: mympi->start_y to mympi->start_y+mymmpi->num_cells_y -1
  - Ghost cells around that
- Border exchange:

Source code uses a Fortran layout, i.e., for C a column is a row and vice versa



Hands-On II
Heat Conduction – MUST – Analyze Error 1 (2)

- Overlap of MPI_Isend@267with MPI_Recv@284
  - Position of column (MPI_DOUBLE)[10]
  - Position of row (VECTOR)[0][0](MPI_DOUBLE)
- Illustration:

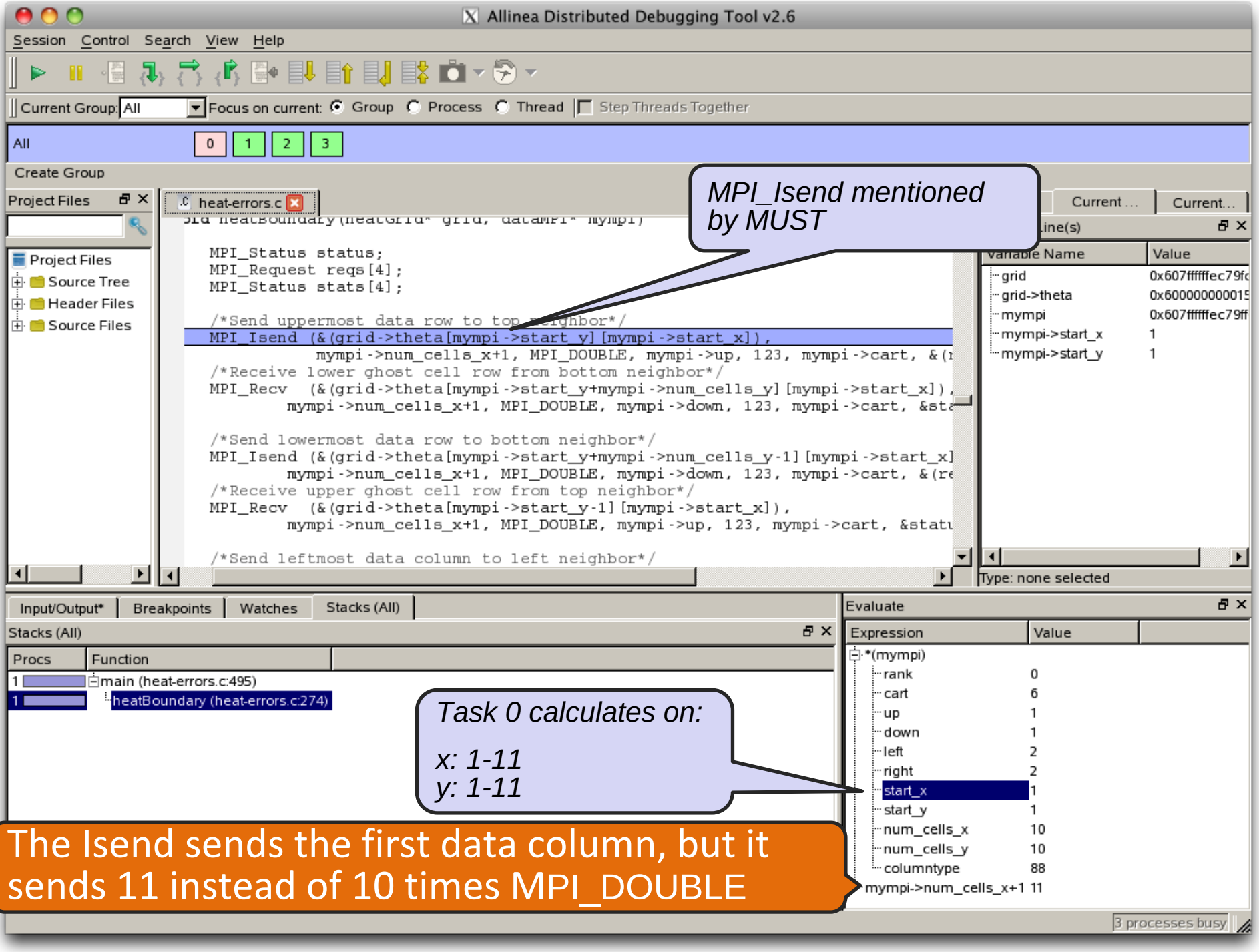


Hands-On II
Heat Conduction – The Strategy

allinea DDT

MUST

- Approach:
  - Let MUST have a look first to identify MPI related issues
  - Let ISP have a look afterwards to investigate non-determinism issues
  - **Use DDT to investigate details around any issues that MUST and ISP find**
  - If the application still fails try to isolate any remaining defects with the help of DDT


Allinea Distributed Debugging Tool v2.6
Session Control Search View Help
Current Group: All
Focus on current: Group Process Thread
Step Threads Together
All 0 1 2 3
Create Group
Project Files
heat-errors.c
MPI_Isend mentioned by MUST
MPI_Status status;
MPI_Request reqs[4];
MPI_Status stats[4];
/*Send uppermost data row to top neighbor*/
MPI_Isend (&(grid->theta[mympi->start_y][mympi->start_x]),
mympi->num_cells_x+1, MPI_DOUBLE, mympi->up, 123, mympi->cart, &(r
/*Receive lower ghost cell row from bottom neighbor*/
MPI_Recv (&(grid->theta[mympi->start_y+mympi->num_cells_y][mympi->start_x]),
mympi->num_cells_x+1, MPI_DOUBLE, mympi->down, 123, mympi->cart, &sta
/*Send lowermost data row to bottom neighbor*/
MPI_Isend (&(grid->theta[mympi->start_y+mympi->num_cells_y-1][mympi->start_x]
mympi->num_cells_x+1, MPI_DOUBLE, mympi->down, 123, mympi->cart, &(re
/*Receive upper ghost cell row from top neighbor*/
MPI_Recv (&(grid->theta[mympi->start_y-1][mympi->start_x]),
mympi->num_cells_x+1, MPI_DOUBLE, mympi->up, 123, mympi->cart, &statu
/*Send leftmost data column to left neighbor*/
Variable Name
Value
grid
grid->theta
mympi
mympi->start_x
mympi->start_y
Type: none selected
Input/Output*
Breakpoints
Watches
Stacks (All)
Procs
Function
main (heat-errors.c:495)
heatBoundary (heat-errors.c:274)
Task 0 calculates on:
x: 1-11
y: 1-11
Evaluate
Expression
Value
*(mympi)
rank 0
cart 6
up 1
down 1
left 2
right 2
start_x 1
start_y 1
num_cells_x 10
num_cells_y 10
columntype 88
mympi->num_cells_x+1 11
3 processes busy
The Isend sends the first data column, but it sends 11 instead of 10 times MPI_DOUBLE

Hands-On II
Heat Conduction – Correct Issue 1

MUST allinea DDT

```
 /*Send left column to left neighbor*/
266: MPI_Isend (&(grid->theta[mympi->start_x][mympi->start_y]),
        mympi->num_cells_y+1, MPI_DOUBLE, mympi->left, 123, mympi->cart, &(requests[0]));
/*Receive Right border column from right neighbor*/
MPI_Recv  (&(grid->theta[mympi->start_x+mympi->num_cells_x][mympi->start_y]),
        mympi->num_cells_y+1, MPI_DOUBLE, mympi->right, 123, mympi->cart, &status);
```

Modify to

```
 /*Send left column to left neighbor*/
266: MPI_Isend (&(grid->theta[mympi->start_x][mympi->start_y]),
        mympi->num_cells_y, MPI_DOUBLE, mympi->left, 123, mympi->cart, &(requests[0]));
/*Receive Right border column from right neighbor*/
MPI_Recv  (&(grid->theta[mympi->start_x+mympi->num_cells_x][mympi->start_y]),
        mympi->num_cells_y, MPI_DOUBLE, mympi->right, 123, mympi->cart, &status);
```

- Repeat for second column exchange

Hands-On II
Heat Conduction – Step 1: MUST

- Rebuild & rerun with MUST after the correction

⇒MUST detects no further errors

⇒However, we still lose energy

⇒So further defects are present!

Hands-On II
Heat Conduction – The Strategy

- Approach:
  - Let MUST have a look first to identify MPI related issues
  - **Let ISP have a look afterwards to investigate non-determinism issues (C only)**
  - Use DDT to investigate details around any issues that MUST and ISP find
  - If the application still fails try to isolate any remaining defects with the help of DDT

Hands-On II
Heat Conduction – Step 2: ISP – Maxima of dTheta

- In “heatTimestep“ each task computes its maximum temperature difference „mymax“
- Communication for global maxima:

```
/* Calculate maximum dtheta of all processes and distribute it */
if (mympi->rank != 0) {
   MPI_Send (&mymax, 1, MPI_DOUBLE, 0, 567, mympi->cart);
   MPI_Recv (dthetamax, 1, MPI_DOUBLE, 0, 234, mympi->cart, &status);
} else {
   …
   for (i = 1; i < size; i++) {
      MPI_Recv (&tempmax, 1, MPI_DOUBLE, i, 567, mympi->cart, &status);
         mymax = fmax (mymax, tempmax);
   }
   for (i = 1; i < size; i++) {
      MPI_Send (dthetamax, 1, MPI_DOUBLE, i, 234, mympi->cart);
   }
}
```

Hands-On II
Heat Conduction – Step 2: ISP – Run

- Build with ISP compiler wrapper:

```
% ispcc -g heatC-MPI-multi.c
```

C

Fortran support currently not available for ISP

- Run with ISP:

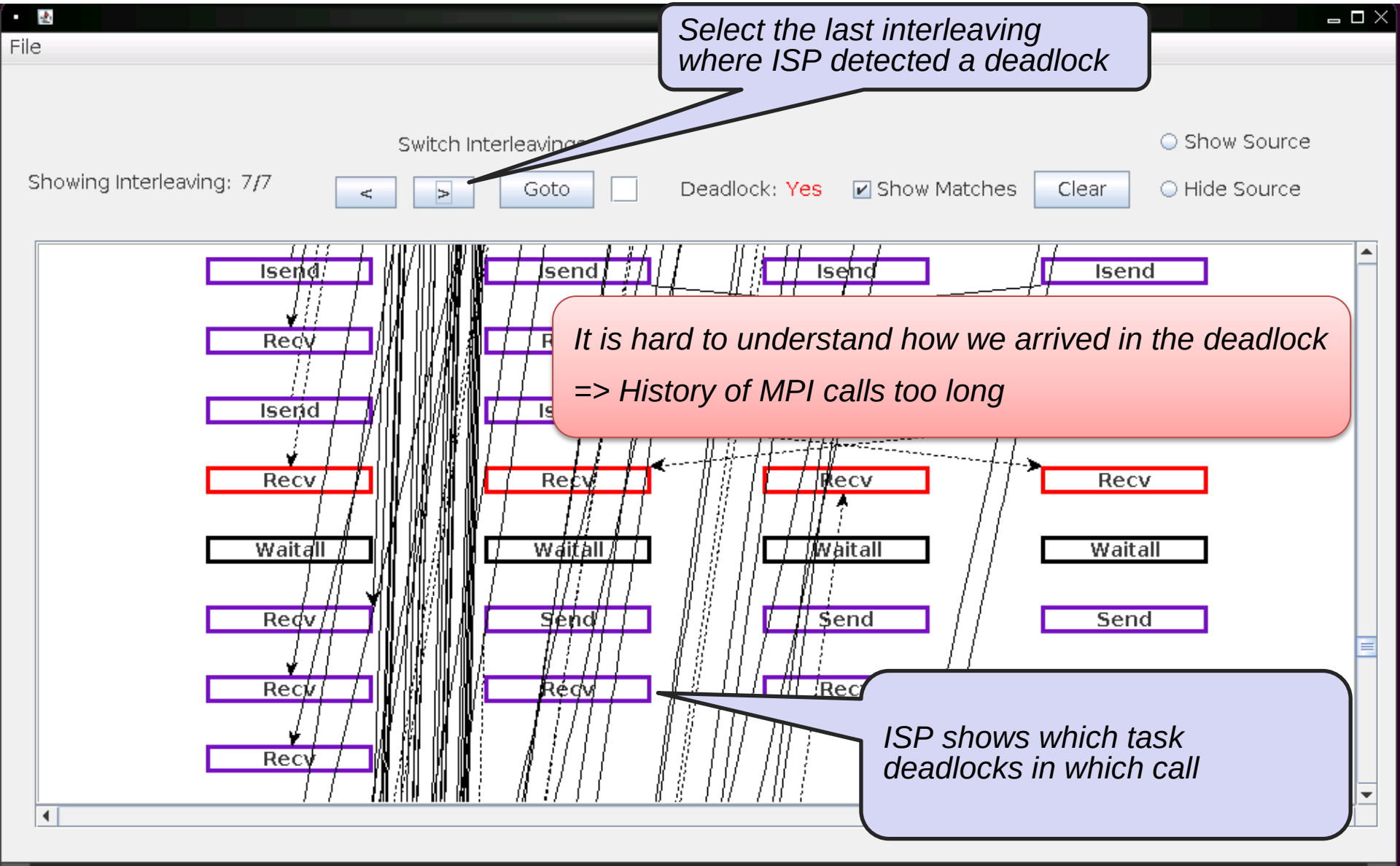
"-z T" => Stop analysis at first deadlock

```
% isp -n 4 –z T –l isp.log a.out
```

- Investigate with ISP UI:

```
% ispUI isp.log
```

Hands-On II
Heat Conduction – Step 2: ISP – Analyze



Hands-On II
Heat Conduction – Step 2: ISP – Analyze (2)

- If the history is too complex, make it simpler!
- We can reduce the number of iterations to derive a simpler output

```
int main (int argc, char** argv)
{
   heatGrid mygrid;
   double dt, dthetamax, energyInitial, energyFinal;
441:   int step, nsteps=20;
   ...
}
```

Just use a single iteration (nsteps=1)

Hands-On II
Heat Conduction – Step 2: ISP – Run number two

- Build with ISP compiler wrapper:

```
% ispcc -g heatC-MPI-multi.c
```

C

Fortran support currently not available for ISP

- Run with ISP:

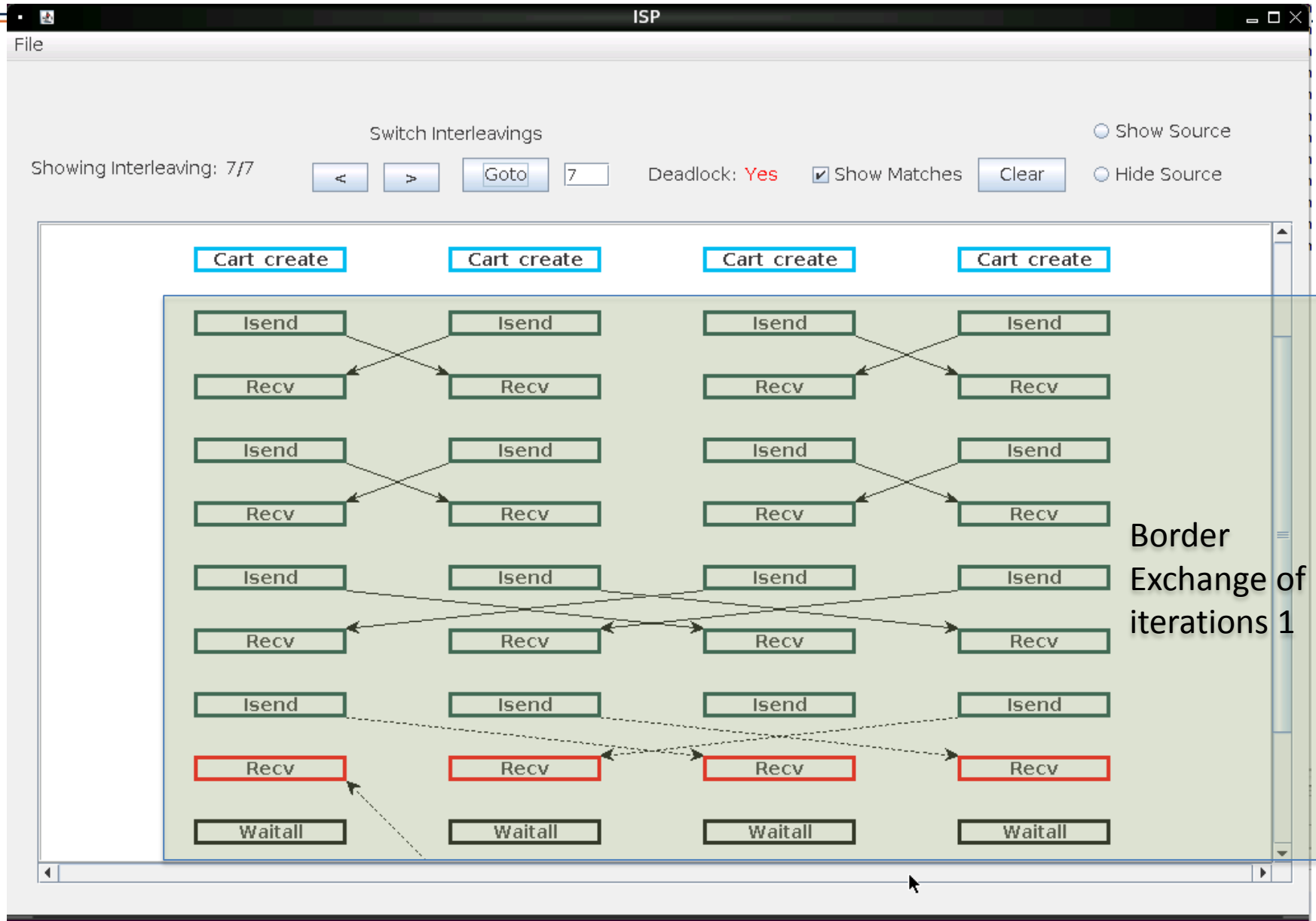
"-z T" => Stop analysis at first deadlock

```
% isp -n 4 –z T –l isp.log a.out
```

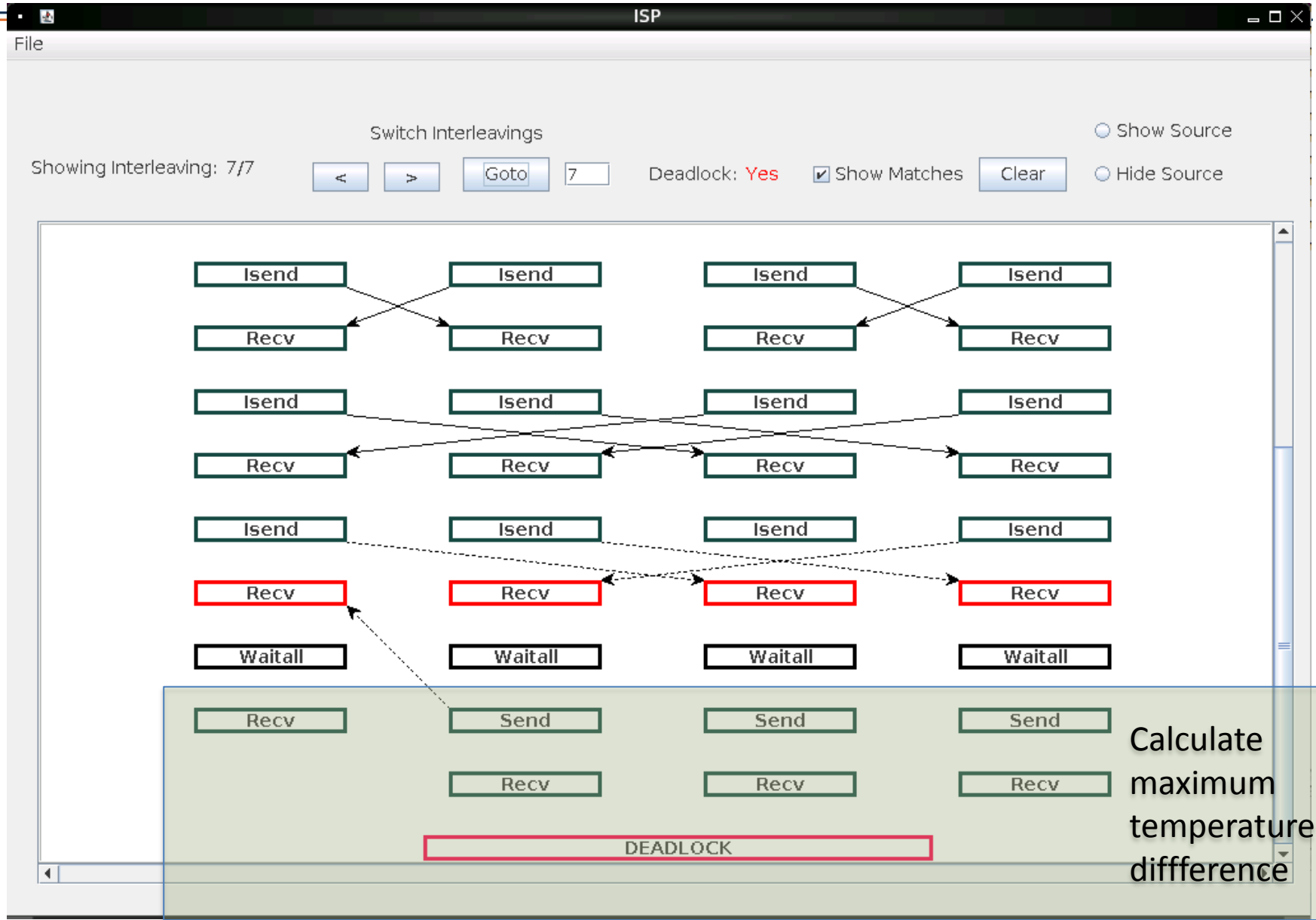
- Investigate with ISP UI:

```
% ispUI isp.log
```

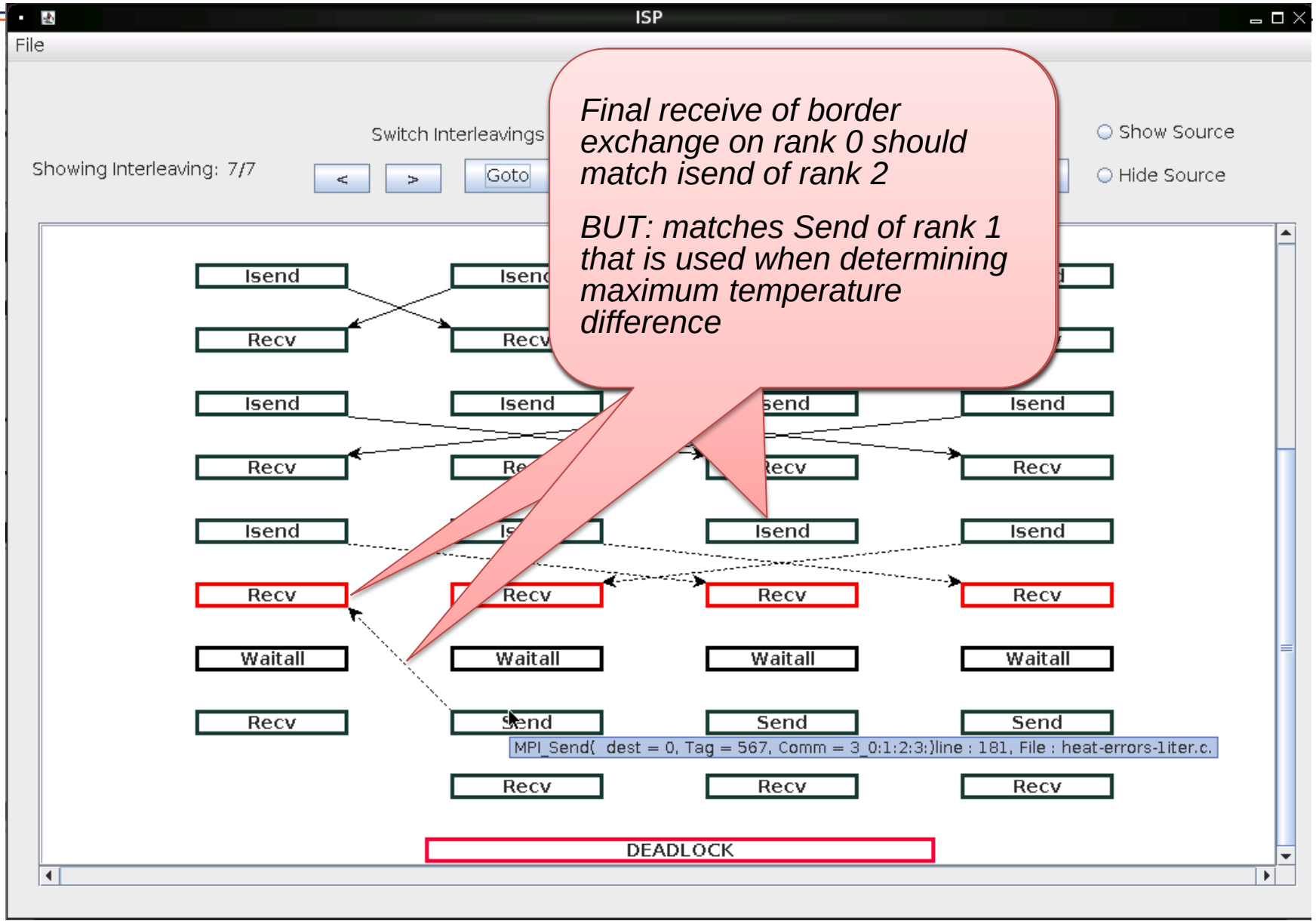
Hands-On II
Heat Conduction – Step 2: ISP – Analyze (4)



Hands-On II
Heat Conduction – Step 2: ISP – Analyze (5)

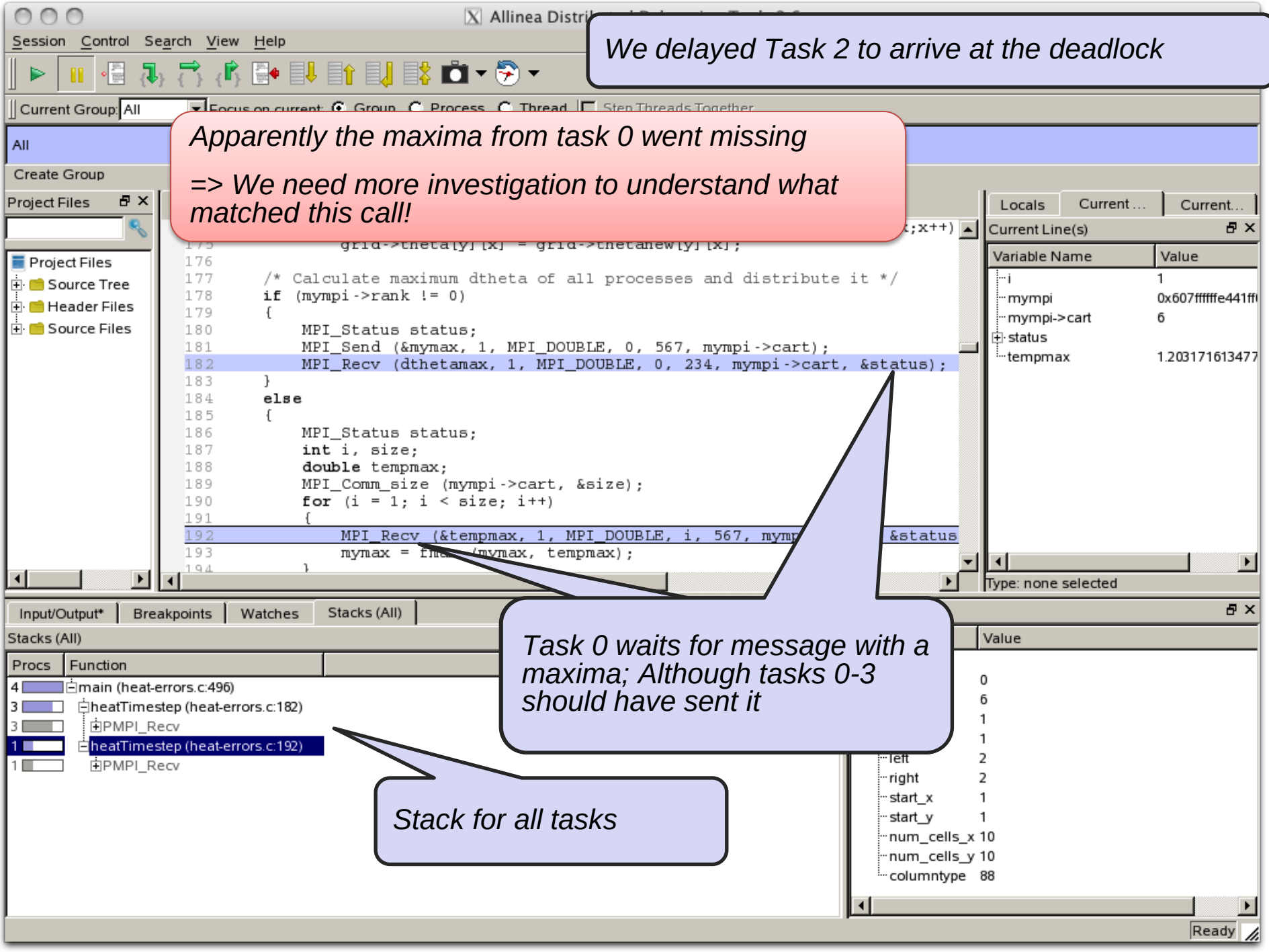


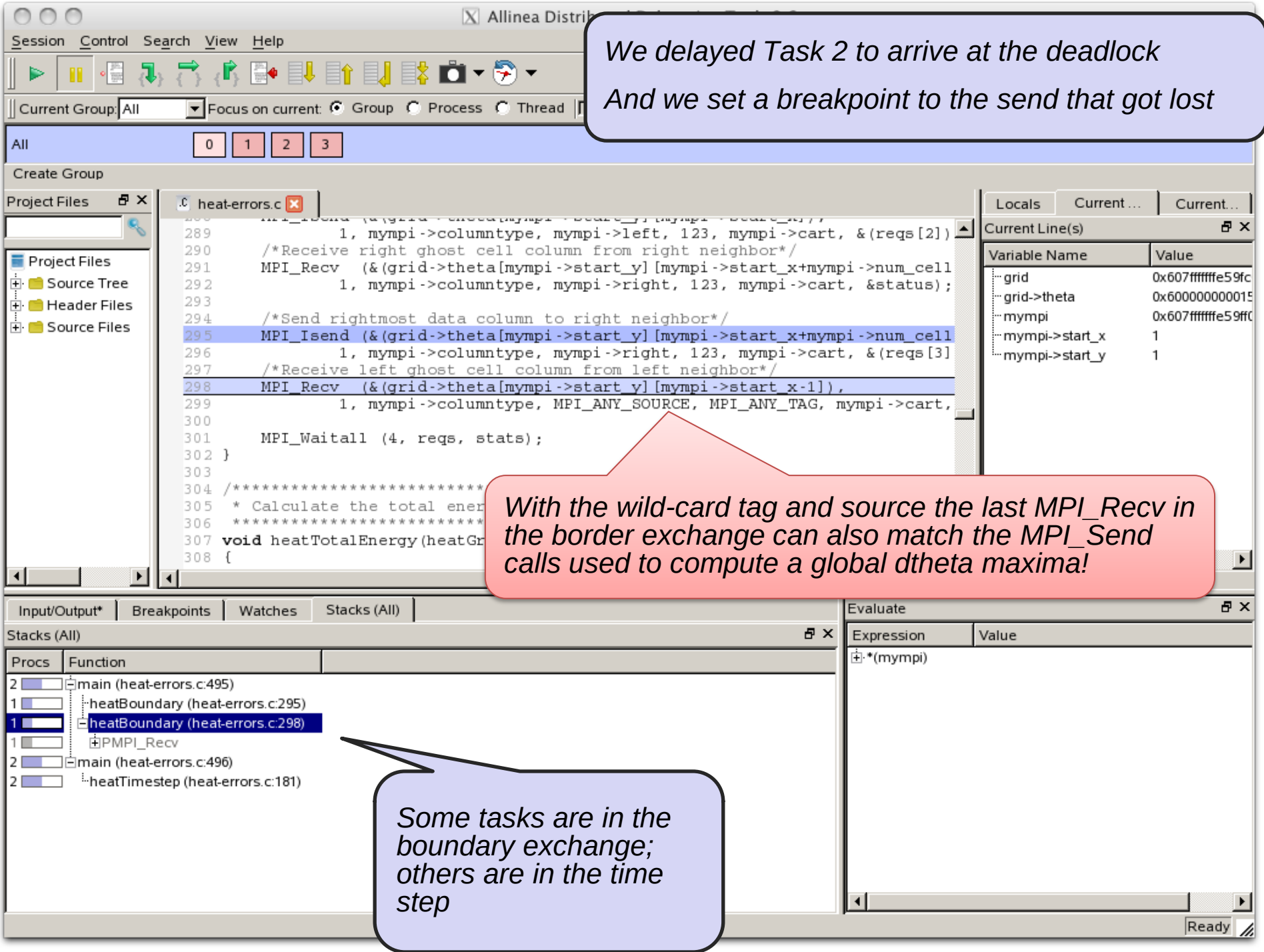
Hands-On II
Heat Conduction – Step 2: ISP – Analyze (6)



Hands-On II
Heat Conduction – The Strategy

- Approach:
  - Let MUST have a look first to identify MPI related issues
  - Let ISP have a look afterwards to investigate non-determinism issues (C only)
  - **Use DDT to investigate details around any issues that MUST and ISP find**
  - If the application still fails try to isolate any remaining defects with the help of DDT


We delayed Task 2 to arrive at the deadlock
Apparently the maxima from task 0 went missing
=> We need more investigation to understand what matched this call!
Task 0 waits for message with a maxima; Although tasks 0-3 should have sent it
Stack for all tasks
Session Control Search View Help
Current Group: All
Create Group
Project Files
Source Tree
Header Files
Source Files
Locals
Current Line(s)
Variable Name
Value
i 1
mympi 0x607fffffe441ff
mympi->cart 6
status
tempmax 1.203171613477
Type: none selected
Input/Output*
Breakpoints
Watches
Stacks (All)
Procs
Function
4 main (heat-errors.c:496)
3 heatTimestep (heat-errors.c:182)
3 PMPI_Recv
1 heatTimestep (heat-errors.c:192)
1 PMPI_Recv
left 2
right 2
start_x 1
start_y 1
num_cells_x 10
num_cells_y 10
columntype 88
Ready


We delayed Task 2 to arrive at the deadlock
And we set a breakpoint to the send that got lost
With the wild-card tag and source the last MPI_Recv in the border exchange can also match the MPI_Send calls used to compute a global dtheta maxima!
Some tasks are in the boundary exchange; others are in the time step
Session Control Search View Help
Current Group: All
Focus on current: Group Process Thread
All 0 1 2 3
Create Group
Project Files
Project Files
Source Tree
Header Files
Source Files
heat-errors.c
MPI_Recv (&(grid->theta[mympi->start_y][mympi->start_x-1]),
1, mympi->columntype, MPI_ANY_SOURCE, MPI_ANY_TAG, mympi->cart,
MPI_Waitall (4, reqs, stats);
void heatTotalEnergy(heatGr
Locals
Current ...
Current...
Current Line(s)
Variable Name
Value
grid
grid->theta
mympi
mympi->start_x
mympi->start_y
Input/Output*
Breakpoints
Watches
Stacks (All)
Stacks (All)
Procs
Function
2 main (heat-errors.c:495)
1 heatBoundary (heat-errors.c:295)
1 heatBoundary (heat-errors.c:298)
1 PMPI_Recv
2 main (heat-errors.c:496)
2 heatTimestep (heat-errors.c:181)
Evaluate
Expression
Value
*(mympi)
Ready

Hands-On II
Heat Conduction – Step 2: ISP – What we learned

- The wild-card in the last MPI_Recv call can consume the MPI_Send calls used to determine the global dtheta maxima
- Either not use the wild-card source, or not use a wild-card tag

Hands-On II
Heat Conduction – Step 2: ISP – Correct the Error

```
...
/*Receive upper border row from top neighbor*/
292: MPI_Recv  (&(grid->theta[mympi->start_x][mympi->start_y-1]),
        1, mympi->rowtype, MPI_ANY_SOURCE, MPI_ANY_TAG, mympi->cart, &status);
...
```

Modify to

```
...
/*Receive upper border row from top neighbor*/
MPI_Recv  (&(grid->theta[mympi->start_x][mympi->start_y-1]),
        1, mympi->rowtype, mympi->up, 123, mympi->cart, &status);
...
```

Hands-On II
Heat Conduction – Step 2: ISP

- Rebuild & rerun with ISP after the correction
- ISP detects no further errors
- However, we still lose energy
- So further defects are present!
- ISP and MUST detected 2 errors, but not the energy leak

Hands-On II
Heat Conduction – The Strategy

- Approach:
  - Let MUST have a look first to identify MPI related issues
  - Let ISP have a look afterwards to investigate non-determinism issues
  - Use DDT to investigate details around any issues that MUST and ISP find
  - **If the application still fails try to isolate any remaining defects with the help of DDT**

Hands-On II
Heat Conduction – Step 3: DDT – Starting Point

grid after 16 iterations:

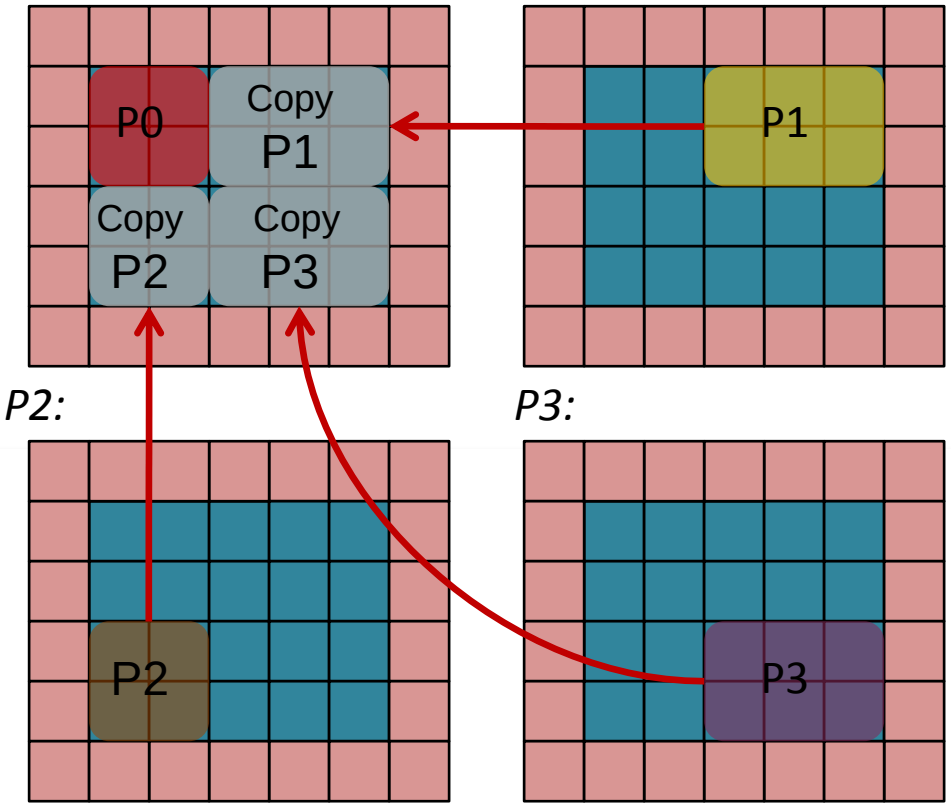
```
==========                                                    ==========
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.2 0.5 0.8 1.0 1.
 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.2 0.5 0.9 1.3 1.5 1.
 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.3 0.8 1.3 1.6 1.8 1.
 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.4 1.0 1.5 1.8 1.9 1.
 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.5 1.0 1.5 1.8 1.9 1.9 1.9      1.0 0.5
 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.4 1.0 1.5 1.8 1.9 1.9 1.9      1.0 0.4 0.
 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.3 0.8 1.3 1.6 1.8 1.8 1.8      3 0.8 0.3 0.1 0
 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.2 0.5 0.9 1.3 1.5 1.5 1.       .9 0.5 0.2 0.1 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.2 0.5 0.8 1.0 1.0 1.       0.5 0.2 0.1 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.2 0.3 .  .  .  .       .  .  .
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.1 0.1 0.1 0.       .1 0.1 0.0 0.0 0.0 0.0
                                     0.0 0.0 0.0 0.0 0.0      .0 0.0 0.0 0.0 0.0 0.0
                                     0.0 0.0 0.0 0.0 0.0      .0 0.0 0.0 0.0 0.0 0.0
                                     0.0 0.0 0.0 0.0 0.0      .0 0.0 0.0 0.0 0.0 0.0
                                     0.0 0.0 0.0 0.0 0.0      .0 0.0 0.0 0.0 0.0 0.0
                                     0.0 0.0 0.0 0.0 0.0      .0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0             .0 0.0 0.0 0.0 0.0 0.    .0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.          0 0.0 0.0 0.0 0.0 0.0    0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.          0 0.0 0.1 0.1 0.1 0.1    0.1 0.1 0.1 0.0 0.0 0.0
  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
= Energy Conservation Check =
... Difference:     -4.590
```

Weird pattern of data points that are exactly 0.0

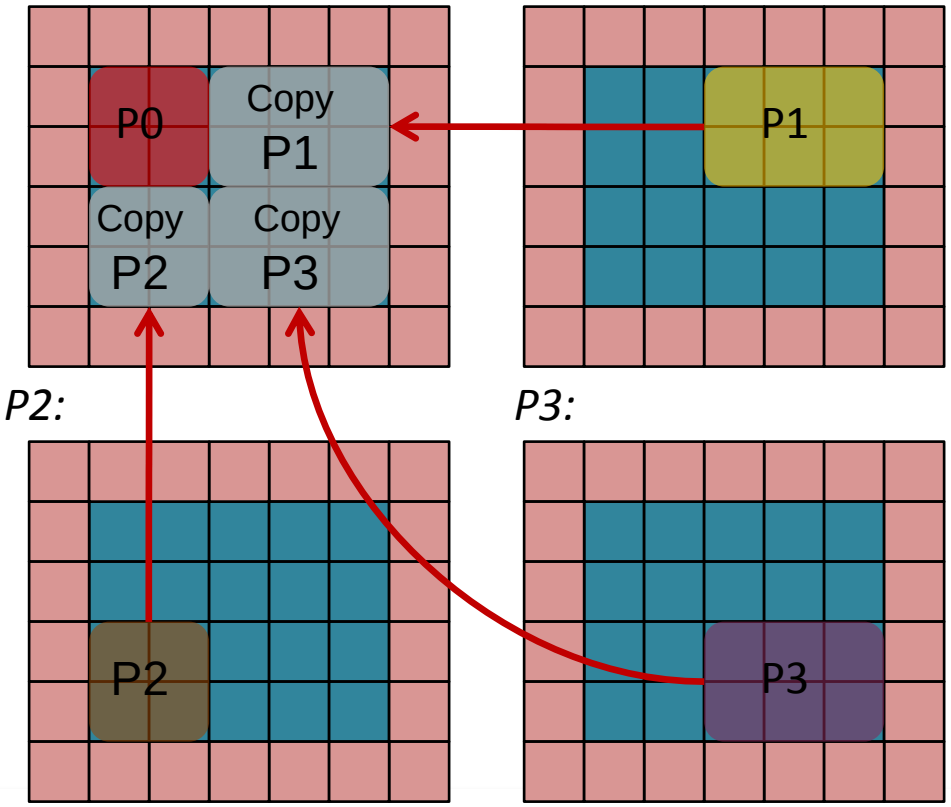
We lose energy during our computation

Hands-On II
Heat Conduction – Step 3: DDT – Data Gathering

- Data distributed between processes
- Printing requires I/O or to gather data on one process
- Implementation gathers data on process 0:



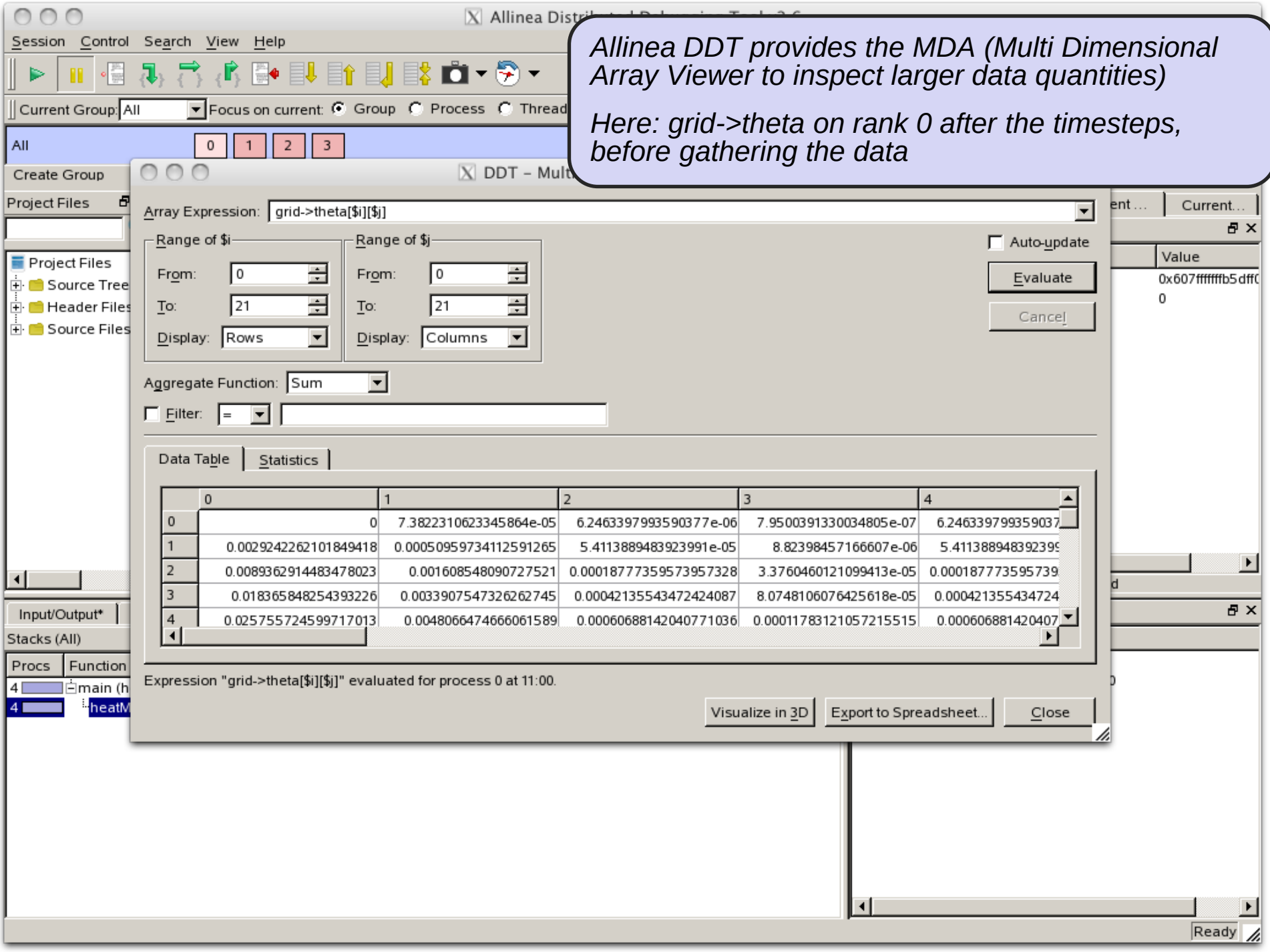
Hands-On II
Heat Conduction – Step 3: DDT – Data Gathering (2)



- Some blocks have a different layout
- A vector datatype is build for each block
- Function: heatMPIGather

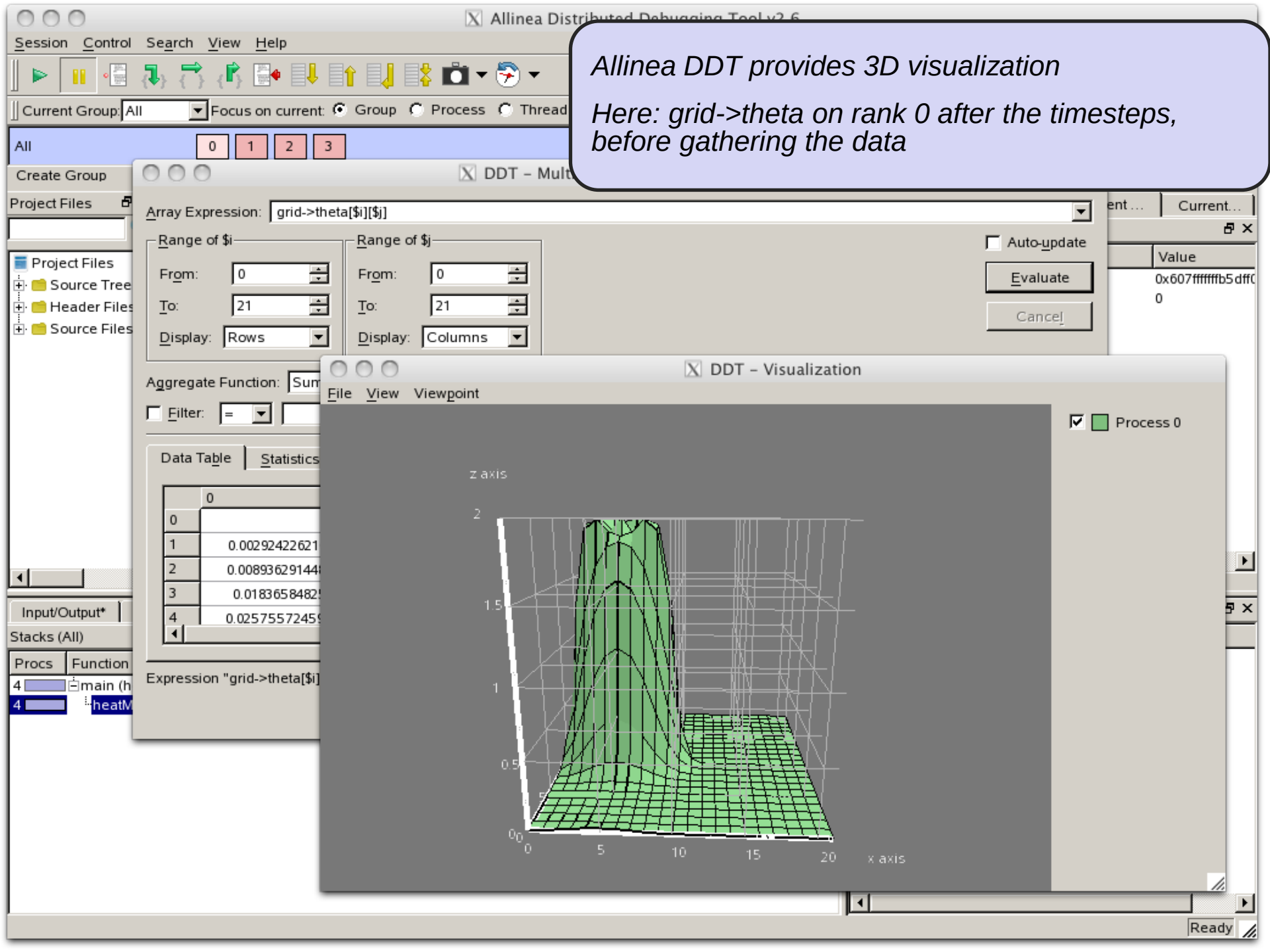
Hands-On II
Heat Conduction – Step 3: DDT – Inspection Point

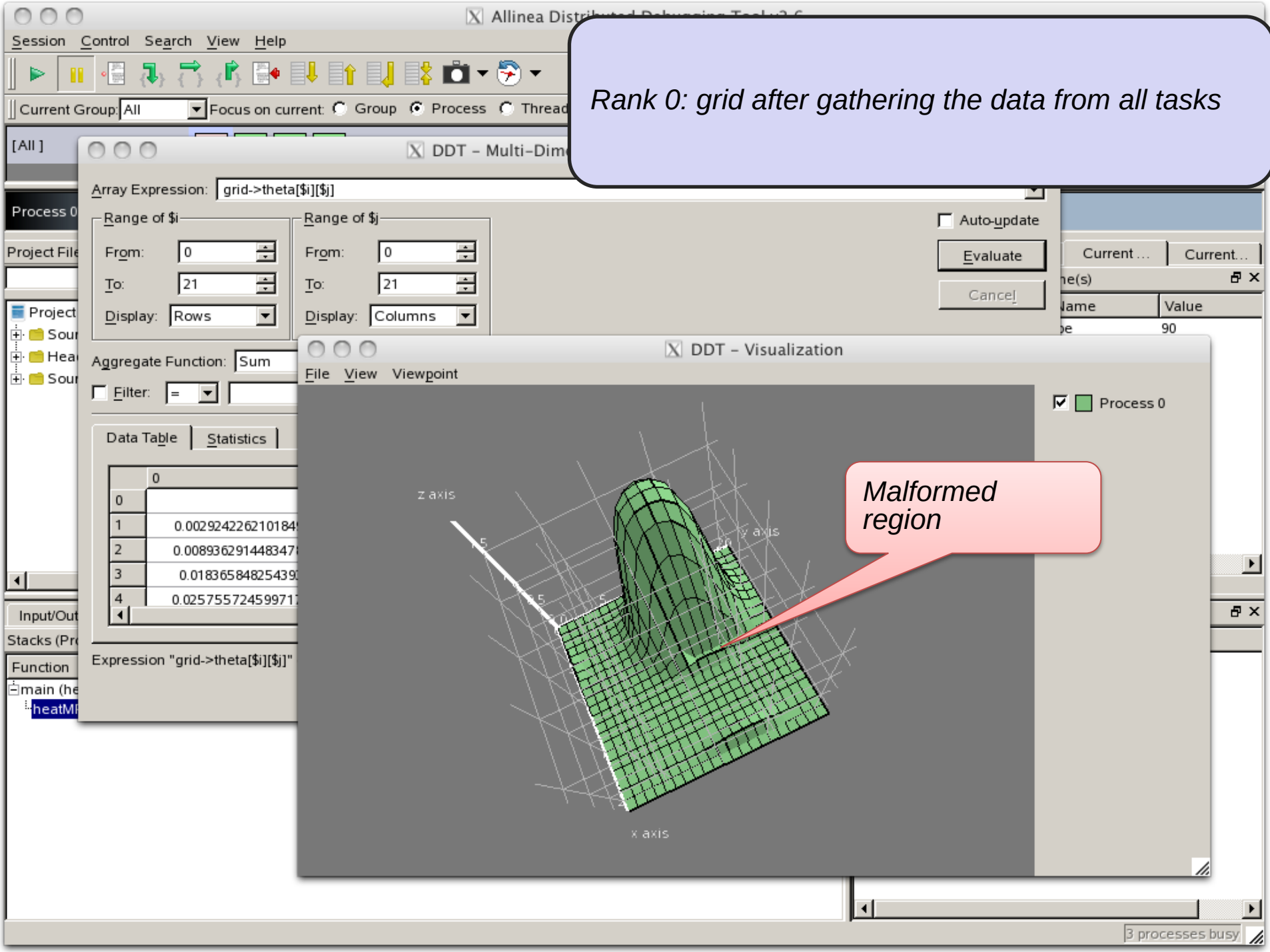
- Communication error likely (weird output pattern)
- Building the vector type for each block might be an error source
- Idea:
  - Investigate data on sender and receiver side
  - Allinea DDT offers an array viewer for such tasks

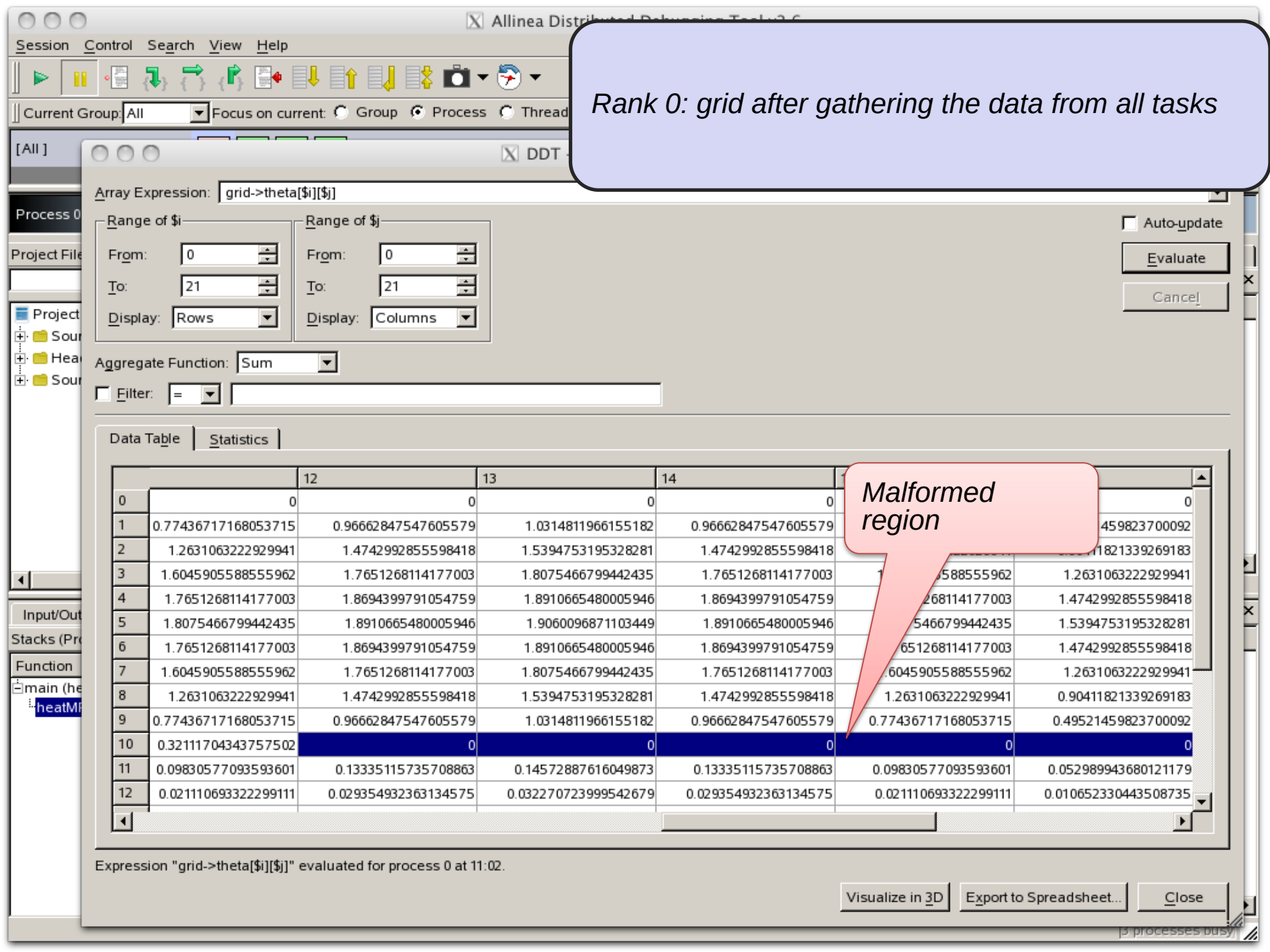

Allinea DDT provides the MDA (Multi Dimensional Array Viewer to inspect larger data quantities)
Here: grid->theta on rank 0 after the timesteps, before gathering the data
Session Control Search View Help
Current Group: All
Focus on current: Group Process Thread
All 0 1 2 3
Create Group
Project Files
Project Files
Source Tree
Header Files
Source Files
DDT – Mult
Array Expression: grid->theta[$i][$j]
Range of $i
From: 0
To: 21
Display: Rows
Range of $j
From: 0
To: 21
Display: Columns
Auto-update
Evaluate
Cancel
Aggregate Function: Sum
Filter: =
Data Table
Statistics
0 1 2 3 4
0 0 7.3822310623345864e-05 6.2463397993590377e-06 7.9500391330034805e-07 6.246339799359037
1 0.0029242262101849418 0.00050959734112591265 5.4113889483923991e-05 8.82398457166607e-06 5.411388948392395
2 0.0089362914483478023 0.001608548090727521 0.00018777359573957328 3.3760460121099413e-05 0.000187773595739
3 0.018365848254393226 0.003390754732626274 5 0.00042135543472424087 8.0748106076425618e-05 0.000421355434724
4 0.025755724599717013 0.0048066474666061589 0.00060688142040771036 0.00011783121057215515 0.000606881420407
Expression "grid->theta[$i][$j]" evaluated for process 0 at 11:00.
Visualize in 3D
Export to Spreadsheet...
Close
Value
0x607fffffffb5dff0
0
Input/Output*
Stacks (All)
Procs
Function
4 main (h
4 heatM
Ready

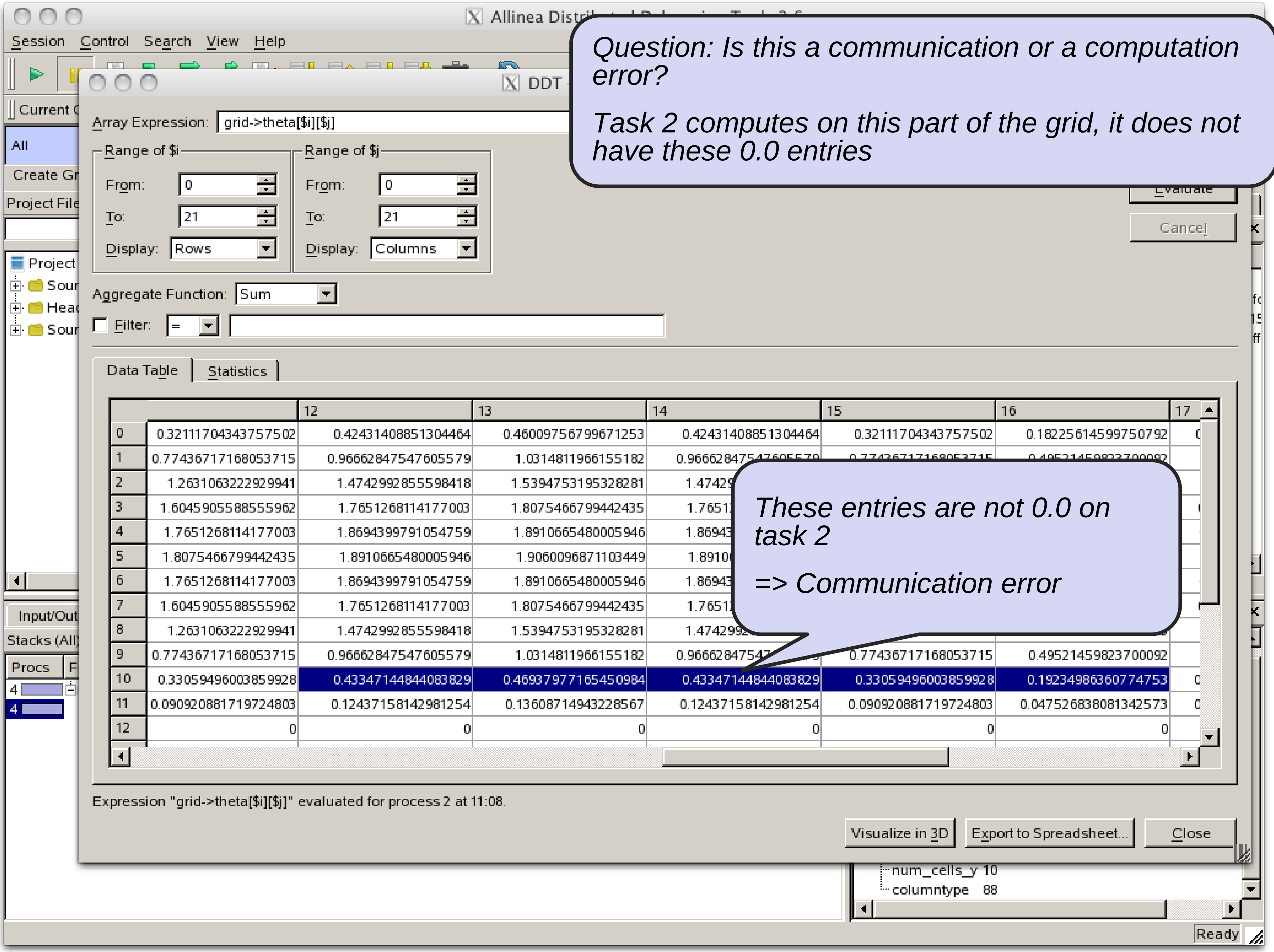
Allinea DDT provides 3D visualization

Here: grid->theta on rank 0 after the timesteps, before gathering the data




Rank 0: grid after gathering the data from all tasks
Session Control Search View Help
Current Group: All
Focus on current: Group Process Thread
DDT – Multi-Dim
Array Expression: grid->theta[$i][$j]
Range of $i
From: 0
To: 21
Display: Rows
Range of $j
From: 0
To: 21
Display: Columns
Auto-update
Evaluate
Cancel
Aggregate Function: Sum
Filter: =
Data Table
Statistics
Expression "grid->theta[$i][$j]"
DDT – Visualization
File View Viewpoint
Process 0
z axis
y axis
x axis
Malformed region
3 processes busy


Rank 0: grid after gathering the data from all tasks
Malformed region
Session Control Search View Help
Current Group: All
Focus on current: Group Process Thread
DDT
Array Expression: grid->theta[$i][$j]
Range of $i
From: 0
To: 21
Display: Rows
Range of $j
From: 0
To: 21
Display: Columns
Auto-update
Evaluate
Cancel
Aggregate Function: Sum
Filter: =
Data Table
Statistics
Expression "grid->theta[$i][$j]" evaluated for process 0 at 11:02.
Visualize in 3D
Export to Spreadsheet...
Close


Question: Is this a communication or a computation error?
Task 2 computes on this part of the grid, it does not have these 0.0 entries
These entries are not 0.0 on task 2
=> Communication error
Array Expression: grid->theta[$i][$j]
Range of $i
Range of $j
Aggregate Function: Sum
Data Table
Statistics
Expression "grid->theta[$i][$j]" evaluated for process 2 at 11:08.
Visualize in 3D
Export to Spreadsheet...
Close

Hands-On II
Heat Conduction – Step 3: DDT – Error Source

- During data gathering the last row of each block is not communicated
- Each task needs to communicate mympi->num_cells_x columns
- Inspect the type constructor on the send & receiver side

Hands-On II
Heat Conduction – Step 3: DDT – Correct the Error

```
/* Create datatype to communicate one block*/
396: MPI_Type_vector (
        mympi->num_cells_x-1, /* #blocks */
        mympi->num_cells_y, /* #elements per block */
        grid->ysize+2, /* #stride */
        MPI_DOUBLE, /* old type */
        &blocktype /* new type */ );
```

Modify to

```
/* Create datatype to communicate one block*/
MPI_Type_vector (
        mympi->num_cells_x, /* #blocks */
        mympi->num_cells_y, /* #elements per block */
        grid->ysize+2, /* #stride */
        MPI_DOUBLE, /* old type */
        &blocktype /* new type */ );
```

Hands-On II
Heat Conduction – Step 4: Validate

- Rebuild and rerun the example
- No energy lost anymore
- One should also rerun with MUST and ISP:
  - Extensions or corrections could introduce new errors
  - Neither of the tools should locate further errors

Hands-On II
Heat Conduction – Summary

- We detected 3 errors in the example
  - Buffer overlap
  - A schedule dependent deadlock
  - A semantic error in a communication
- Only the last error was visible from the start
- Tools aided, but they didn't put your nose onto the defects directly
  - In real debugging: tools help, scientific debugging guides in the search for the defect

Beyond MPI and Threads

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], and Joachim Protze[4]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Booth #2143

Booth #1508

Booth #3624

Booth #1030

Organization

- Introduction and Tools
- Hands-On I

Coffee Break

- The Tao of Debugging
- Systematic Debugging

Lunch Break

- Defects in Practice
- MPI Applications
- Threaded Applications

Coffee Break

- Hands-On II
- **Beyond MPI and Threads**
- Discussion and Wrap-Up

DATA RACE CHECKING OF GPU PROGRAMS

Contrast between CPUs and GPUs

Example: Increment Array Elements

Fine-grained threads scheduled to run like this:
tid = 0, 1, 2, 3, …

CPU program

```
void inc_cpu(float* a,
       float b, int N) {
  for (int idx = 0; idx<N; idx++)
    a[idx] = a[idx] + b;
}

void main() {
  .....
  increment_cpu(a, b, N);
}
```

CUDA program

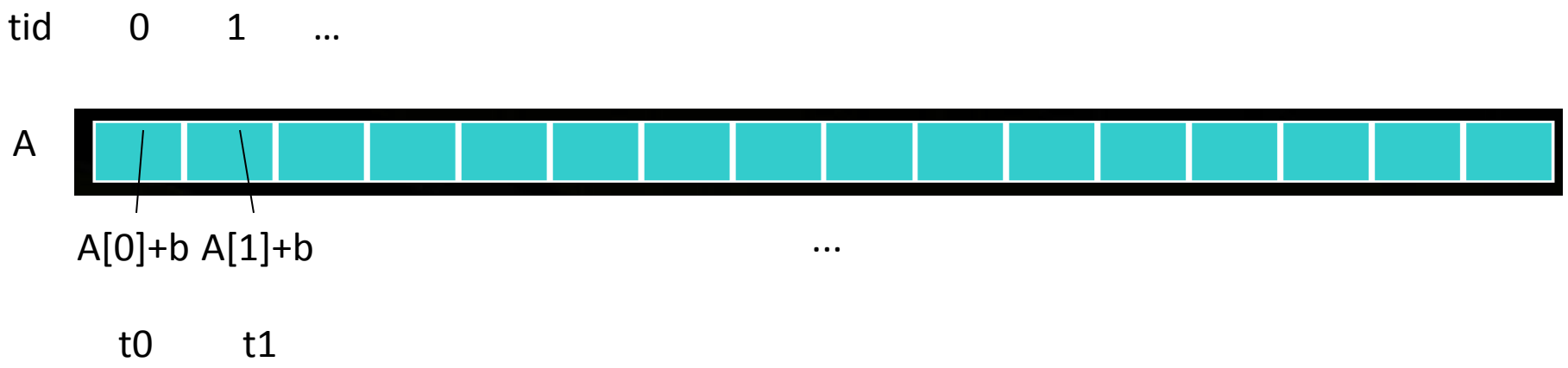
```
__global__ void inc_gpu(float* A, float b, int N) {
  int tid = blockIdx.x* blockDim.x+ threadIdx.x;
  if (tid < N)
    A[tid] = A[tid] + b;
}

voidmain() {
  .....
  dim3dimBlock (blocksize);
  dim3dimGrid( ceil( N / (float)blocksize) );
  increment_gpu<<<dimGrid, dimBlock>>>(a, b,
N);
}
```

GKLEE : A Symbolic Debugger for GPU programs

- GKLEE helps detect
  - Deadlocks
    - All threads in a thread block do not encounter the same Syncthread instruction
  - Data races
    - There are unsynchronized reads/writes
  - Illegal memory accesses
    - Memory accesses going out of bounds
- GKLEE outputs summary dot-graph “certificate”
  - Allows users to gain confidence in GKLEE’s checks

Example: Increment Array Elements

Increment N-element array A by scalar b (no race here)




```
__global__ void inc_gpu(int*A, int b, intN) {
   int idx = blockIdx.x * blockDim.x + threadIdx.x;
   if (idx < 64)  {
      A[idx] = A[idx] + b;
  }
```

Illustration of Race

Increment N-element vector A by scalar b (racing accesses here)

t0:
read A[63]

t63:
write A[63]

```
__global__ void inc_gpu(int*A, int b, int N) {
   int idx = blockIdx.x * blockDim.x +  threadIdx.x;

   if (idx < 64)  {
      A[idx] = A[(idx – 1 + 64) % 64] + b;
  }
```

t63 t0

Threads t63 and t0 have racing read/write at location A[63], causing unpredictable results!

1. Data race found/fixed: (Courtesy, GKLEE user Gaurav Gupta,IIT Delhi)

WITH DATA RACES

```
__global__ void pmul(...)
{
......
for(i=0;i<k;i++){
        product = m[i]*ndig;
        //low word
        radix_type lword = product%2;
        //high word
        radix_type hword = product/2;
        int ind = x+i;
        //lword
        radix_type carry =
                (inter_buf[ind]+lword)/2;
        inter_buf[ind] =
                (inter_buf[ind]+lword)%2;
        carry_buf[ind+1] += carry;

        //hword
        carry = (inter_buf[ind+1]+hword)/2;
        inter_buf[ind+1] =
                (inter_buf[ind+1]+hword)%2;

        carry_buf[ind+2] += carry;

}
}
```

GKLEE RESULT

```
********** Start checking races at
Device Memory **********

[GKLEE]: Under the pure canonical
schedule, within a block,
thread 0 and 1 incur a Write-Write race
with the same value (Benign) on
[GKLEE] Inst:
Instruction Line: 39, In File:
bkernels.cpp, With Dir Path: /
[File:bkernels.cpp, Line: 39, Inst:

inter_buf[ind+1] =
(inter_buf[ind+1]+hword)&2;]
 store i8 %98, i8* %103, align 1, !dbg
!1038
<W: 31930816, 1:0, b0, t0>
[GKLEE] Inst:
Instruction Line: 34, In File:
bkernels.cpp, With Dir Path: /
[File: bkernels.cpp, Line: 34, Inst:

inter_buf[ind] =
(inter_buf[ind]+lword)&2;]
 store i8 %59, i8* %63, align 1, !dbg
!1035
<W: 31930816, 1:0, b0, t1>
```

MODIFIED VERSION WITHOUT RACES:

```
__global__ void pmul(...)
{
.....
for(i=0;i<k;i++){
        product = m[i]*ndig;
        //low word
        radix_type lword = product%2;
        //high word
        radix_type hword = product/2;
        int ind = x+i;
        //lword
        radix_type carry =
                (inter_buf[ind]+lword)/2;
        inter_buf[ind] =
                (inter_buf[ind]+lword)%2;
        carry_buf[ind+1] += carry;
        __syncthreads();
        //hword
        carry = (inter_buf[ind+1]+hword)/2;
        inter_buf[ind+1] =
                (inter_buf[ind+1]+hword)%2;

        carry_buf[ind+2] += carry;

}
}
```

2. Out of bounds pointer error found/fixed (Courtesy, GKLEE user Gaurav Gupta, IIT Delhi)

BUGGY

```
__global__ void padd(...){
....
for(i=x;i<x+batch_size && i<bl;i++){
        radix_type sum = m[i] + n[i];
        res[i] = sum%2;
        __syncthreads();
        res[i+1] += sum/2;

}
}
```

GKLEE RESULT

type:2 flow:0 instr: %63 = load i8* %62, align 1, !dbg !1032 cond: merged:
KLEE: ERROR: bkernels.cpp:57: memory error: **out of bound pointer**
KLEE: NOTE: now ignoring this error at this location

MODIFIED VERSION:

```
__global__ void padd(...){
...
for(i=x;i<x+batch_size && i<bl;i++){
        radix_type sum = m[i] + n[i];
        res[i] = sum%2;
        __syncthreads();
        i+1!=bl?res[i+1] += sum/2:j=1;
}
}
```

3. Memory out of bounds due to type-error (Courtesy, GKLEE user Gaurav Gupta, IIT Delhi)

BUGGY

```
typedef char radix_type;
__global__ void pmul(...)
{
......
for(int i=0;i<k;i++){
        ...
radix_type  ind = x+i; //-128<=ind<128
        //lword
        radix_type carry =
                (inter_buf[ind]+lword)/2;
        inter_buf[ind] = //ind= -128
                (inter_buf[ind]+lword)%2;
        carry_buf[ind+1] += carry;
        __syncthreads();
        //hword
        carry = (inter_buf[ind+1]+hword)/2;
        inter_buf[ind+1] =
                (inter_buf[ind+1]+hword)%2;

        carry_buf[ind+2] += carry;

}
}
```

GKLEE RESULT

```
type:2  flow:0  instr:  %43 = load i8*
%42, align 1, !dbg !1034  cond: merged:
KLEE: ERROR: bkernels.cpp:33: memory
error: out of bound pointer
KLEE: NOTE: now ignoring this error at
this location
```

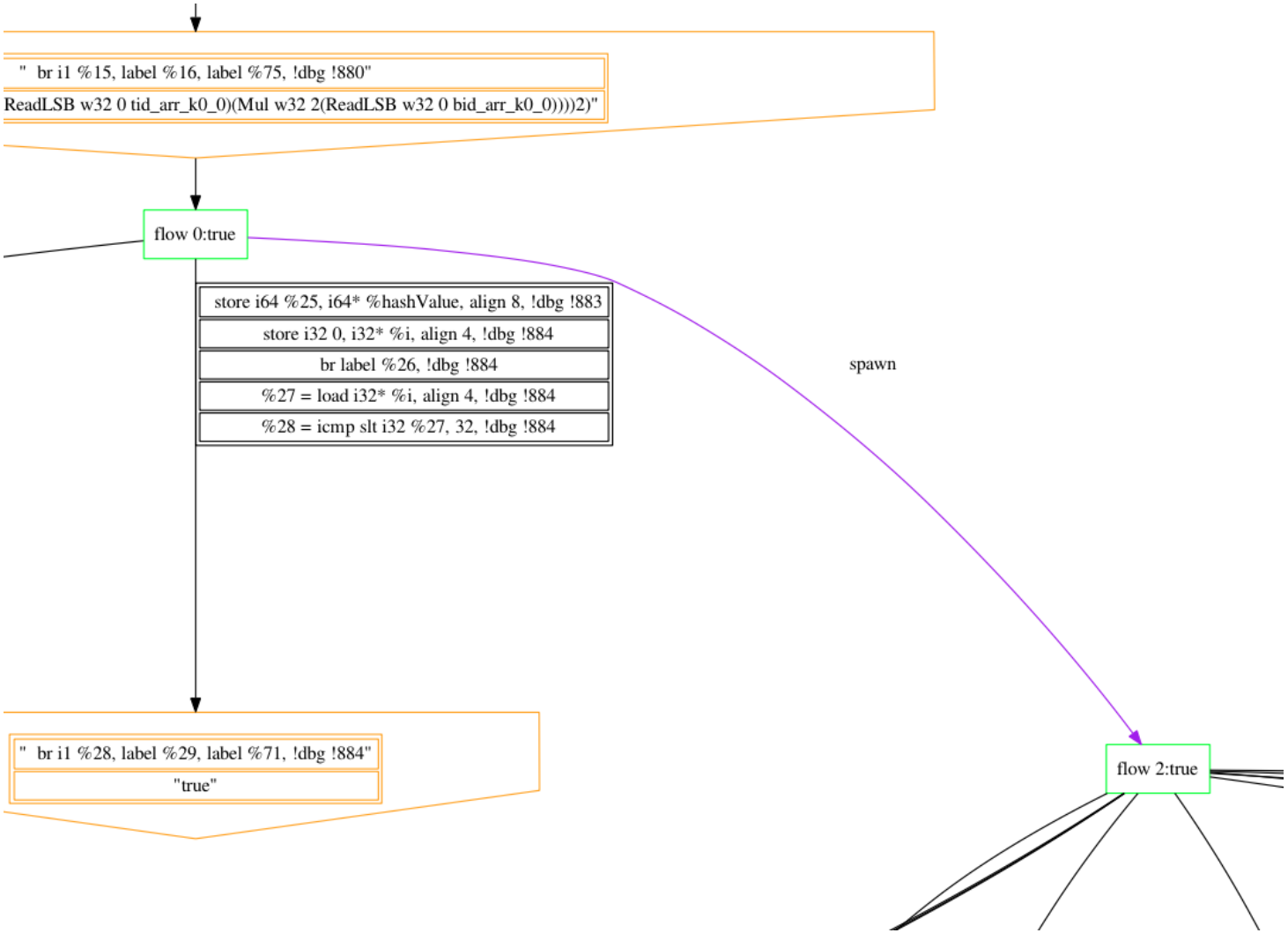
MODIFIED VERSION:

```
typedef char radix_type;
__global__ void pmul(...)
{
......
for(i=0;i<k;i++){
        ...
int ind = x+i;//-32,768<=ind<=32,767
        //lword
        radix_type carry =
                (inter_buf[ind]+lword)/2;
        inter_buf[ind] =
                (inter_buf[ind]+lword)%2;
        carry_buf[ind+1] += carry;
        __syncthreads();
        //hword
        carry = (inter_buf[ind+1]+hword)/2;
        inter_buf[ind+1] =
                (inter_buf[ind+1]+hword)%2;

        carry_buf[ind+2] += carry;

}
}
```

GKLEE with Static Analysis ("SESA") Results

- **We have been able to run SESA on**
  - **Lonestar Benchmarks (UT)**
  - **Parboil Benchmarks (UIUC)**
    - **It scales well and finds issues**
      - **Races**
      - **Out of bounds accesses**

GKLEE's "certificate" of its work



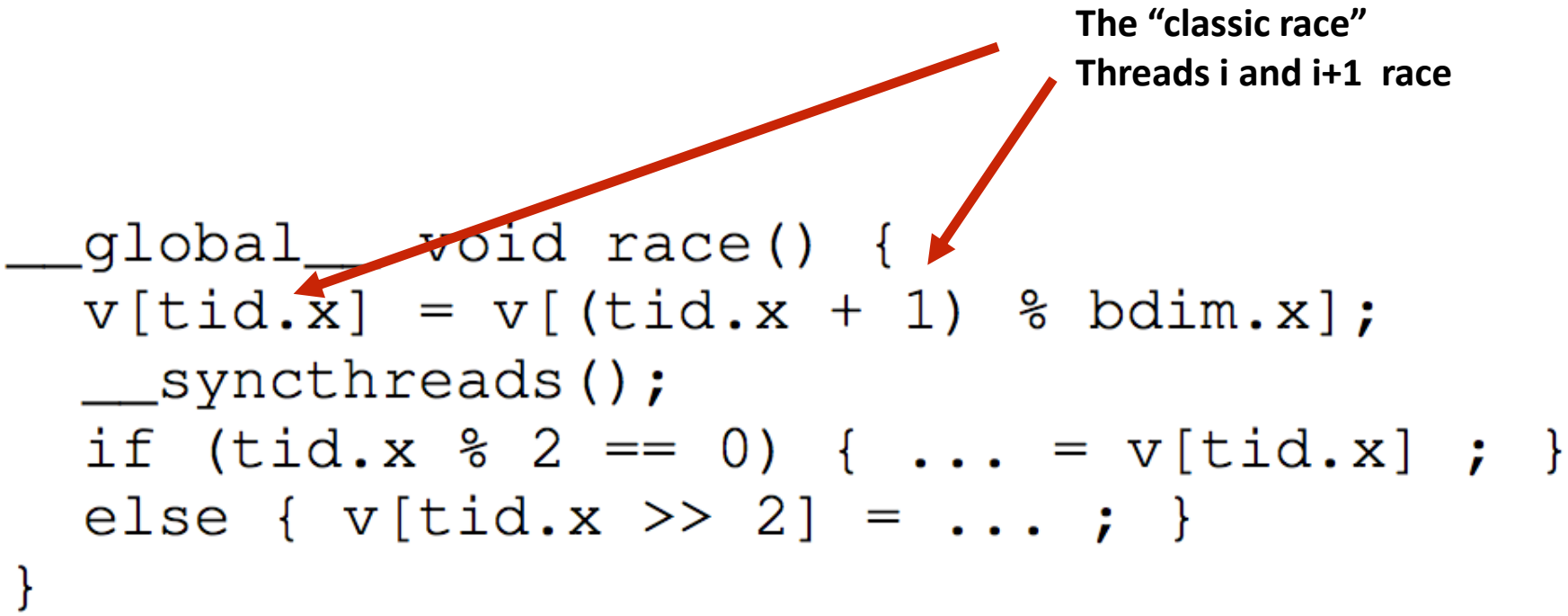
GKLEE Details (as time permits)

- We now present the internals of GKLEE
- Overview of schedule-generation
- Parameterized flows (handles scale)
- Combination with static-analysis

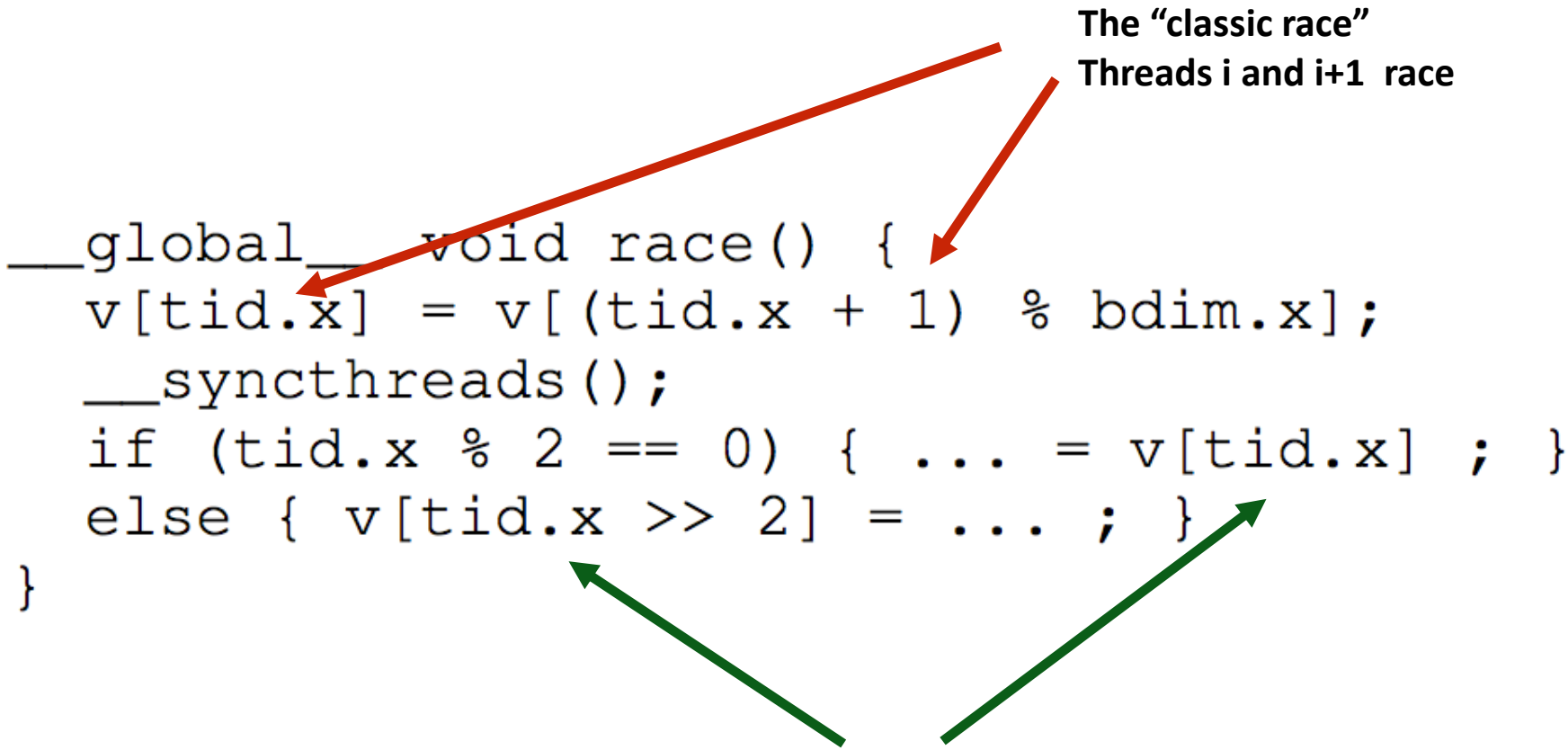
An Example with Two Data Races

```
__global__ void race() {
  v[tid.x] = v[(tid.x + 1) % bdim.x];
  __syncthreads();
  if (tid.x % 2 == 0) { ... = v[tid.x] ; }
  else { v[tid.x >> 2] = ... ; }
}
```

An Example with Two Data Races



An Example with Two Data Races

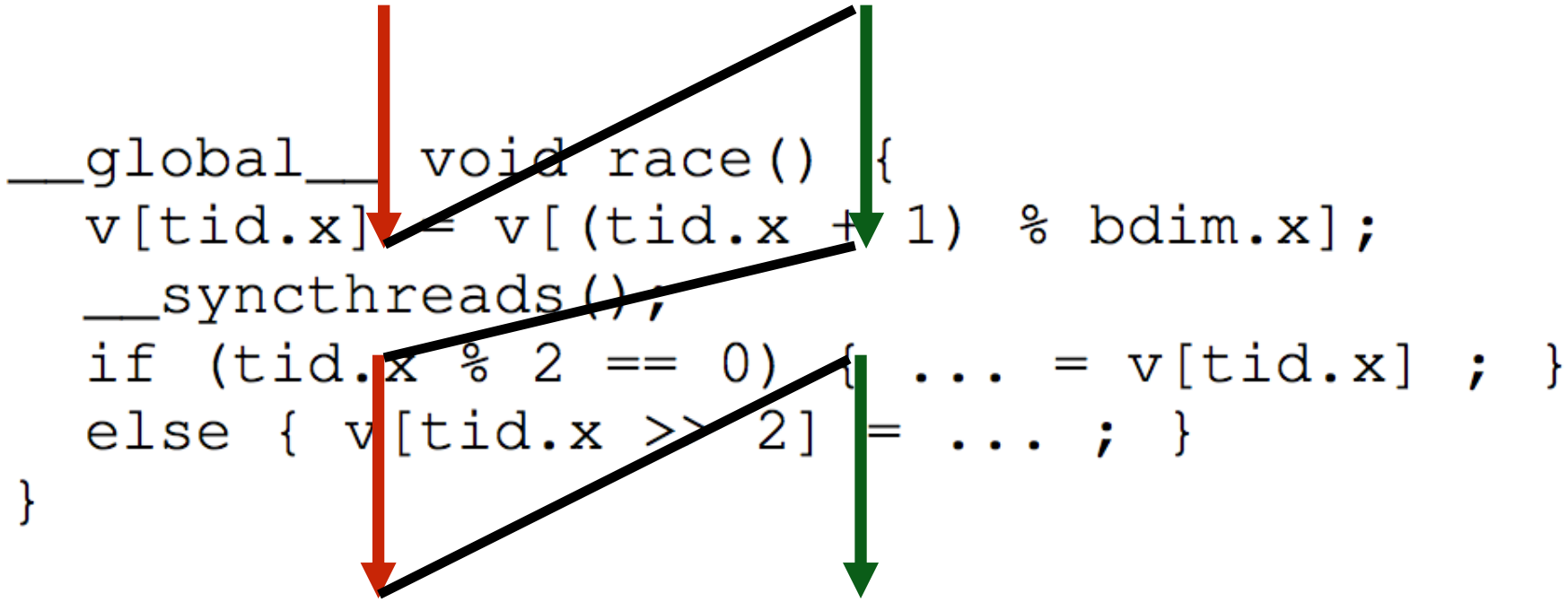


GKLEE's steps

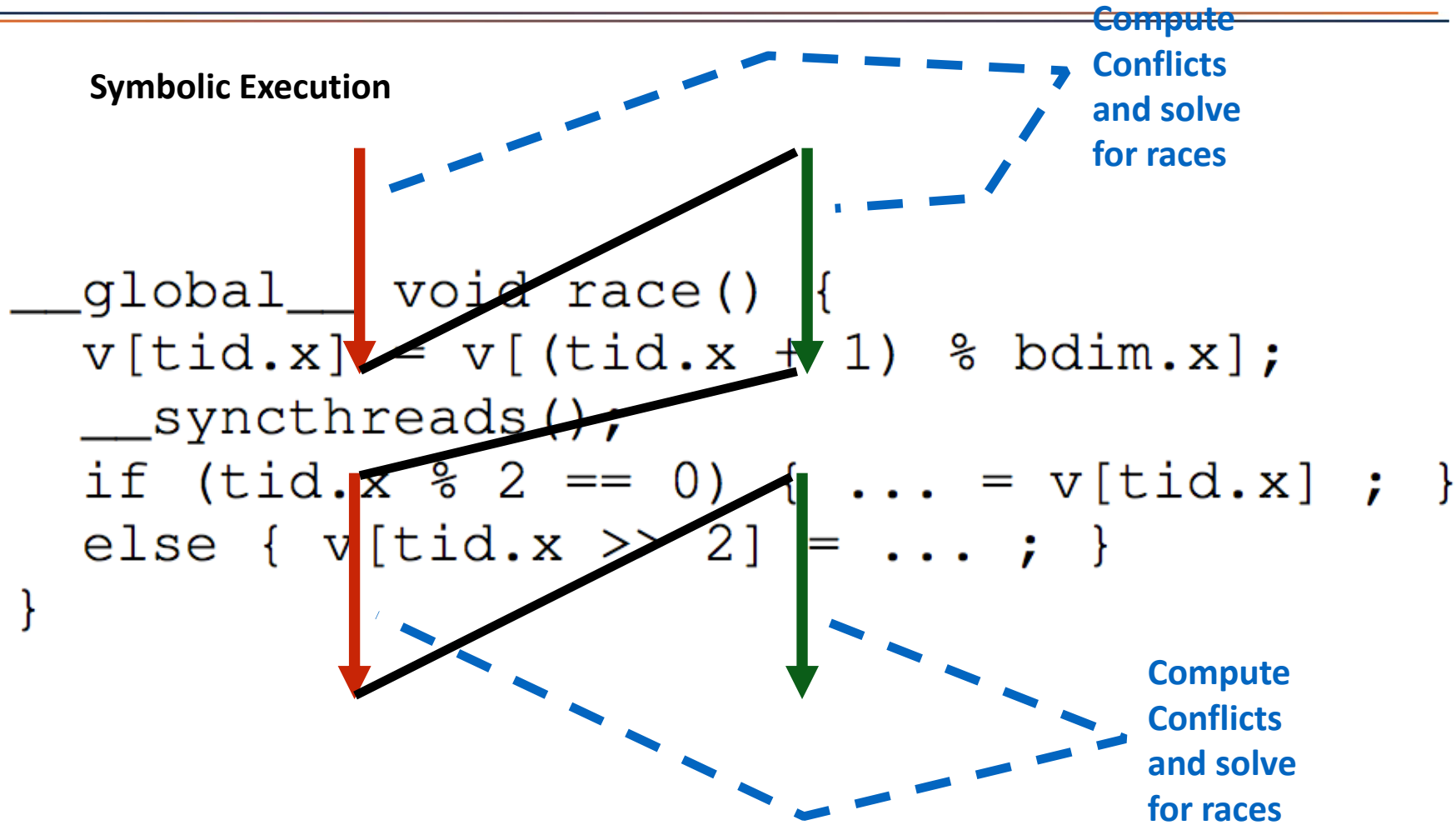
```
__global__ void race() {
  v[tid.x] = v[(tid.x + 1) % bdim.x];
  __syncthreads();
  if (tid.x % 2 == 0) { ... = v[tid.x] ; }
  else { v[tid.x >> 2] = ... ; }
}
```

GKLEE's steps (PPoPP'12)

Symbolic Execution



GKLEE's steps



Race Checking under Parameterized Flows (GKLEEp, SC'12) helps handle large # of threads


```
// a[4 * 2048] in device memory;
// b[2048] in shared memory;
__global__ void test(unsigned * a) {
1:   unsigned bid = blockIdx.x;
2:   unsigned tid = threadIdx.x;
3:
4:   if (bid % 2 != 0) {
5:     if (tid < 1024) {
6:       unsigned idx = bid * blockDim.x + tid;
7:       b[tid] = a[idx] + 1;    // Write of Race-1
8:       if (tid % 2 != 0) {
9:         b[tid] = 2;           // Write of Race-2
10:      } else {
11:        if (tid > 0)
12:          b[tid] = b[tid-1]+1; // Read of Race-2
13:        }
14:    } else {
15:      b[tid] = b[tid-1];       // Read of Race-1
16:    }
17:  } else {
18:    unsigned idx = bid * blockDim.x + tid;
19:    b[tid] = a[idx] + 1;
20:  }
}
```

Fig. 2. The motivating example

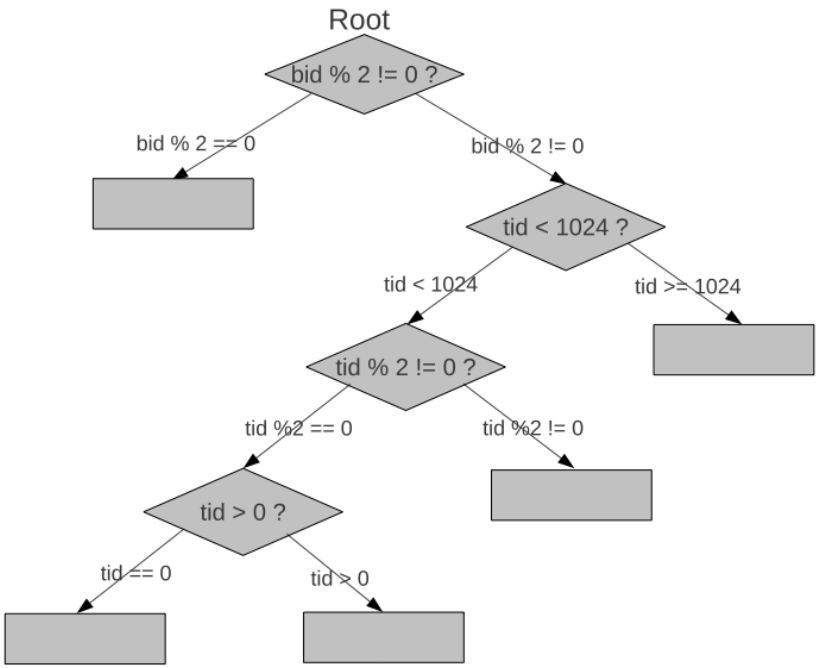


Fig. 3. Parametric flows for the motivating example

Race Checking under Parameterized Flows (GKLEEp, SC'12)

Keep two symbolic threads per flow-group and race-check per flow.

```
// a[4 * 2048] in device memory;
// b[2048] in shared memory;
__global__ void test(unsigned * a) {
1:   unsigned bid = blockIdx.x;
2:   unsigned tid = threadIdx.x;
3:
4:   if (bid % 2 != 0) {
5:     if (tid < 1024) {
6:       unsigned idx = bid * blockDim.x + tid;
7:       b[tid] = a[idx] + 1;    // Write of Race-1
8:       if (tid % 2 != 0) {
9:         b[tid] = 2;           // Write of Race-2
10:      } else {
11:        if (tid > 0)
12:          b[tid] = b[tid-1]+1; // Read of Race-2
13:        }
14:    } else {
15:      b[tid] = b[tid-1];       // Read of Race-1
16:    }
17:  } else {
18:    unsigned idx = bid * blockDim.x + tid;
19:    b[tid] = a[idx] + 1;
20:  }
}
```

Fig. 2. The motivating example

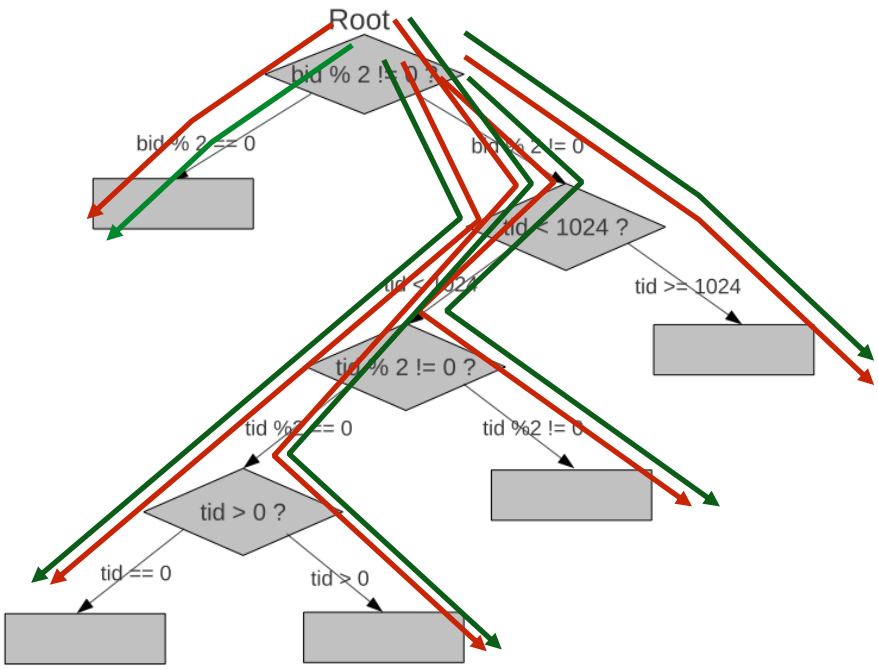
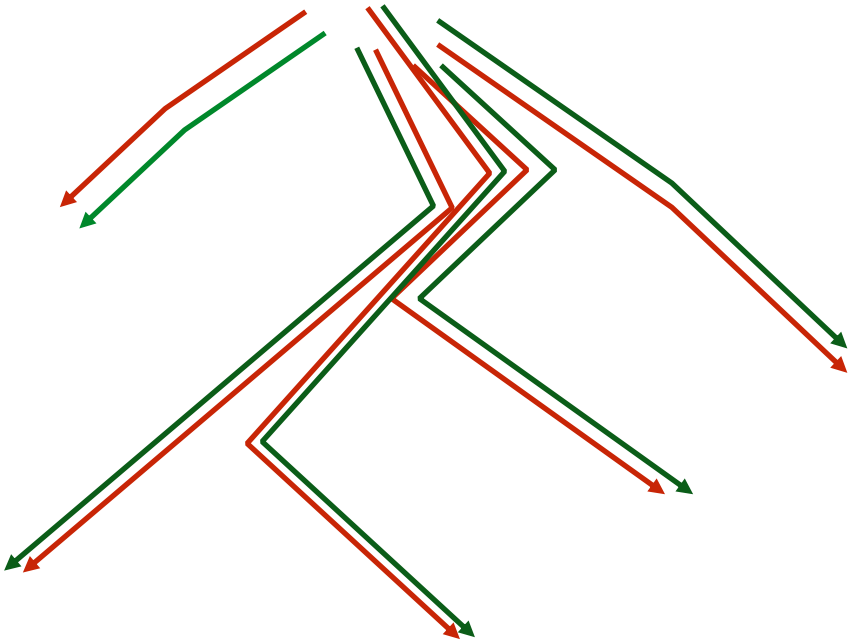


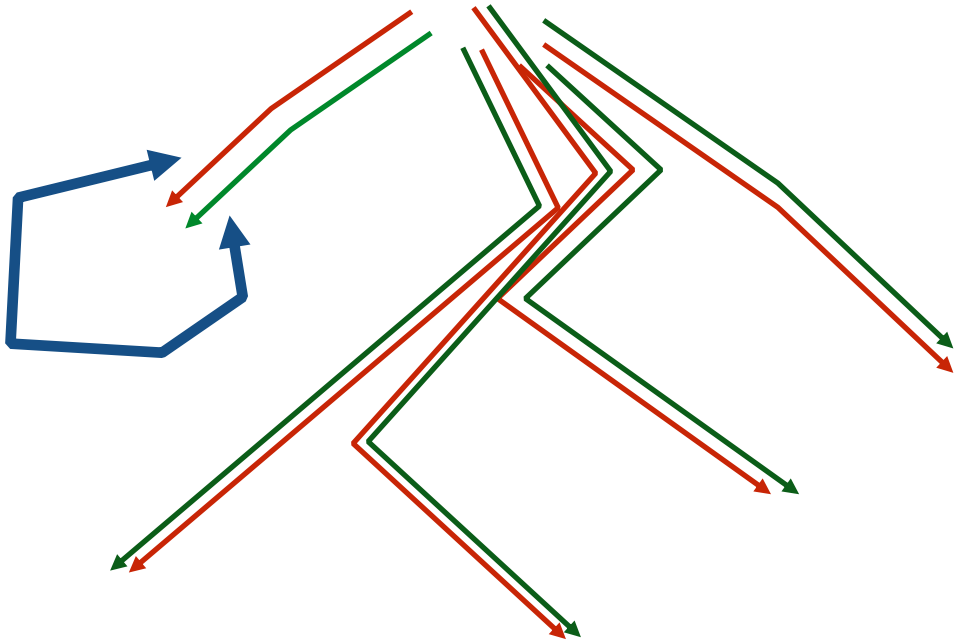
Fig. 3. Parametric flows for the motivating example

Race Checking under Parameterized Flows (GKLEEp, SC'12)



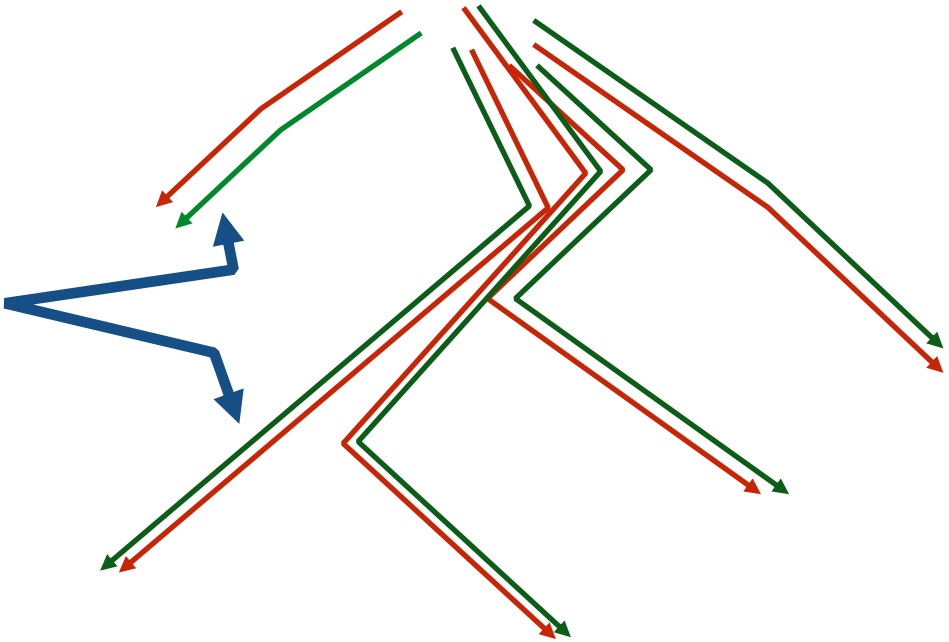
Race Checking under Parameterized Flows (GKLEEp, SC'12)

Intra-Flow Race-checks are done



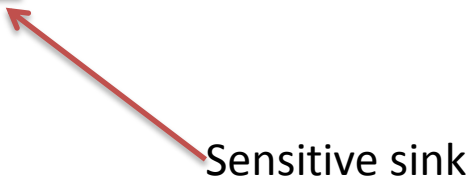
Race Checking under Parameterized Flows

Also,
Inter-Flow
Race-checks
are done



SESA (SC'14): GKLEE with Static Analysis

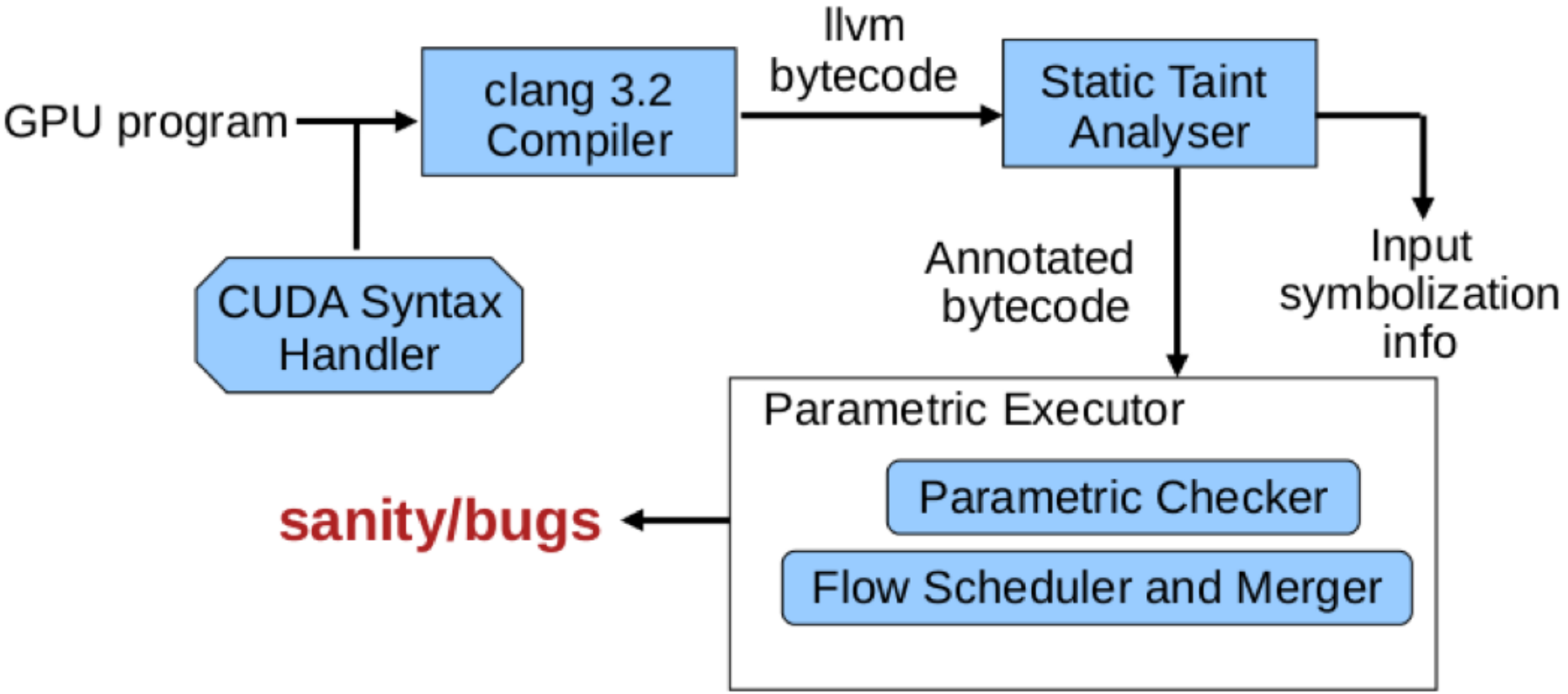
```
__global__ void set_sphere(int cx, int cy, int cz, int r, bool* out)
{
  int x  =threadIdx.x + blockIdx.x*blockDim.x - r + cx;
  int y = threadIdx.y + blockIdx.y*blockDim.y - r + cy;
  int z = threadIdx.z + blockIdx.z*blockDim.z - r + cz;

  bool inside = false, outside = false;

  // Test if block is on surface of sphere
  for(int dx = 0; dx < 2; ++dx)
    for(int dy = 0; dy < 2; ++dy)
      for(int dz = 0; dz < 2; ++dz)
      {
        int vertex_dist =(x+dx - cx)*(x+dx - cx) +
                         (y+dy - cy)*(y+dy - cy) +
                         (z+dz - cz)*(z+dz - cz);
        if(vertex_dist <= r*r)
          inside = true;
        else
          outside = true;
      }

  int idx = threadIdx.z*blockDim.x*blockDim.y
    + threadIdx.y*blockDim.x + threadIdx.x;

  out[idx] = inside && outside;
}
```



- Further enhances scalability of GKLEE
- Merges "un-necessary" flows back together

- Uses static analysis to determine control flow paths which do not contribute to a race
- Determines "sensitive sinks" and annotates branch instructions which do not modify these values
- Annotations allow Gkleep to prune flows not modifying sensitive variables
- In the example, dozens of flows could be created in the loop code
- The array indexing variable is not modified in the loop, allowing Gkleep to explore 1 path for the purpose of data race checking

GKLEE with Static Analysis (called “SESA” or Symbolic Execution with SA), SC’14 paper



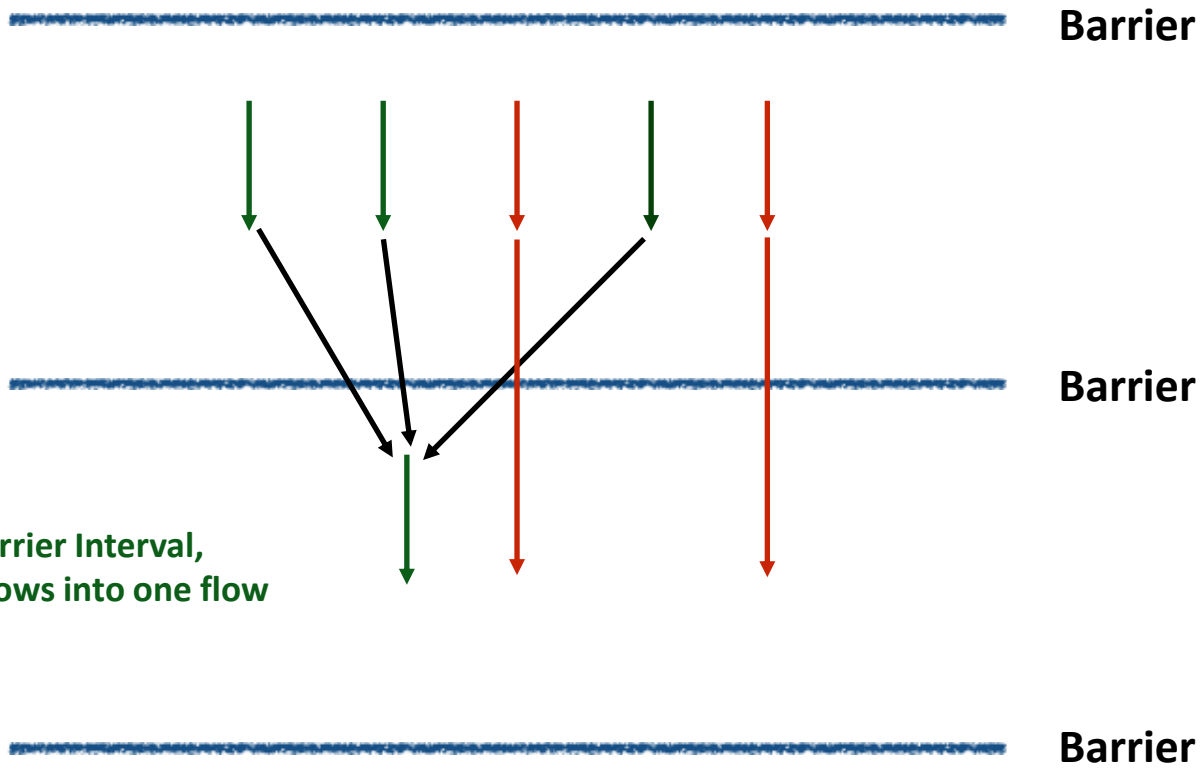
How SESA works

- **Static Analysis Pass marks how vars affected by each flow in a barrier interval may affect the generation of addresses in the next barrier interval**

There are two classes of flows:

(1) Flows that modify Global or Shared Var That flow into Control Predicates or Array Indexing Positions

(2) Flows that don't do so



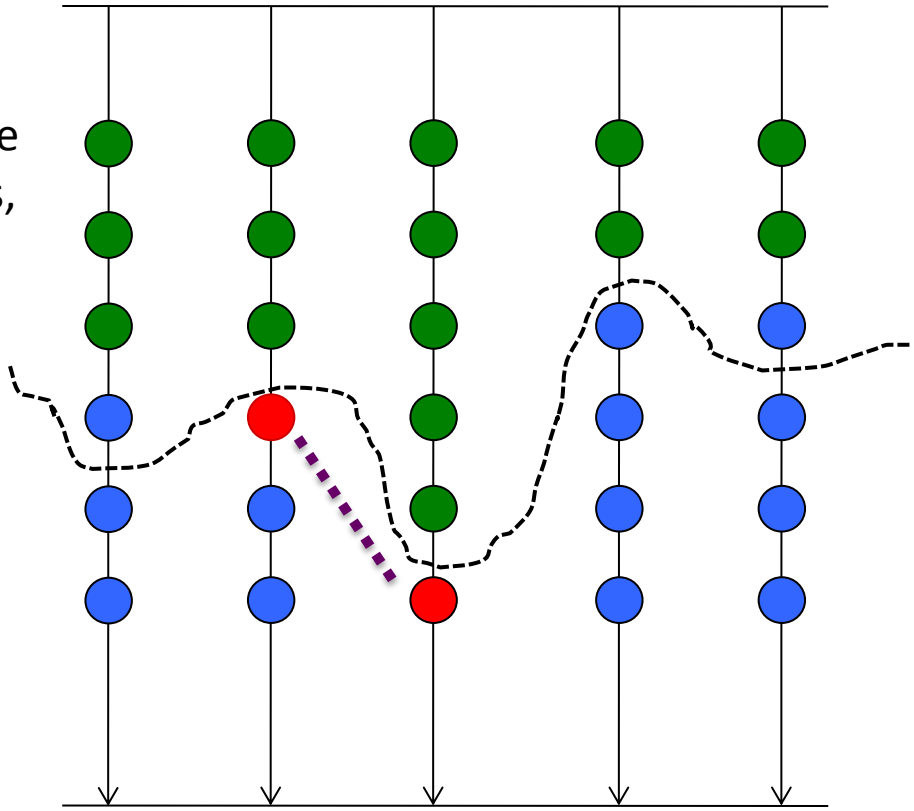
Within the next Barrier Interval, "OR" the Green Flows into one flow

Still more details

- For those who really want to understand the schedule-generation process of GKLEE and what guarantees it offers

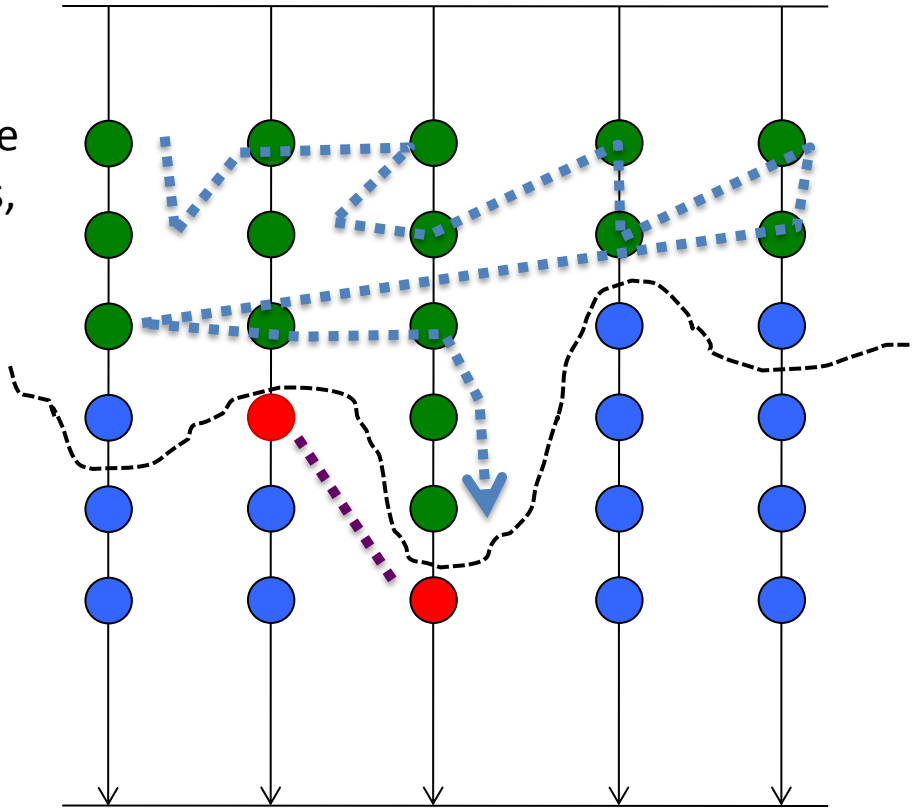
Solution to the interleaving problem: Find *representative* interleavings

For Example:
If the green dots are local thread actions, then all schedules that arrive at the "cut line" are equivalent!



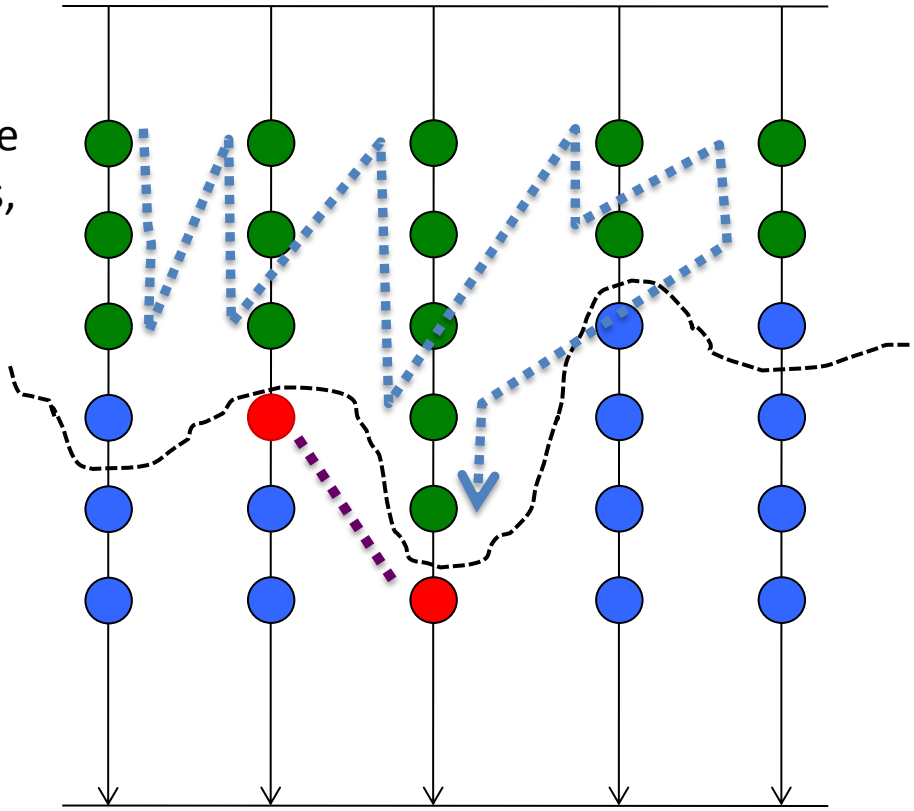
Solution to the interleaving problem: Find *representative* interleavings

For Example:
If the green dots are local thread actions, then all schedules that arrive at the "cut line" are equivalent!

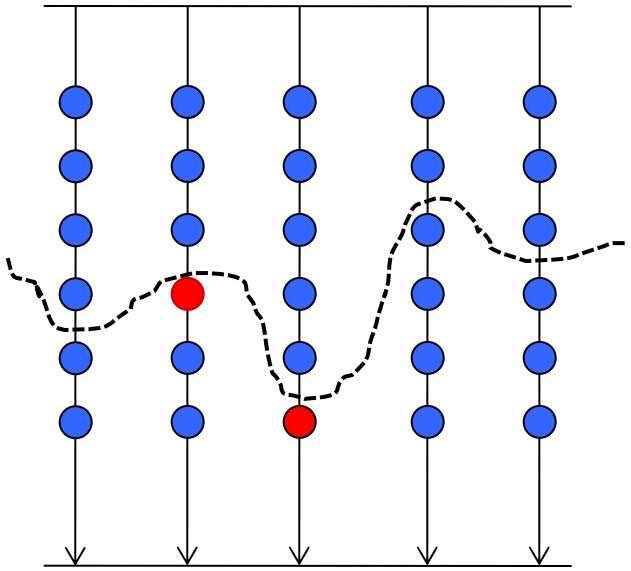


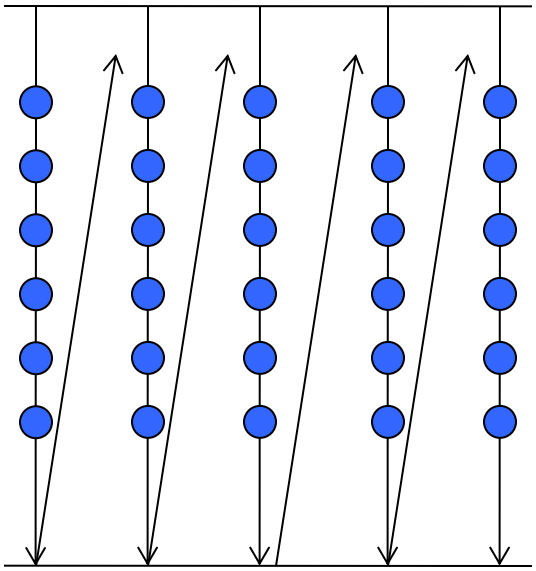
Solution to the interleaving problem: Find *representative* interleavings

For Example:
If the green dots are local thread actions, then all schedules that arrive at the "cut line" are equivalent!



GKLEE Avoids Examining Exp. Schedules !!





Instead of considering all Schedules and All Potential Races…

Consider JUST THIS SINGLE CANONICAL SCHEDULE !!

Folk Theorem (proved in our paper): "We will find A RACE If there is ANY race" !!

ACCELERATORS AND ALLINEA DDT

Beyond MPI and Threads
Accelerators, Threads and The Future

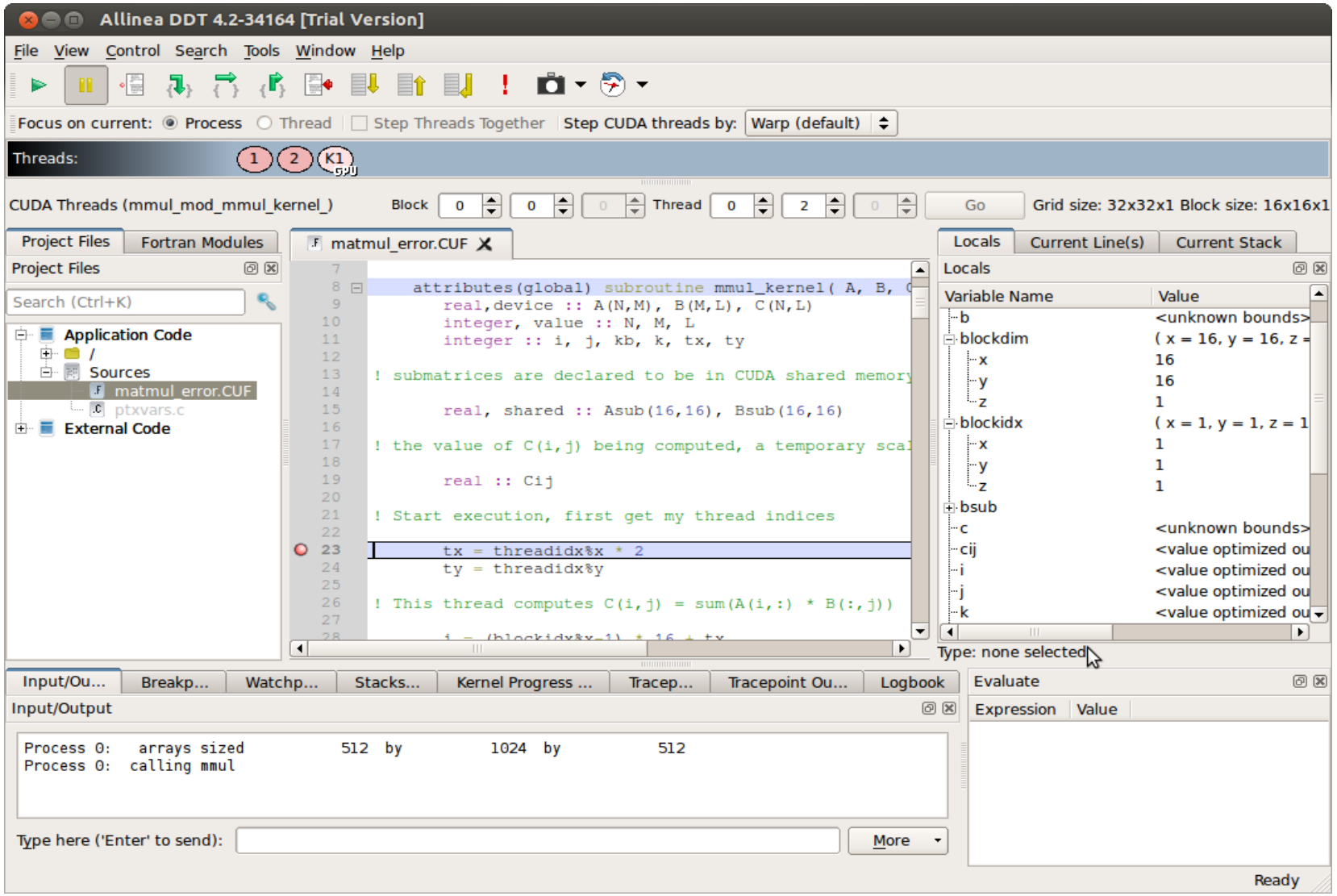
- Two major technologies
  - NVIDIA GPUs
  - Intel Xeon Phi

- What impact do they have on today's techniques and tools?

Beyond MPI and Threads
Impact on bugs

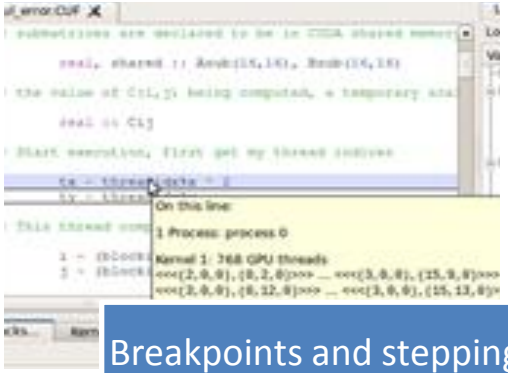
- NVIDIA - OpenACC and CUDA
  - Data transfer, coherency, race conditions, new memory classes, synchronization …
  - 100,000+-way parallelism within nodes
- Intel - OpenMP
  - Shared variables, coherency, critical regions, race conditions, synchronization …
  - Almost identical to current CPU threading – but 64-256 threads per CPU

Beyond MPI and Threads
GPUs - Familiar debugging support …

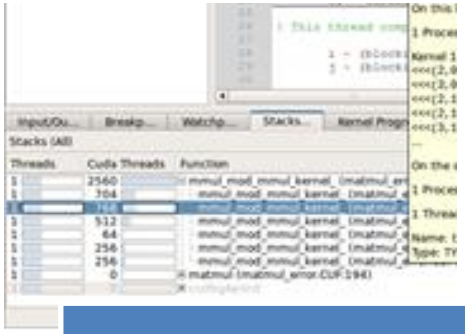


Beyond MPI and Threads
Things to Try for GPU Debugging

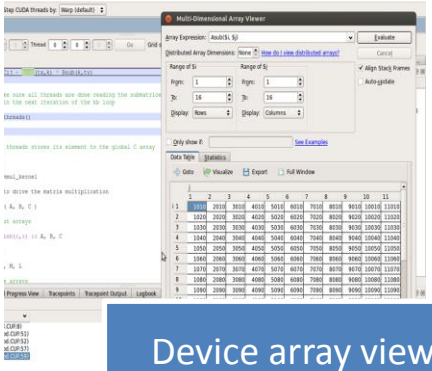
Variable inspection



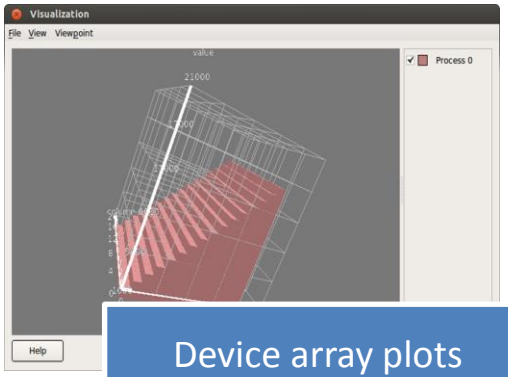
Breakpoints and stepping



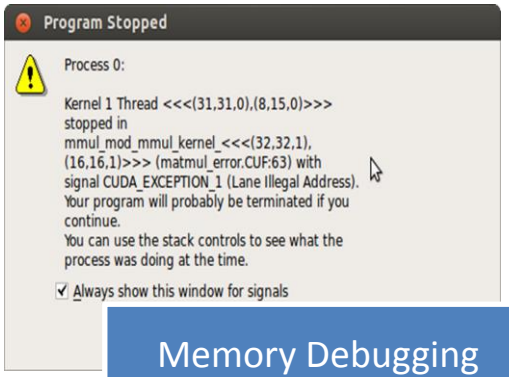
Thread display



Device array viewing



Device array plots



Memory Debugging

Beyond MPI and Threads
NVIDIA GPU Debugging: Compilers

High-level languages	OpenACC, CUDA Fortran, C and C++ - full support
Low-level languages	OpenCL – no support
Tools	NVIDIA - cuda-gdb and Nsight
	Allinea DDT
	Rogue Wave

Discussion and Wrap-Up

Matthias S. Müller[4], David Lecomber[3], Bronis R. de Supinski[2], Tobias Hilbrich[5],
Ganesh Gopalakrishnan[1], Mark O'Connor[3], and Joachim Protze[4]

University of Utah[1]
Lawrence Livermore National Laboratory[2]
Allinea Software Ltd[3]
RWTH Aachen University[4]
Technische Universität Dresden[5]

Organization

- Introduction and Tools
- Hands-On I

Coffee Break

- The Tao of Debugging
- Systematic Debugging

Lunch Break

- Defects in Practice
- MPI Applications
- Threaded Applications

Coffee Break

- Hands-On II
- Beyond MPI and Threads
- **Discussion and Wrap-Up**

Wrap-up and Discussion
Conclusions

- Tools can aid in HPC code debugging:
  - MUST checks complex restrictions for MPI
  - ISP catches interleaving dependent deadlocks
  - DDT memory debugging + full fledged debugger
  - Inspector XE detects threading deadlocks and races
  - Archer detects threading races
  - STAT gathers stack traces at extreme scales
- Scientific debugging:
  - Feel free to be guided by intuition for a set time
  - Afterwards: form hypothesis, run experiments to test them, keep a logbook

Wrap-up and Discussion Pointers

- Get MUST:
  - https://www.itc.rwth-aachen.de/MUST
- Get ISP:
  - http://www.cs.utah.edu/fv/ISP-release
- Get a demo of Allinea DDT:
  - http://www.allinea.com/products/downloads
- Get STAT:
  - https://computing.llnl.gov/code/STAT/
- Get a trial of Intel Inspector XE:
  - https://software.intel.com/en-us/intel-inspector-xe
- Get Archer:
  - https://github.com/soarlab/Archer
- Get GKLEE (or remotely run thru web):
  - http://www.cs.utah.edu/fv/GKLEE

Introduction and Tools
More Debugging on Monday

- More debugging …
  - Monday 1:30pm-5pm, room “14”
  - SC Tutorial: *“Debugging and Performance Tools for MPI and OpenMP 4.0 Applications for CPU and Accelerators/Coprocessors.”*
  - In-depth exploration of parallel debugging and optimization focused on techniques that can be used with accelerators and coprocessors

Wrap-up and Discussion
Meet us at our Booths, Let us know your feedback

Allinea booth #1508

University of Utah booth #2143

ZIH (Dresden) booth #3624

DOE booth (LLNL participation) #1030