RED ESPAÑOLA DE SUPERCOMPUTACIÓN

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Parallel Debugging with TotalView

-

BSC-CNS

AGENDA

- What debugging means?
- Debugging Tools in the RES
- Allinea DDT as alternative
- Introduction to TotalView (RogueWave Software)
  - ✓ What is TotalView
  - ✓ Compiling Your Program
  - ✓ Starting totalview
  - ✓ TotalView's Basic Look and Feel
  - ✓ Basic usage: Action points, groups, navigating the code...
  - ✓ Built-in variables and statements
  - ✓ Expression Evaluation and Code Fragments
  - ✓ Memory Debugging
  - ✓ Remote Display
  - ✓ Some notes on CLI
- Hands on

What debugging means?

Debugging 1.0: is a methodical process of finding and reducing the number of bugs.

Originally It literally meant the process to eliminate “bugs” … like this:

What debugging means?

Debugging 2.0: Is the process to confirm all the things that you believe are true .... because there is, at least one, that is not

Things that you believe:

- This variable has been set before entering the loop
- This variable is only written by master process
- I am sending the right data type in all MPI communications

Debugging Tools

Serial Debugging:

- **Printf()**
- gdb and its frontends (DDD)
- Others ...

Parallel Debugging:

Both serial and parallel debuggers are useful.

Serial debuggers, like gdb, are what most programmers are used to, but parallel debuggers can attach to all the individual processes in an MPI job simultaneously, treating the MPI application as a single entity.

TOTALVIEW TECHNOLOGIES

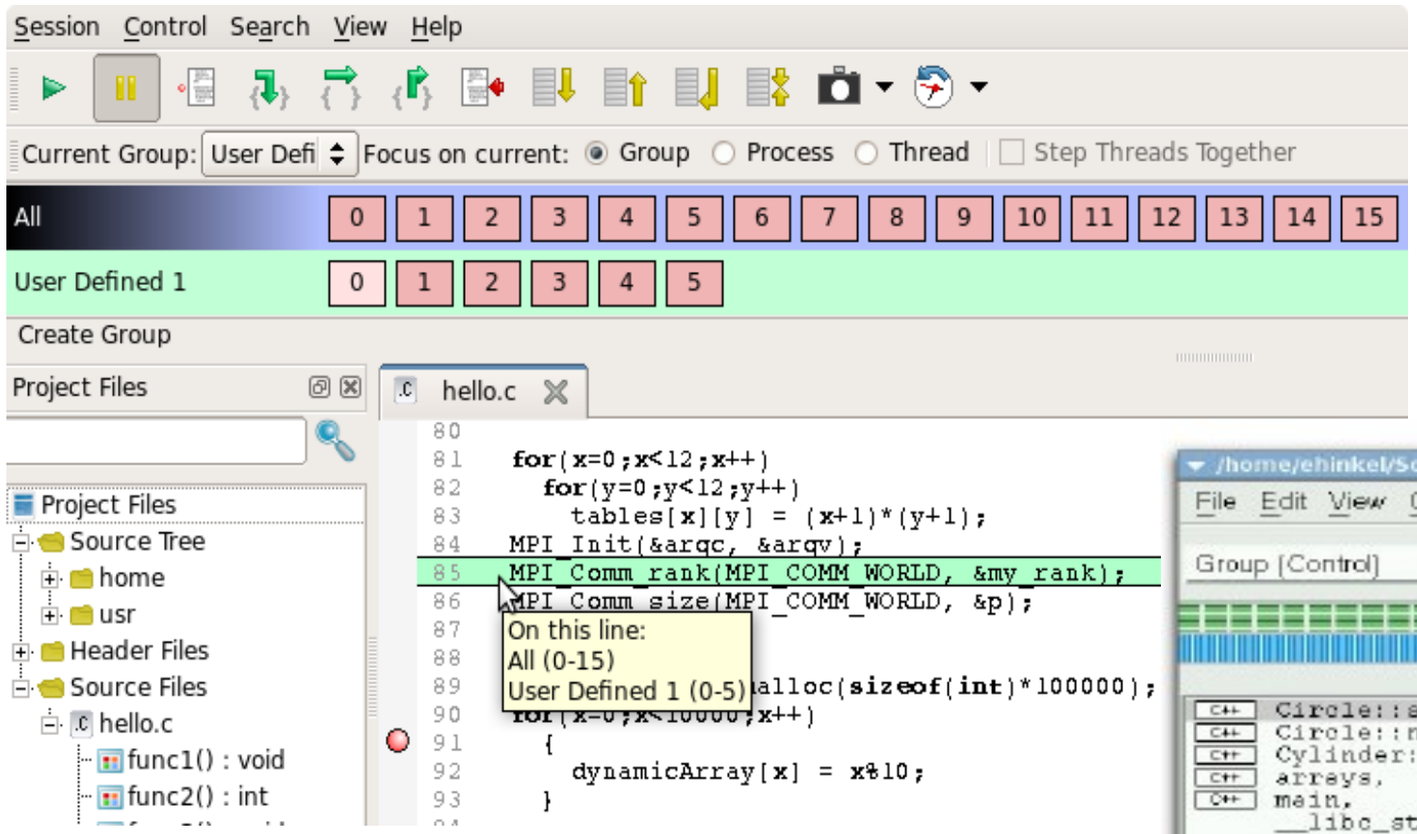
Linux Power TotalView 8.7.0-7

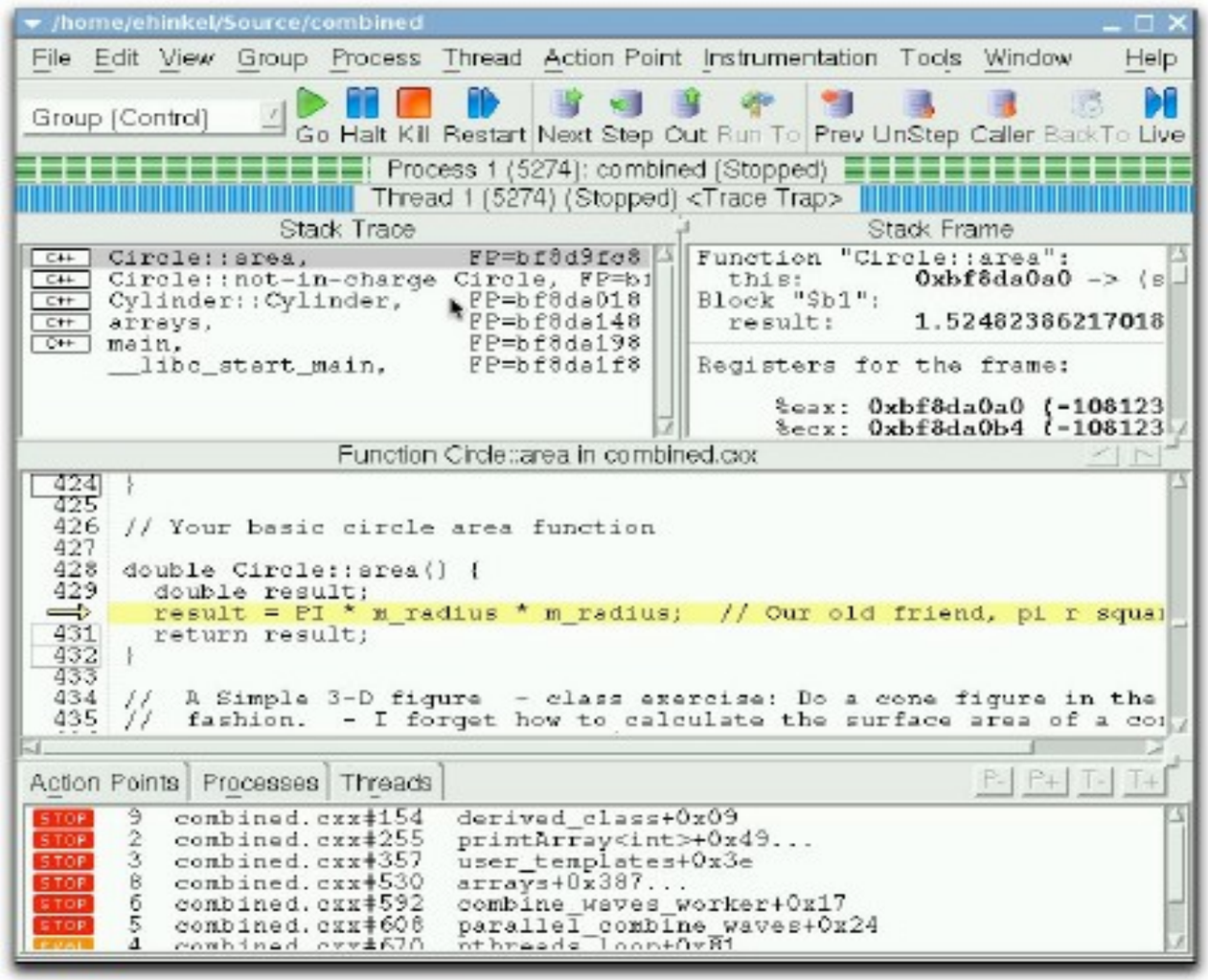
allinea ddt
the distributed debugging tool

(c) Allinea Software 2002-2009

Version: 2.4.1
Build: Suse 10 ppc64
Build Date: Mar 20 2009

Debugging Tools





DDT as alternative

DDT (Distributed Debugging Tool) from the Allinea Corporation

- Parallel debugger which provides many of the same basic features as Totalview, as well as some new elements.

- Totalview has a much larger feature set than DDT:
  - with debugging capability for more than one executable at a time
  - machine level language support
  - Tcl command line options
  - other advanced components …

But..

These sometimes are not the primary reasons why scientists need a parallel debugger.

Why? … Because ...

Most scientists need a simple and user friendly way to set breakpoints, step through code and halt execution while they examine variable values and code logic across different processors.

DDT as alternative

Some Features:

- DDT has an more intuitive user interface, especially for beginners
- Different ways of navigating through the processes
- None of both interfaces are suitable to debug apps with hundreds of processors (might become cumbersome)
- Both allow to dive into distributed multidimensionals arrays, subarrays, slices.
- And many more …

http://www.allinea.com/index.php?page=48

http://www.totalviewtech.com/support/documentation.html

Introduction to TotalView

What is TotalView?

- TotalView is a sophisticated software debugger product of RogueWave Software
- Used for debugging and analyzing both serial and parallel programs.
  - Multi-threaded Debugging
  - Parallel Debugging: MPI, PVM, Others
- Especially designed for use with complex, multi-process and/or multi-threaded applications.
- Wide compiler & platform support
  - C, C++, Fortran 77 & 90, UPC
  - Unix, Linux, OS X
- Reverse debugging (Replay Engine)
- Long distance remote debugging
- Unattended Batch Debugging
- TVD along with DDT are the most popular HPC debuggers to date.

Compiling your program

Always compile with -g and -O0

- O0 is important because with some optimizations, even when they not modify the code semantics, the source code may not reflect what is really happening.

- In the IBM compilers, some optimizations levels might alter the code semantics. That's why it is important to use -qstrict when using -O3

- TVD can debug code compiled without -g but assembler will be shown

Starting Totalview

TVD must be sent through the batch system

➔ Connect to MareNostrum using -X option:
ssh -X bsc99704@mn4.bsc.es

➔ Jump to a node above login4 (from login5 to login8)
ssh -Y login6

➔ Sumbit the batch script:
mnsubmit *run.sh*

```
1 #!/bin/bash
2 # @ job_name = simple2
3 # @ initialdir = ./
4 # @ output = mpi_%j.err
5 # @ error = mpi_%j.err
6 # @ total_tasks = 4
7 # @ cpus_per_task = 4
8 # @ wall_clock_limit = 01:10:00
9 # @ mining_level = 0
10# @ x11 = 1
11
12
13 /gpfs/apps/TOTALVIEW/totalview  -mpi SLURM -np 4 ./simple -a “Prueba de TVD”
```

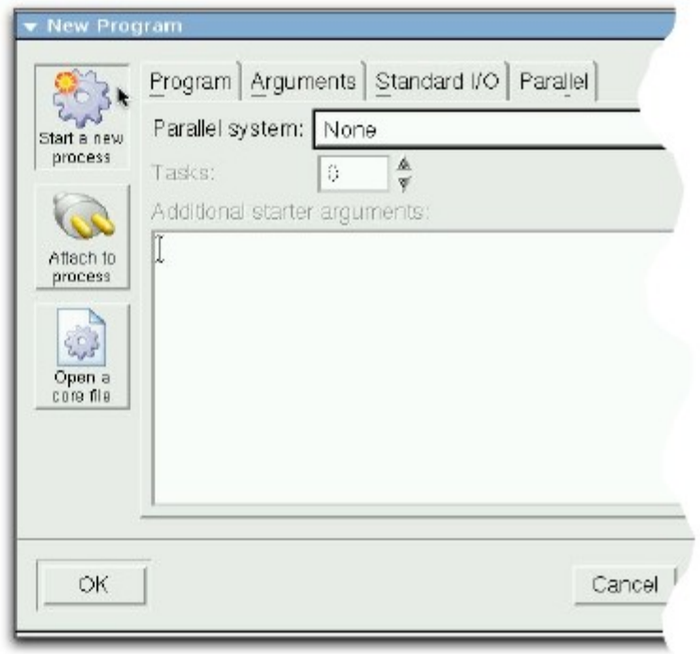
Starting Totalview

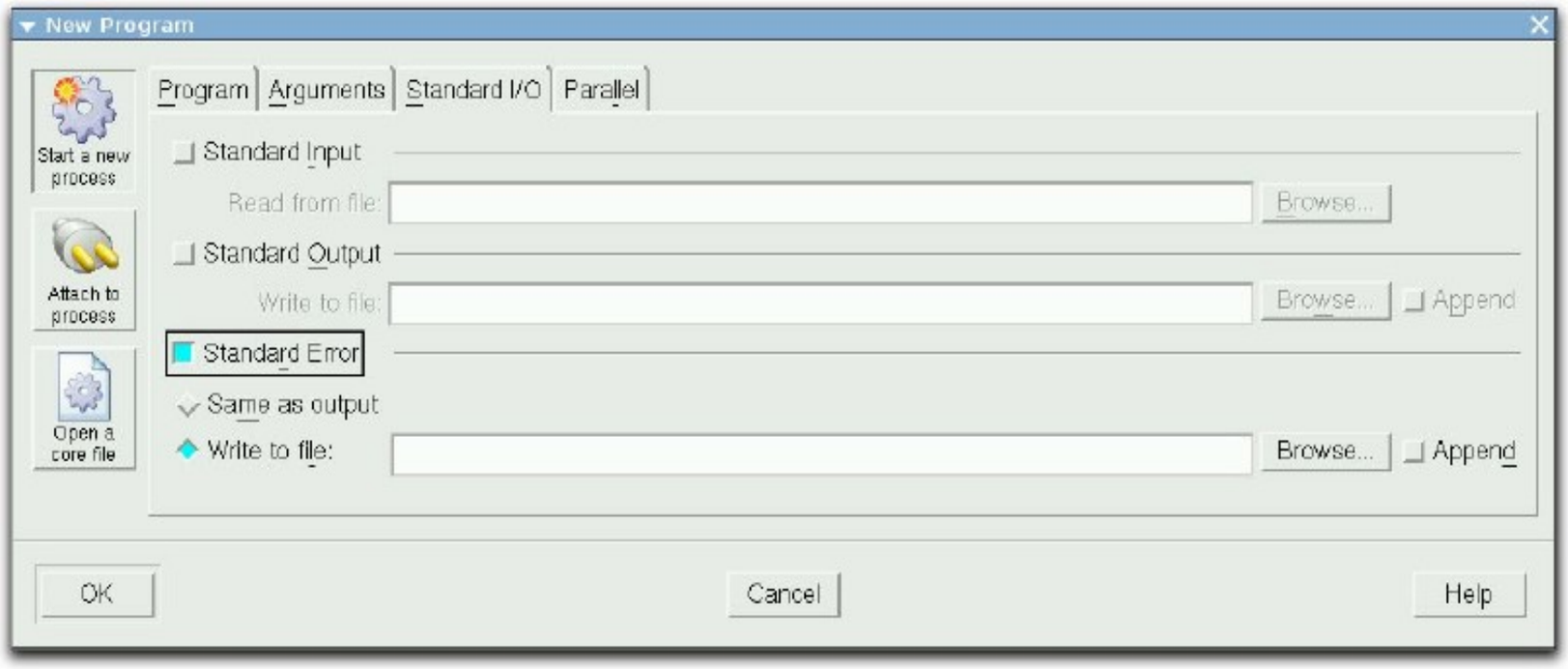
The New Program Screen lets you:

Start a New Process

Attach to an Existing Process

Open a Core File

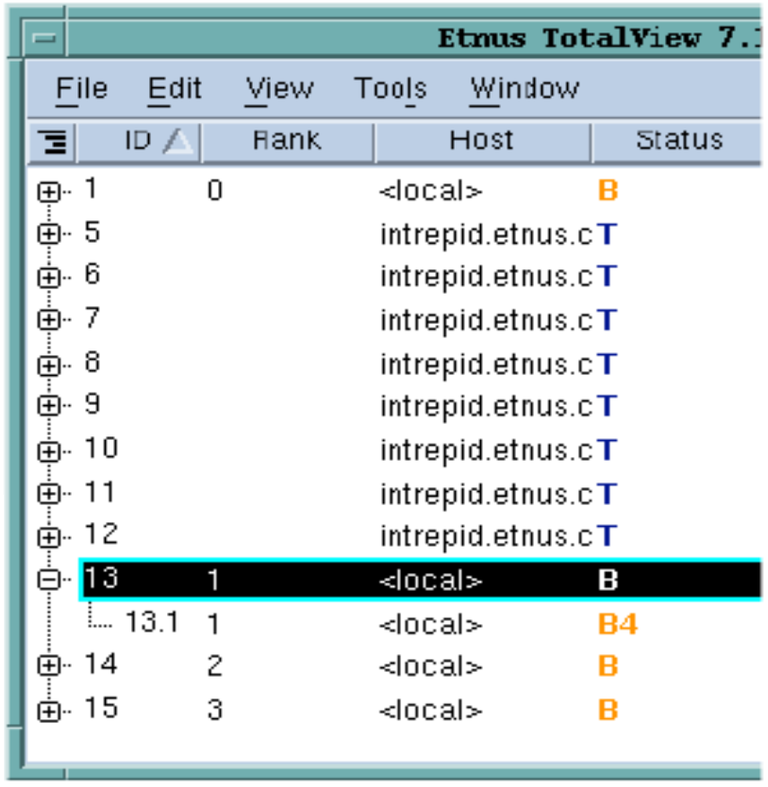




TotalView's Basic Look and Feel

TVD contains two kinds of windows:

- *Root Window (Control)*
  * States of processes
  * Processes and thread Status
  * Instant navigation access.



➤Status Info

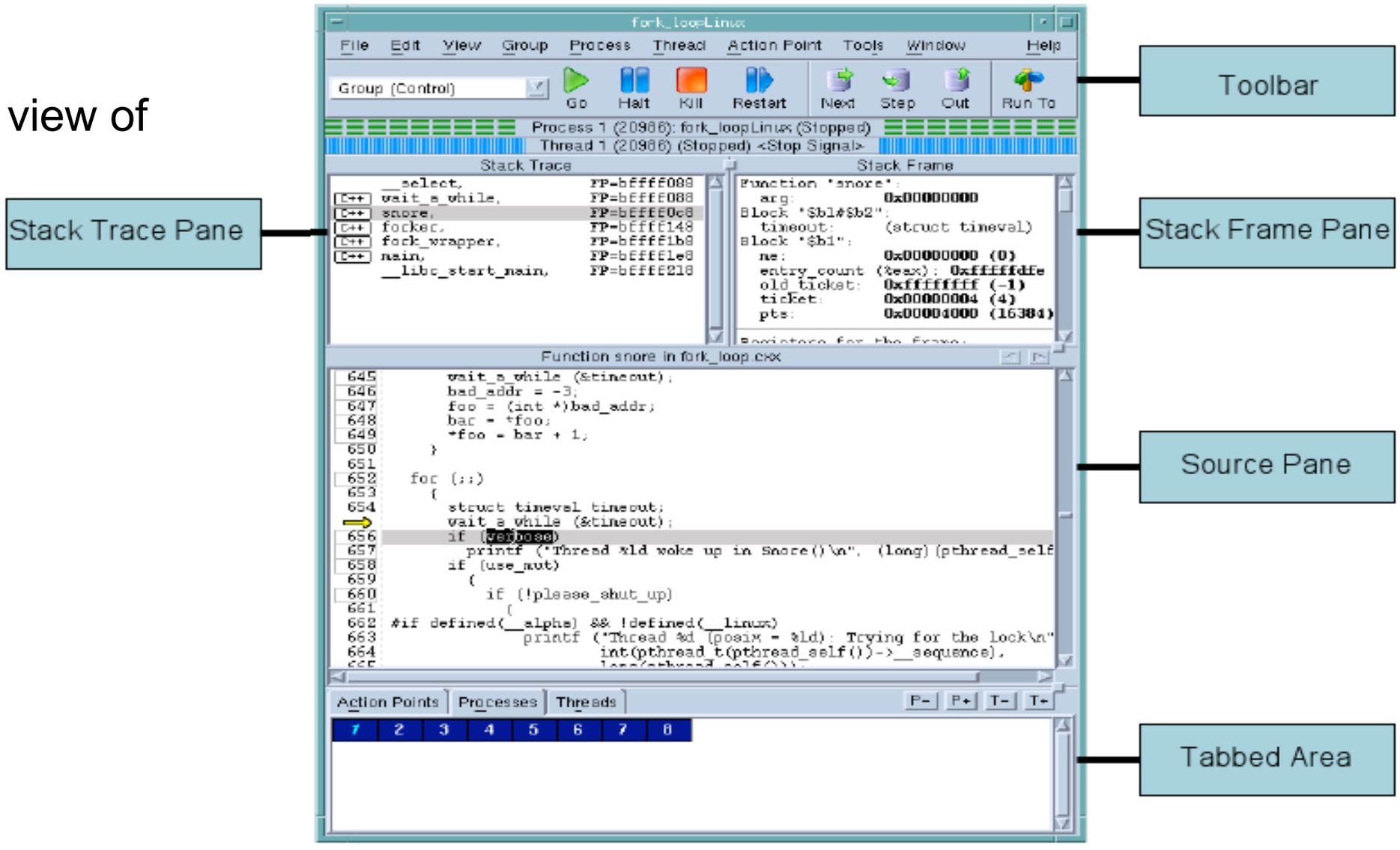
- T = stopped
- B = Breakpoint
- E = Error
- W = Watchpoint
- R = Running
- M = Mixed
- H = Held

State Code	Description
B	Stopped at a breakpoint
E	Stopped because of an error
H	In a Hold state
K	Thread is executing within the kernel
M	Mixed - some threads in a process are running and some not
R	Running
T	Thread is stopped
W	At a watchpoint

TotalView's Basic Look and Feel

TVD contains two kinds of windows:

- *Process Window*:
  Provides detailed view of one process



Basic usage

Controlling execution

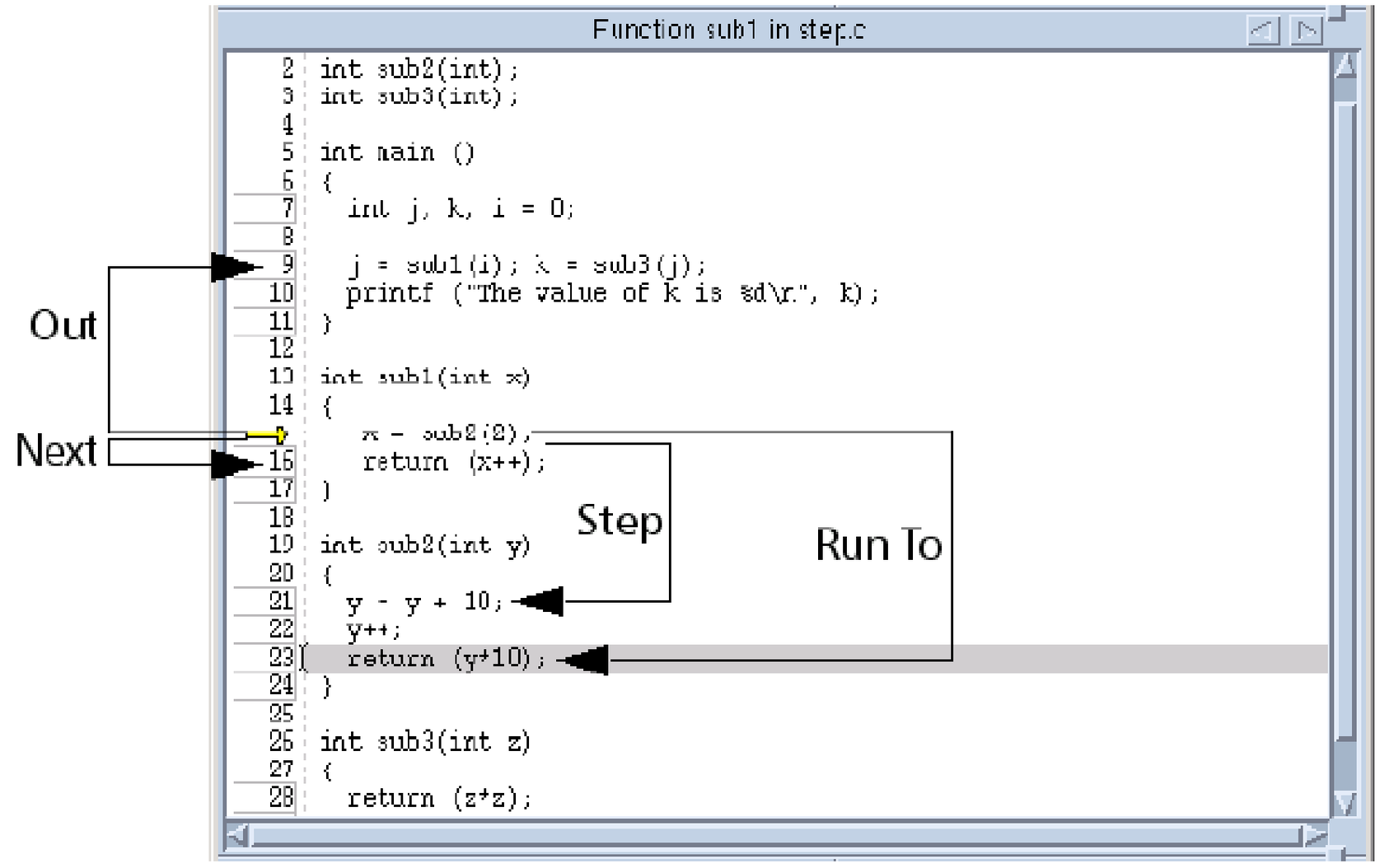
Command	Description
Go	Start/resume execution
Halt	Stop execution
Kill	Terminate the job
Next	Run to next source line or instruction. If the next line/instruction calls a function, the entire function will be executed and control will return to the next source line or instruction (the function is "stepped over").
Step	Run to next source line or instruction. If the next line/instruction calls a function, the function will be "stepped into". Execution will stop within the function.
Out	Execute to the completion of a function. Returns to the instruction after the one which called the function.
Run To	Allows you to arbitrarily click on any source line and then run to that point
Next Instruction	Similar to Next, but applies only to machine instructions
Step Instruction	Similar to Step, but applies only to machine instructions
Hold/Release	Hold ignores other commands to resume execution Release allows other run commands to have effect
Restart	Restarts a running program, or one that has stopped without exiting
Set PC	Sets the Program Counter to a desired source line, machine instruction, or absolute address

Basic usage

Controlling execution

Based on

PC location



Basic usage

"Diving" : In TVD this concept is widely used to refer the way user navigates through the application in a debugging session in order to:

- Obtain more information
- Refocus the process window
- Open variables
- ...

You can "dive" by:

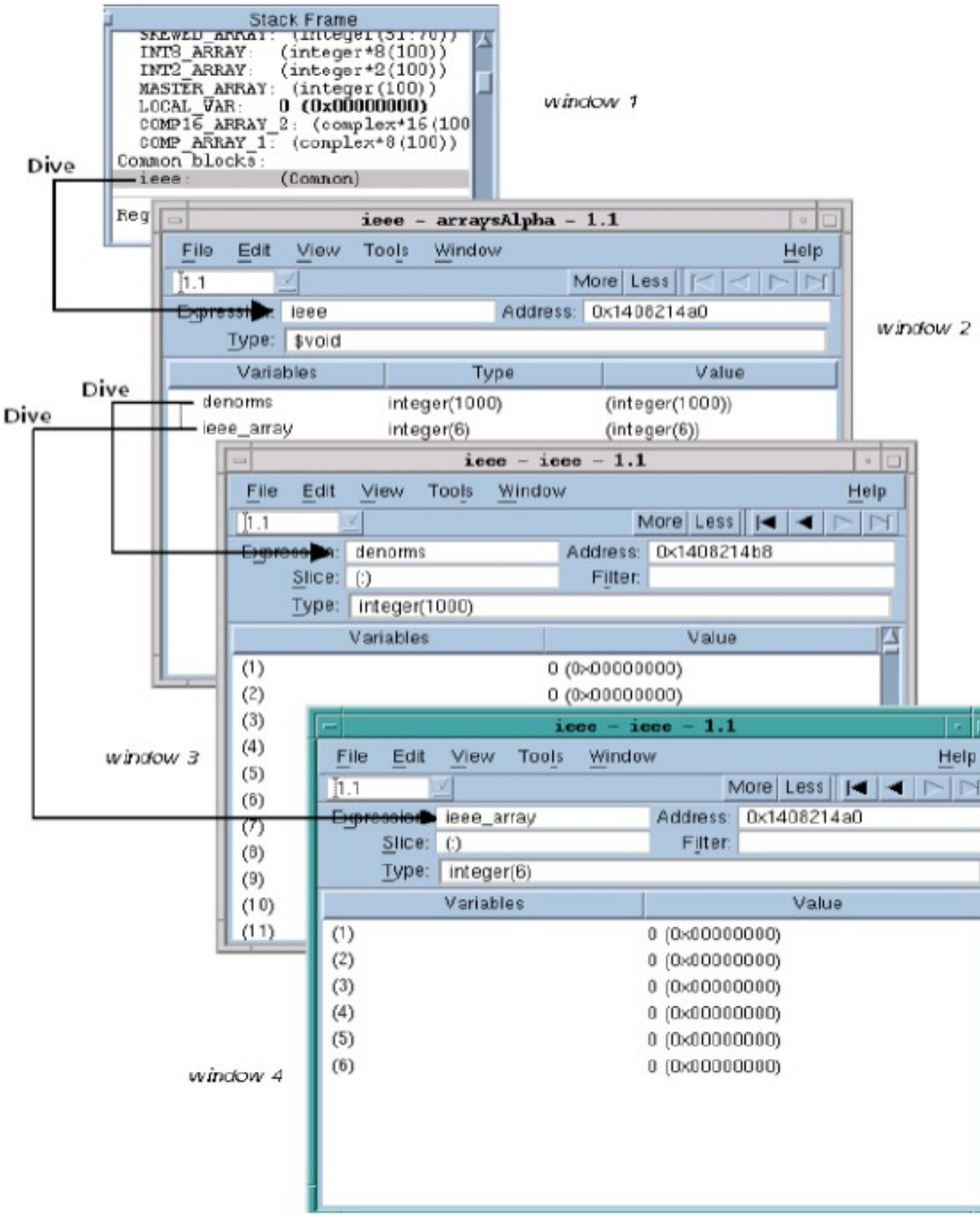
- Double-clicking the left mouse button
- Selecting "Dive" in the context menu

You can dive on:

- Variables names to open a variable window (viewing data)
- Function names to open the source
- Process and threads in the root window to open a process window

Basic usage

"Diving" example over a variable in the common block in the stack frame



Basic usage

Viewing Data:

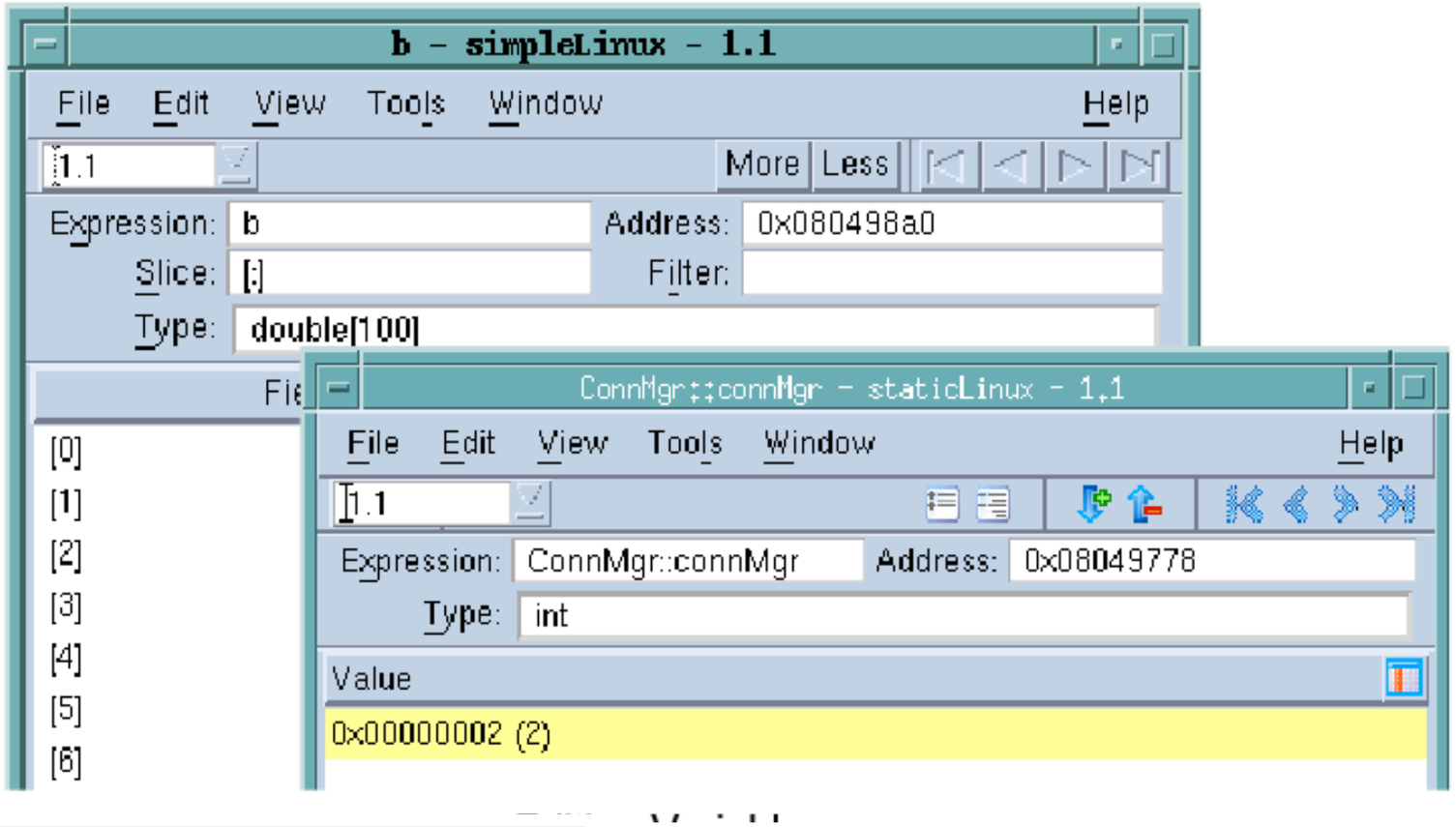
- Diving on a variable opens the Variable Window
  - Contents are updated automatically
  - Changed values are highlighted
  - “Last value” column
  - Clicking on the variable let the user to edit it:
    - Editing values changes the memory of the program
    - “Enter” to commit changes
    - “Esc” to cancel changes

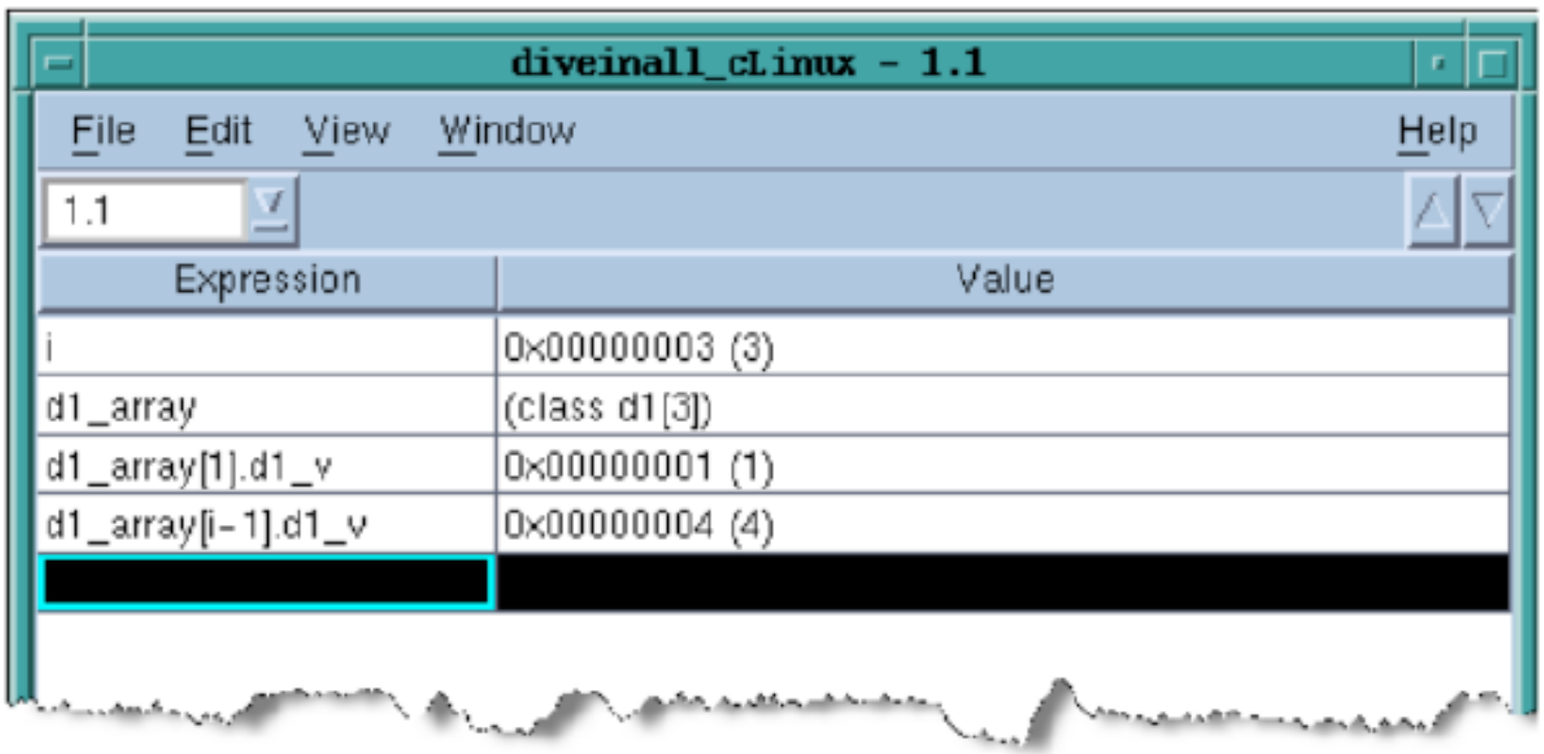
- Using the Expression List Window
  - Variables can be added using right click on the variable
  - Adding expression directly in the window

Introduction to TotalView

Basic usage

Viewing Data:





Basic usage

Viewing Data: Viewing arrays

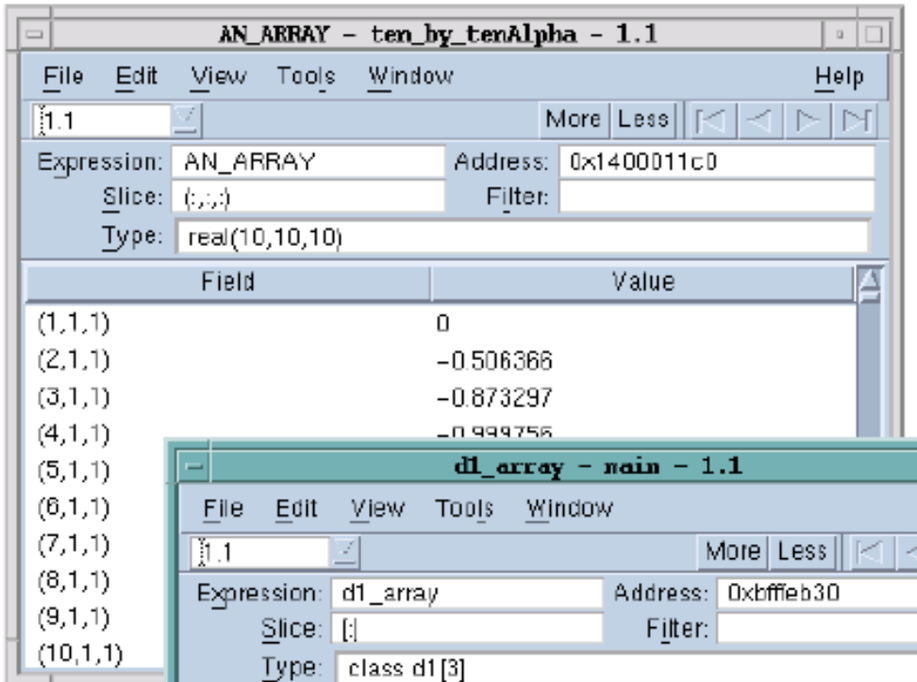
For array data, TotalView provides several additional features:

➔Displaying array slices

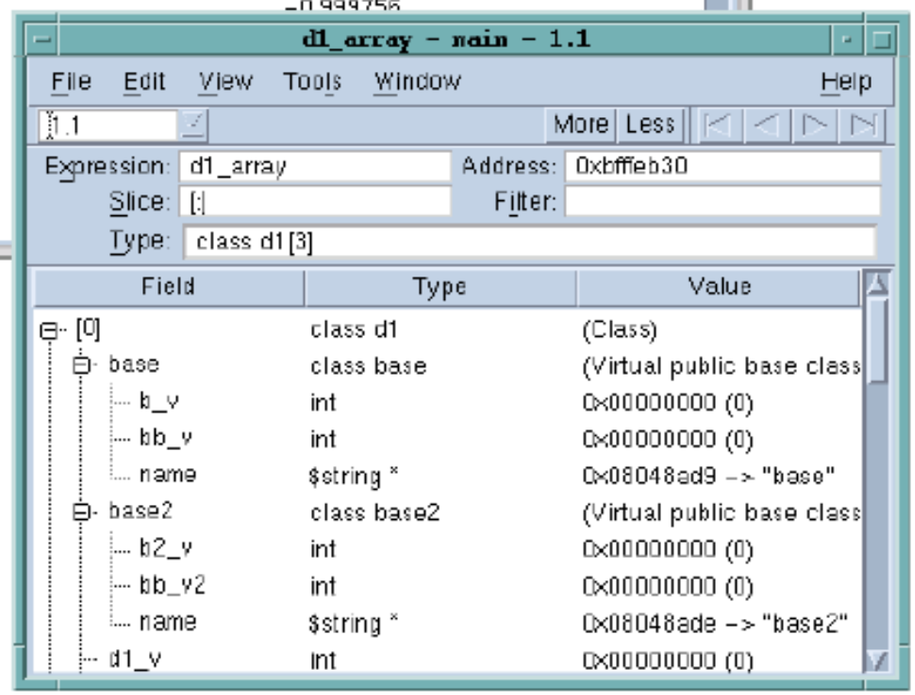
➔Data filtering

➔Data Sorting

➔Array statistics



Data Arrays



Structure Arrays

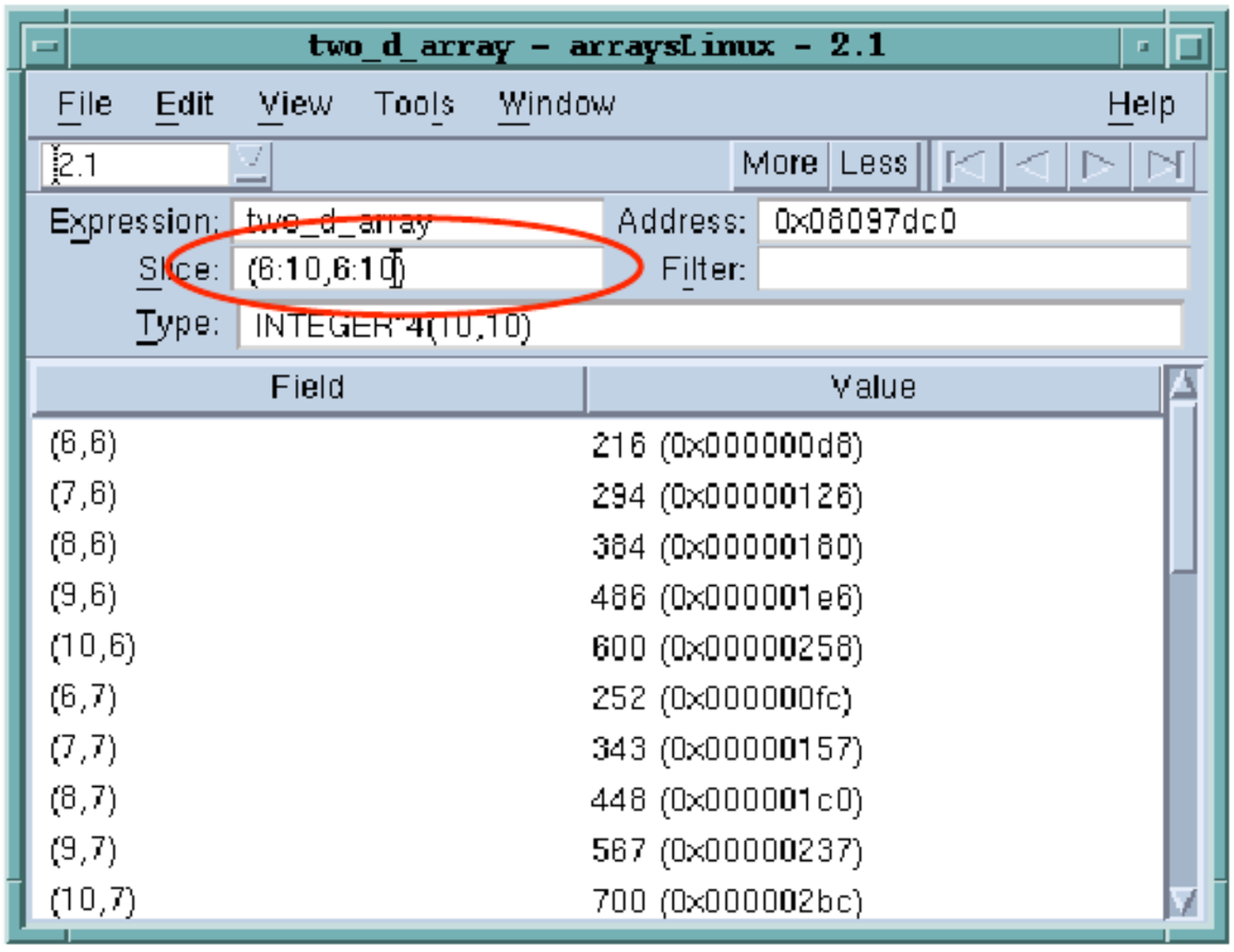
Basic usage

Viewing Data: Slicing arrays

Used to display subsections of an array. Particularly useful if only a small section of a large array is of interest.

lower_bound:upper_bound:stride

Fortran Slice: (1:5, 3:8)
C/C++ Slice: [::2][1:20]



Slice notation is `[start:end:stride]`

Basic usage

Viewing Data: Filtering arrays

Arrays containing data types of character, integer or floating point can be filtered to display only desired data.

Filtering can be:

- By arithmetic comparison
- For IEEE values
- By a range of values
- Within an expression

See the TotalView documentation for additional examples, syntax options and other important information.

Examples:

Fortran

Filter: .gt. 250
Filter: .eq. $nan
Filter: 7:512

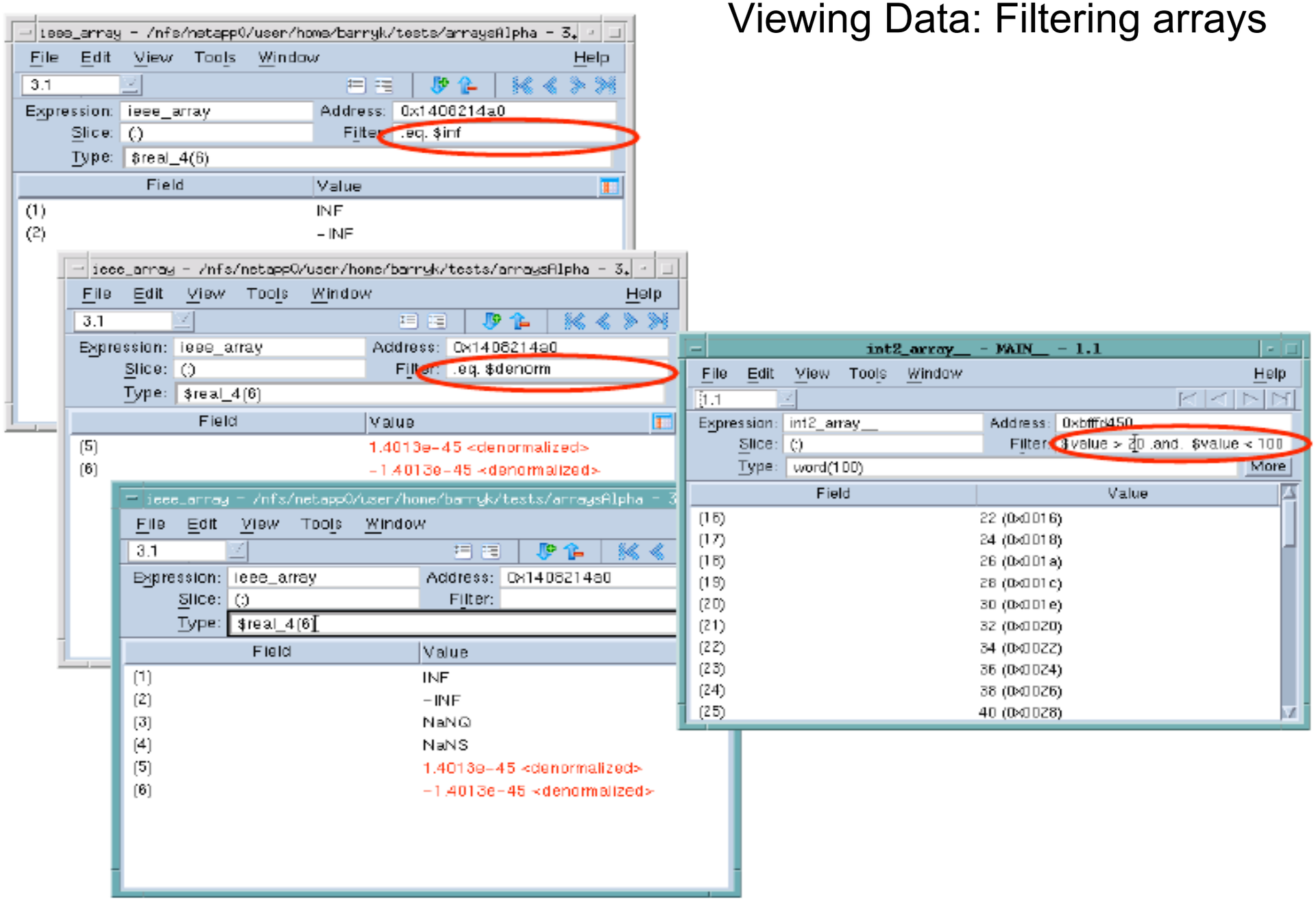
C/C++

Filter: >= 100
Filter: != $inf
Filter: 128:<1024

Introduction to TotalView

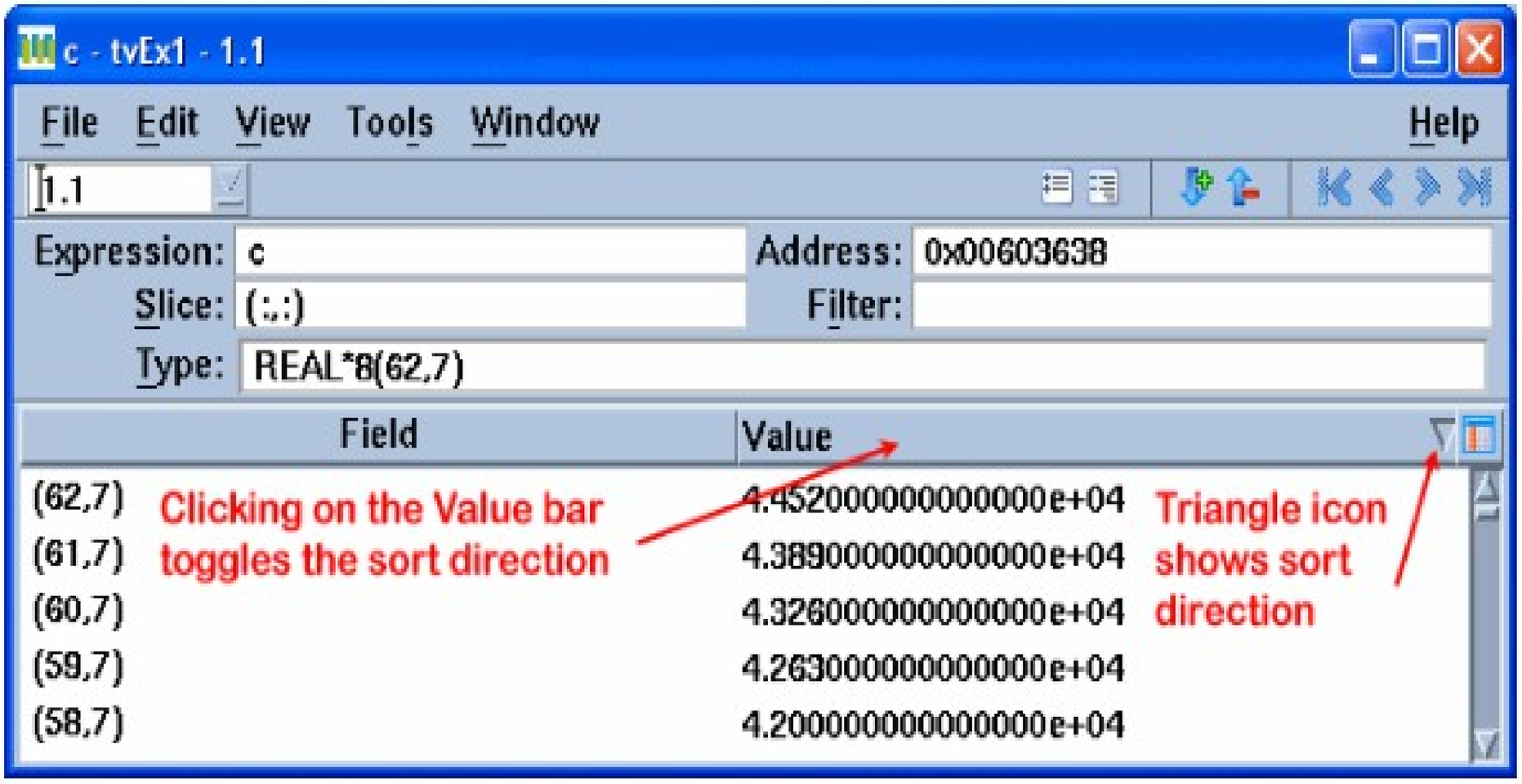
Basic usage

Viewing Data: Filtering arrays



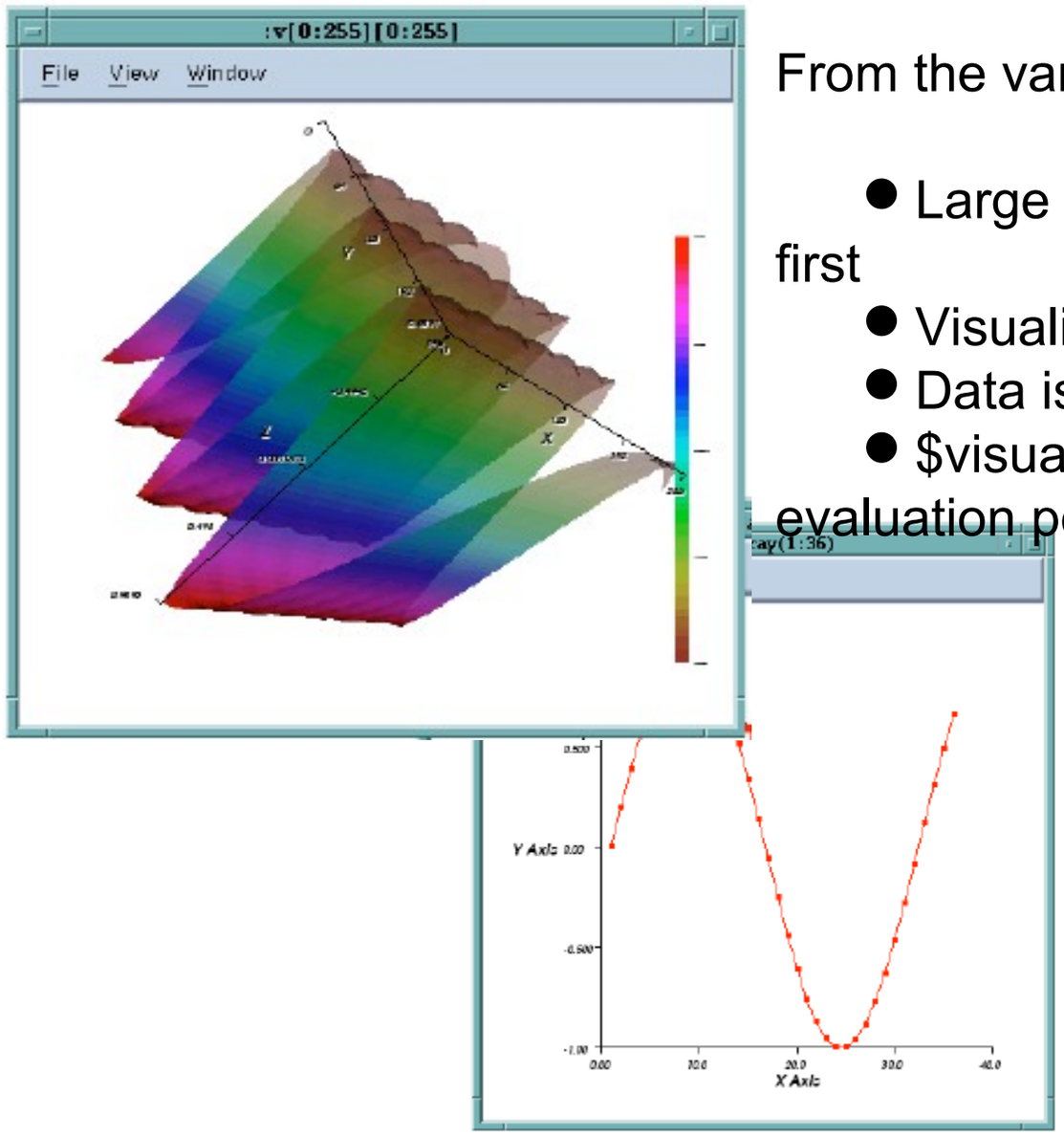
Basic usage

Sorting Array Data



Basic usage

Visualizing Array Data



From the variable window click on Tools → Visualize

- Large arrays can be sliced down to a reasonable size first
- Visualize allows to spin, zoom, etc …
- Data is not updated. You must revisualize
- $visualize() is a directive and can be used in evaluation ponts

Basic usage

Viewing Data: Laminating arrays (view across processes)

Totalview allows you to look at the values of a variable in all MPI processes

- Righ Click on the variable
- Select the variable window → view across

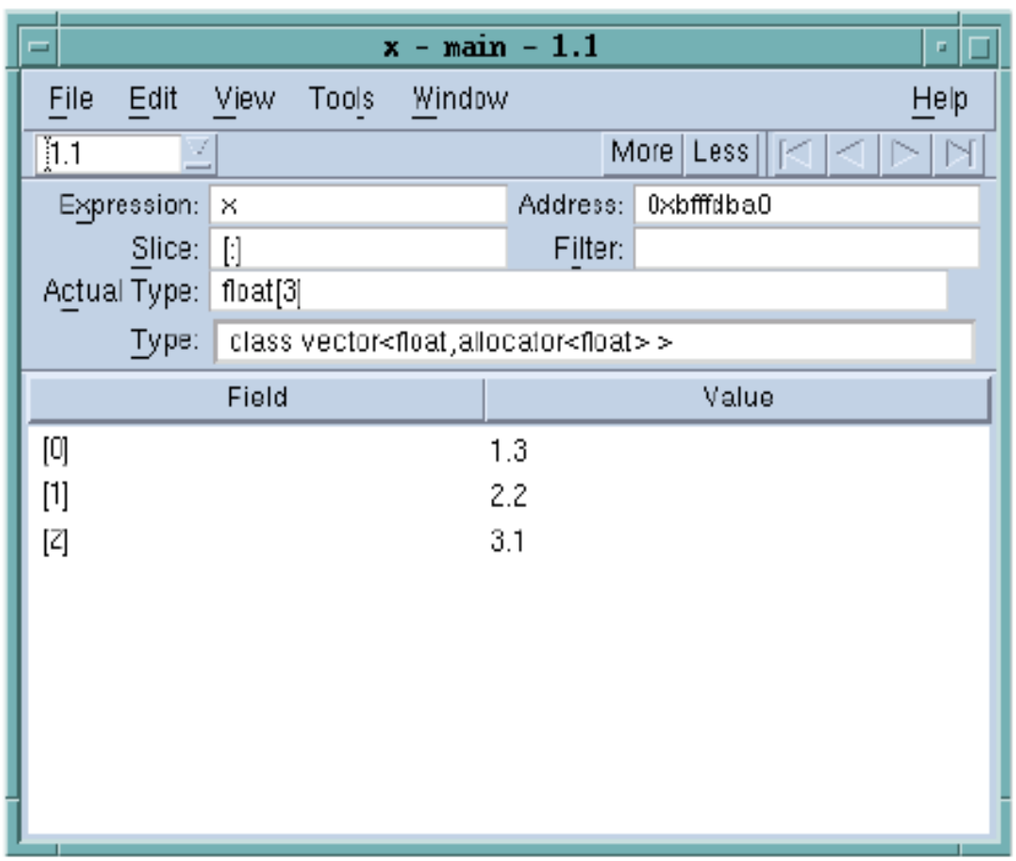
You can Filter, visualize, explore distributed arrays....

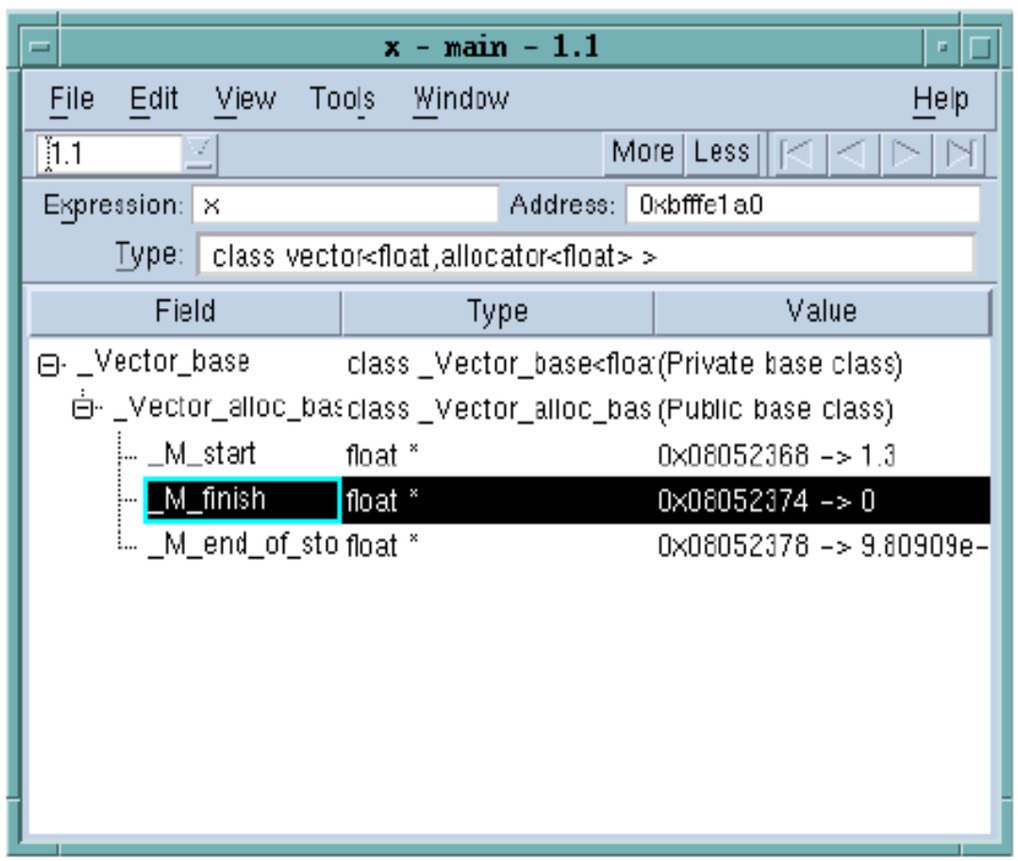
Basic usage

Viewing Data: Viewing STL

TVD transform templates into understandable information:

–STLView supports std::vector, std::list, std::map, std::string

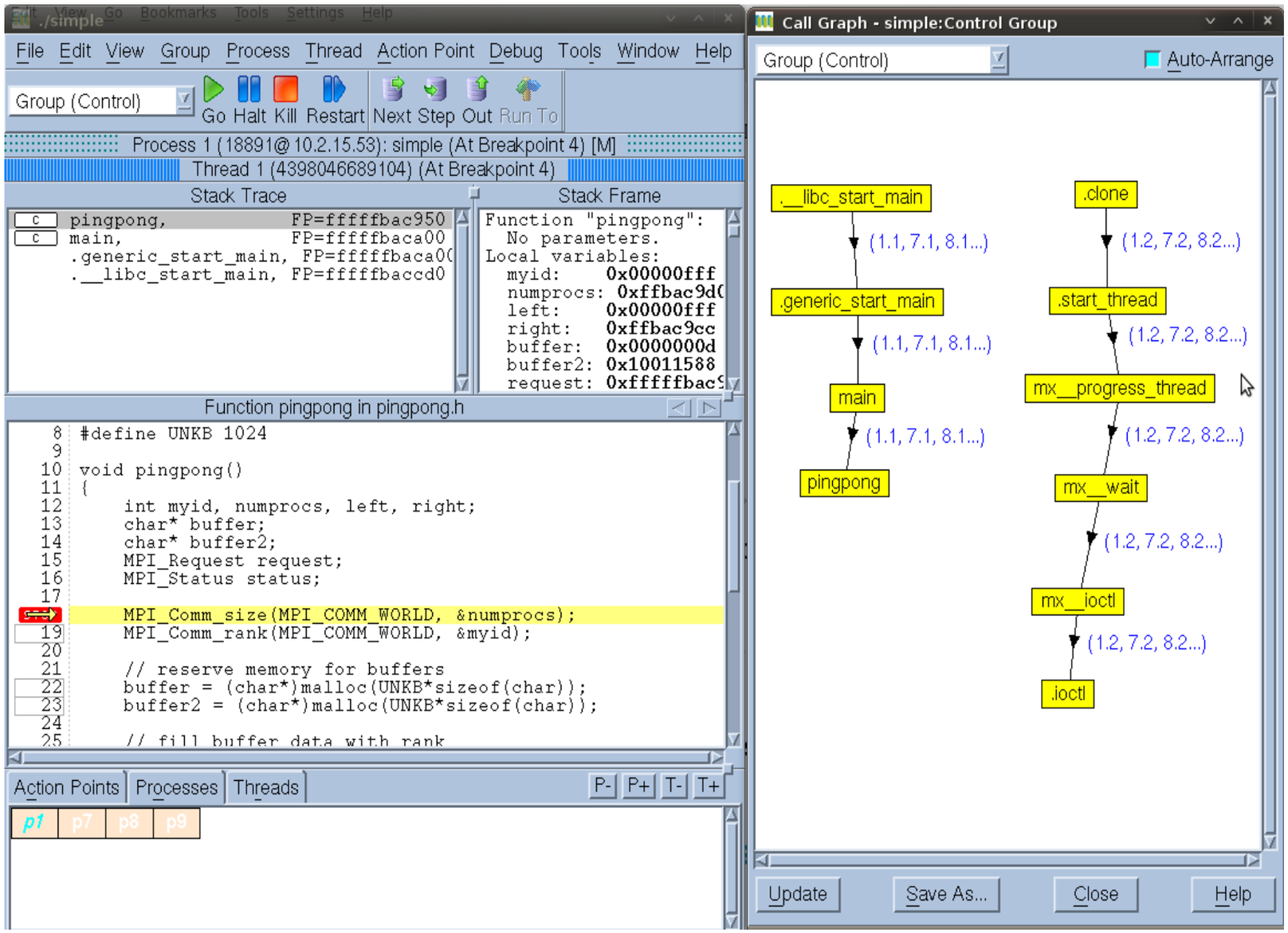




Basic usage

Call graph: Allows a quick view of the program state

- Functions are nodes
- Calls are edges
  Labels are MPI rank



Basic usage

MPI Messages queue and graph:

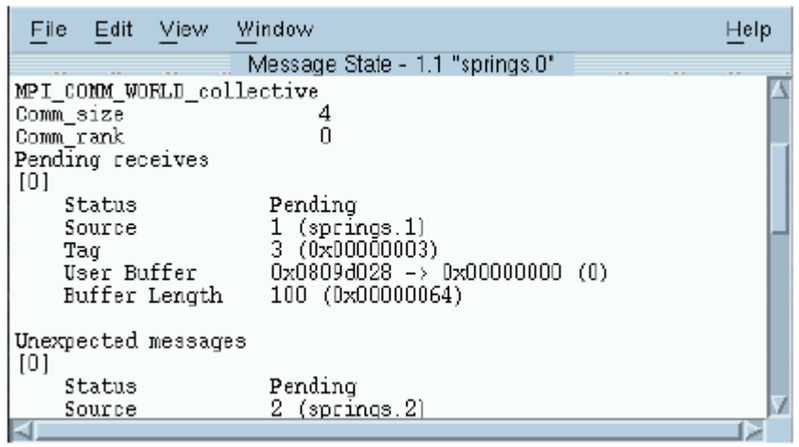
- Provides information from the MPI layer

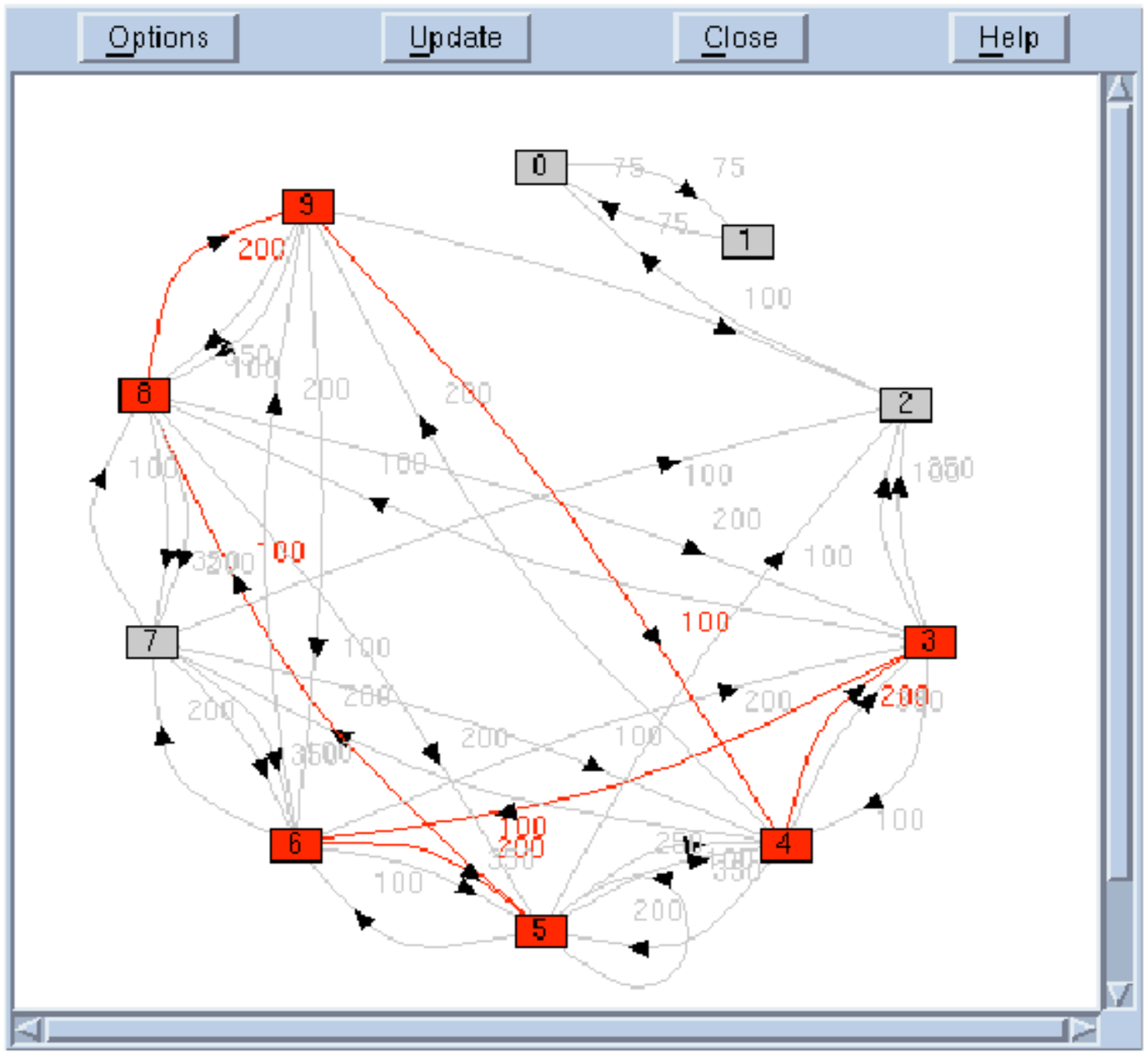
→ pending messages
→ unexpected messages

- Messages can be filtered by tags, MPI Communicators.
- Useful in deadlock situations and load balancing studies.
- May be to be enabled in the MPI Library
  - --enable debug

Introduction to TotalView

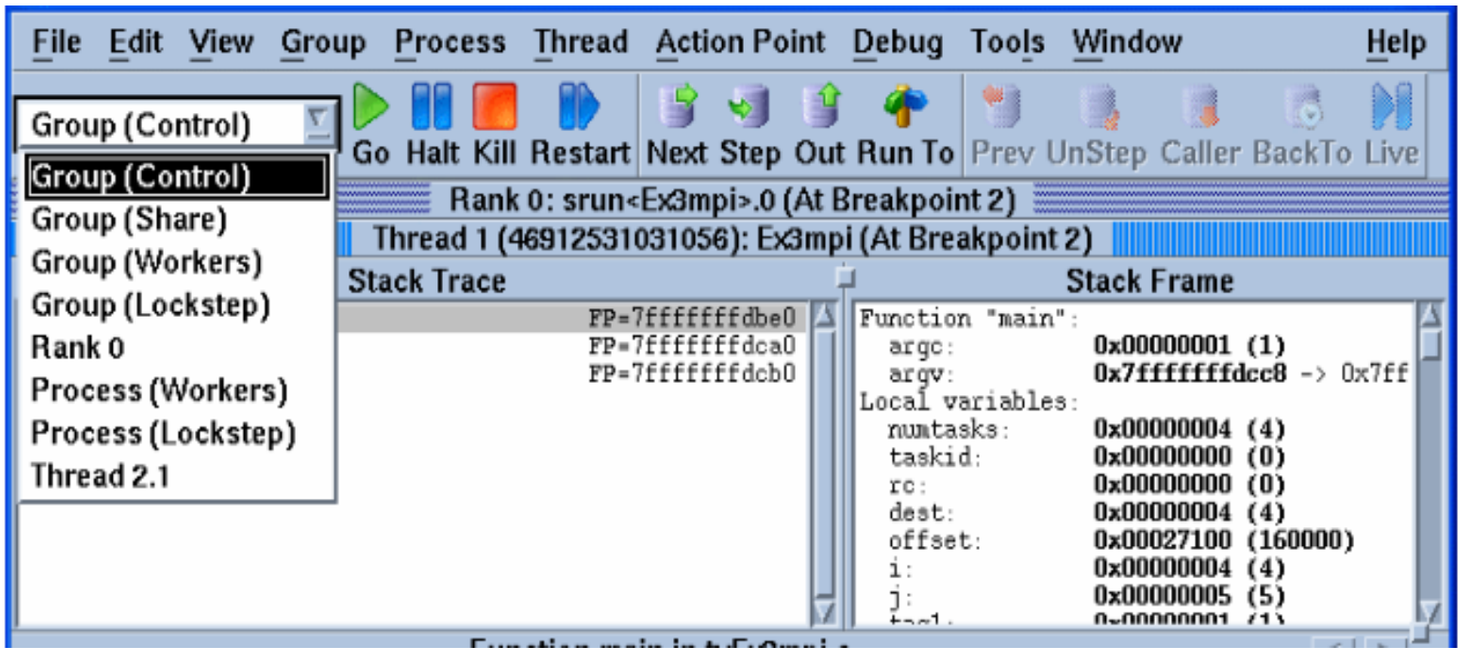
Basic usage





Basic usage

Working with groups



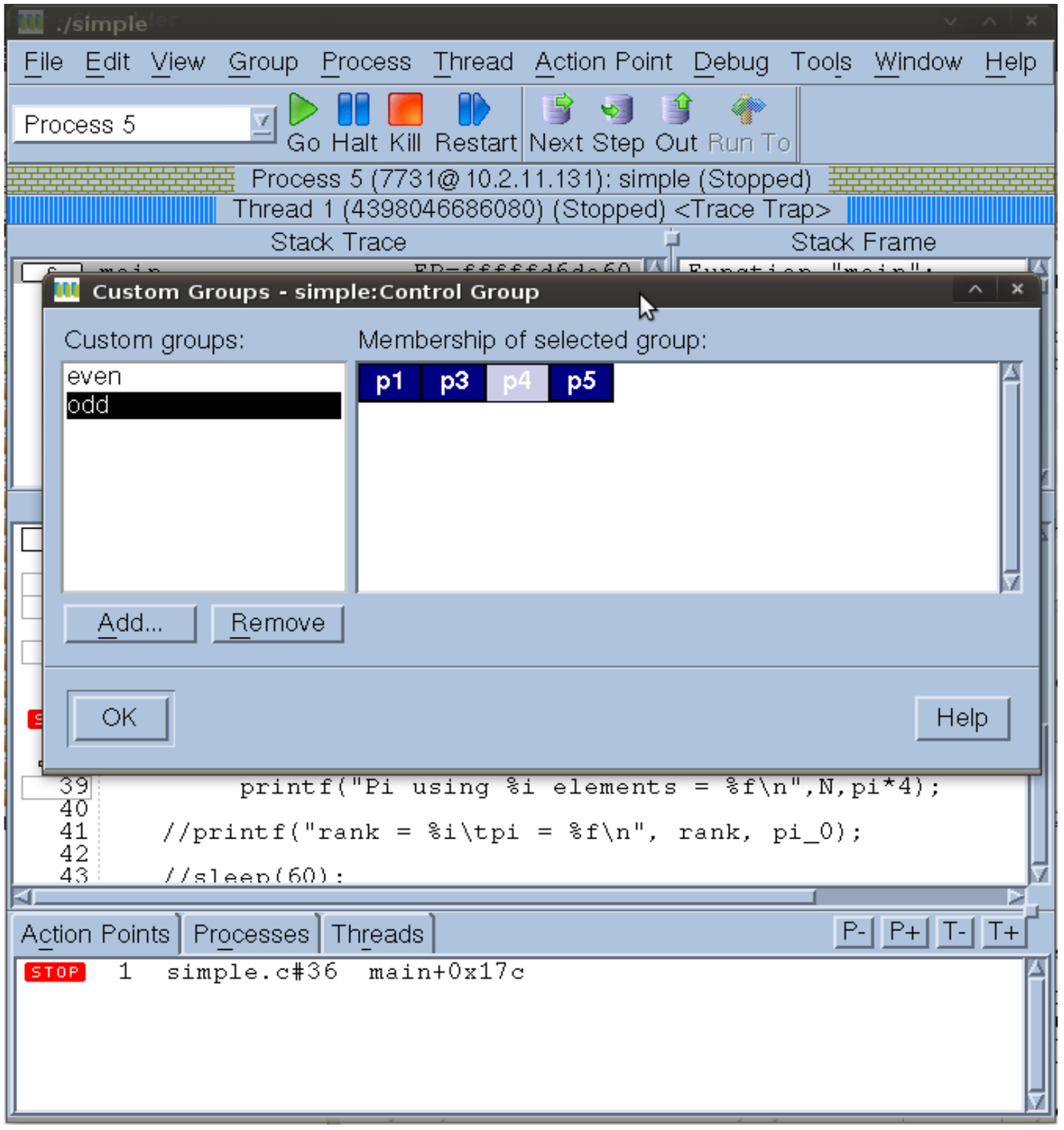
P/T Selection	What is affected by any execution Command
Group (Control)	Default. All processes and their threads.
Group (Share)	All processes and their threads that are in the same share group as the POI (process-of-interest)
Group (Workers)	All threads that are executing user code
Group (Lockstep)	All user threads that are stopped at the same PC
Rank 0	Only the POI and its threads. In the above example, the POI happens to have an MPI rank of 0
Process (Workers)	User threads in the POI
Process (Lockstep)	User threads stopped at the same PC in the POI
Thread 2.1	Only the TOI (thread-of-interest). In the above example, the TOI happens to be 2.1

Introduction to TotalView

Basic usage

Creating custom groups

Group → Custom Groups ...



Basic usage

Action Points

TotalView supports four different types of action points:

- ***Breakpoint*** - stops execution of the processes or threads that reach it. Note that breakpoints apply to the entire process - if any thread executing a process reaches a breakpoint, TotalView will stop the entire process.

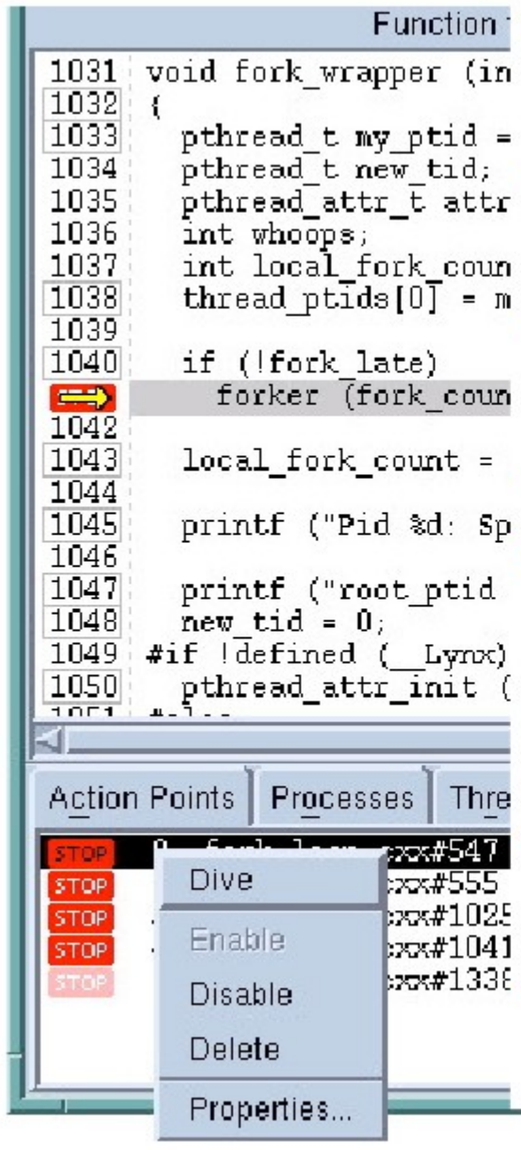
- ***Process Barrier Point*** - holds each process when it reaches the barrier point until all processes in the group have reached the barrier point. Primarily for MPI programs.

- ***Evaluation Point*** - causes a code fragment to execute when it is reached. Enables you to set **"conditional breakpoints"** and perform conditional execution.

- ***Watchpoint*** - enables you to monitor a location in memory and either stop execution or evaluate an expression when the value stored in memory is modified.

Basic usage

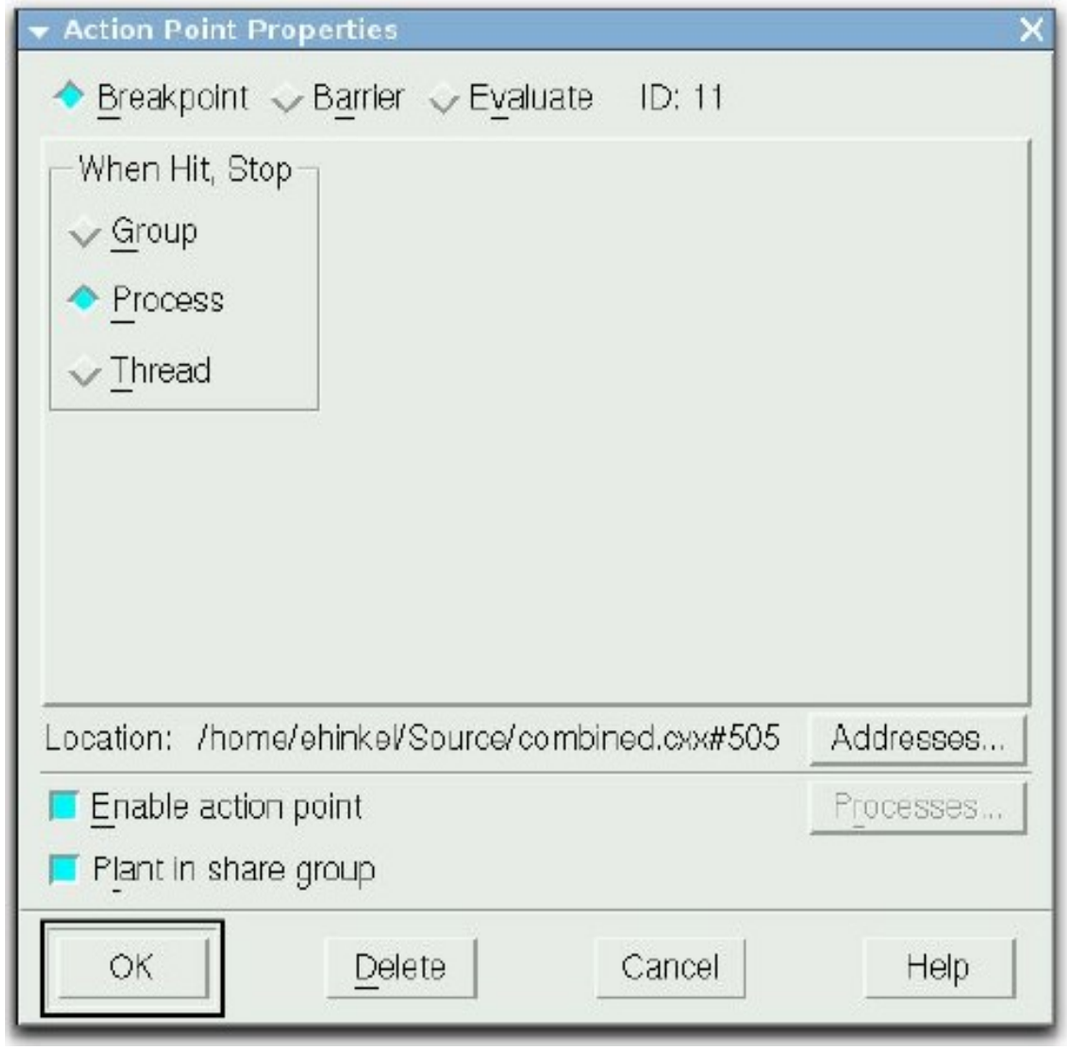
Action Points: managing breakpoints

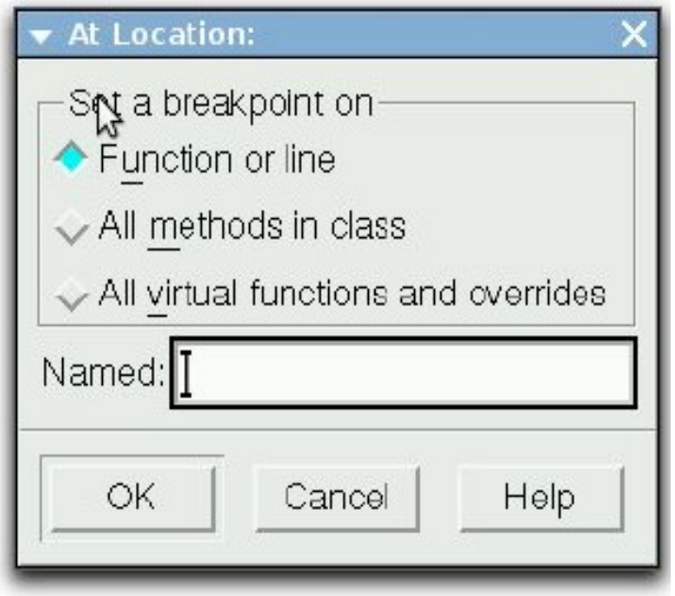


- Setting action points
  - Single-click line number
- Deleting action points
  - Single-click action point line
- Disabling action points
  - Single-click in Action Points Tab Pane
- Optional contextual menu access for all functions
- Action Points Tab
  - Lists all action points
  - Dive on an action point to focus it in source pane
- Action point properties
  - In Context menu
- Saving all action points
  - Action Point > Save All

Basic usage

Action Points: Setting



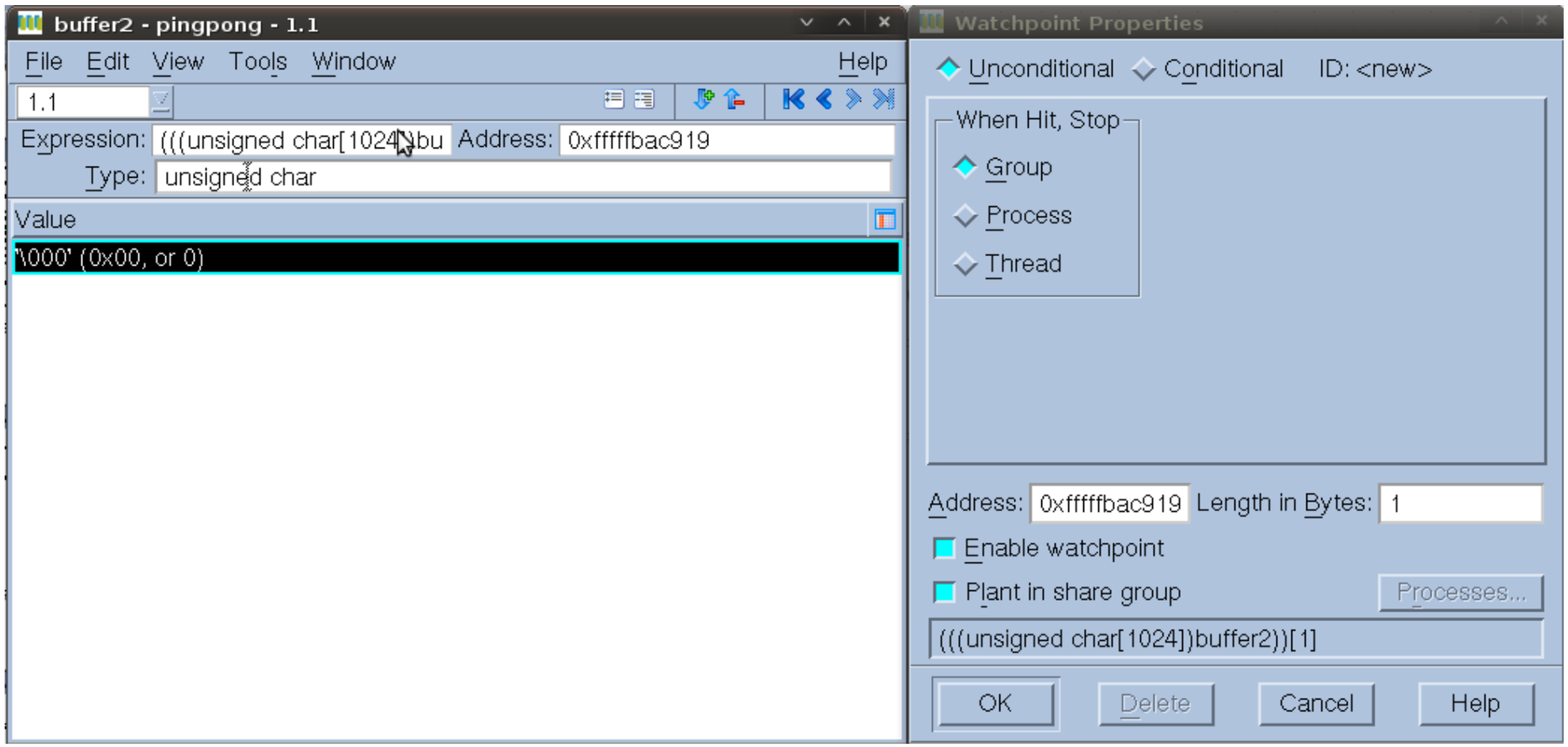


Basic usage

Action Points: Watchpoint

When the contents of a watched variable change, TVD stops the program

Watchpoints are set from the Variable Window: **Tools** → **Watchpoint**



Watchpoints are NOT set on a variable but on a memory region as well
So, user must be aware of the scope of the variable

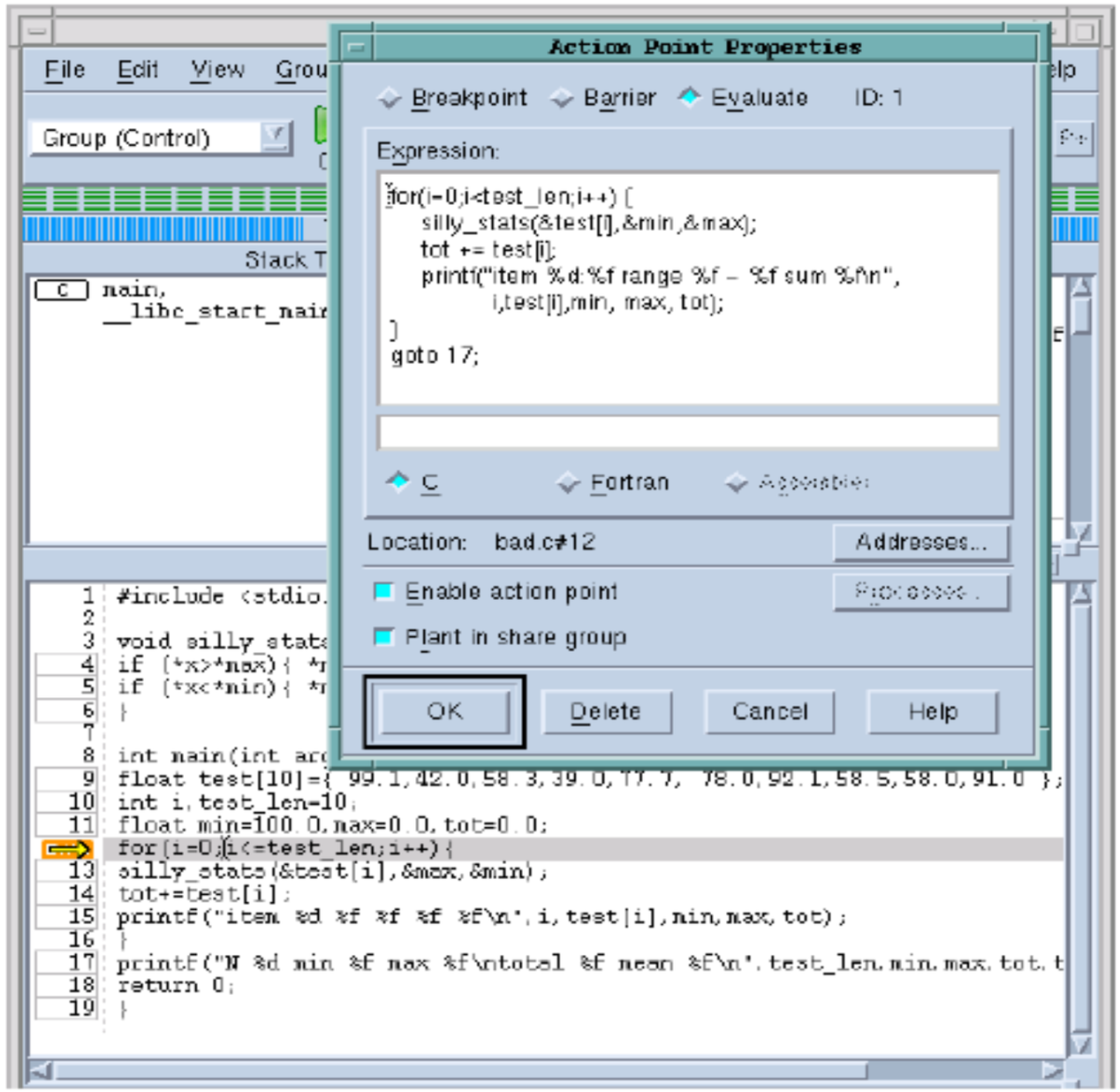
Basic usage

Action Points: *Evaluation and conditional breakpoints*

It is a cool feature that allows:

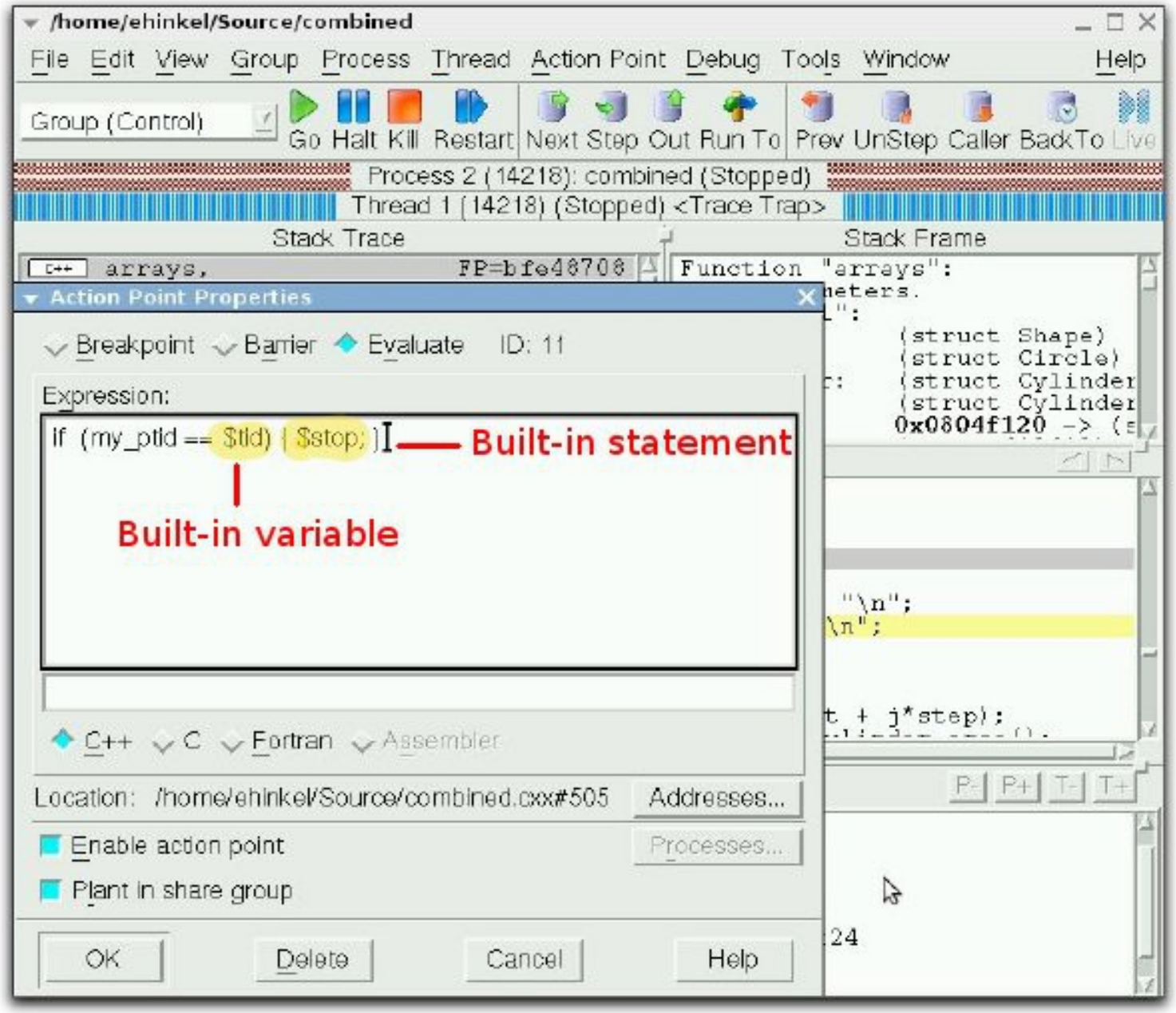
- Testing small source code patching
- Call functions
- Set variables
- Test conditions
- Use program variables

Can't be used with Replay Engine



Basic usage

Action Points:
Evaluation
And
conditional breakpoints



Built-in variables and statements

Built-in Variable	Description
$clid	Returns the cluster ID.
$duid	Returns the TotalView-assigned Debugger Unique ID (DUID).
$newval	Returns the value just assigned to a watched memory location (watchpoints only).
$nid	Returns the node ID.
$oldval	Returns the value that existed in a watched memory location before a new value modified it (watchpoints only).
$pid	Returns the process ID.
$processduid	Returns the DUID of the process.
$systid	Returns the system-assigned thread ID. When referenced from a process, generates an error.
$tid	Returns the TotalView-assigned thread ID. When referenced from a process, generates an error.

Built-in variables and statements

TotalView provides a set of built-in statements that you can use when writing code fragments. The statements are available in all languages, and are shown in the table below.

Built-In Statement	Description
$count expression	Sets a process-level countdown breakpoint.
$countprocess expression	When any thread in a process executes this statement for the number of times specified by expression, the process stops. The other processes in the program group continue to execute.
$countall expression	Sets a program-group-level countdown breakpoint. All processes in the program group stop when any process in the group executes this statement for the number of times specified by expression.
$countthread expression	Sets a thread-level countdown breakpoint. When any thread in a process executes this statement for the number of times specified by expression 1 , it stops. The other threads in the process continue to execute. If the target system does not support asynchronous stop, this executes as a $countprocess.
$hold $holdprocess	Holds the current process. If all other processes in the group are already held in breakpoint state at this eval point, then all will be released. If other processes in the group are running, they continue to run.
$holdstopall $holdprocessstopall	Exactly like $hold, except any processes in the group which are running are stopped. Note that the other processes in the group are not automatically held by this call -- they are just stopped.
$holdthread	Freezes the current thread leaving other threads running.
$holdthreadstop $holdthreadstopprocess	Exactly like $holdthread except it stops the process. The other processes in the group are left running.
$holdthreadstopall	Exactly like $holdthreadstop except it stops the entire group.
$stop $stopprocess	Sets a process-level breakpoint. The process that executes this statement stops, but other processes in the program group continue to execute.
$stopall	Sets a program-group-level breakpoint. All processes in the program group stop when any thread or process in the group executes this statement.
$stopthread	Sets a thread-level breakpoint. The thread that executes this statement stops, but all other threads in the process continue to execute. If the target system does not support asynchronous stop, this executes as a $stopprocess.
$visualize(expression[,slice])	Visualizes the data specified by expression and modified by the optional slice. Expression and slice must be written in the syntax of the code fragment's language. The expression can be any valid expression that yields a data-set (after modification by slice) that can be visualized. The slice is a quoted string containing a slice expression. For more information on how to use $visualize in an expression, see "Visualizing Data in Expressions" in the TotalView User Guide.

Expression Evaluation and Code Fragments

Code fragments interact with your program, and are evaluated within its runtime context.

They can therefore be used for a variety of purposes, such as:

- Setting conditional breakpoints
- Program patching - branching around code and/or adding new code
- Effecting conditional execution
- Displaying program data
- Modifying program data

TotalView enables you to enter "code fragments" during a debugging session.

Code fragments can include a mixture of:

- C, Fortran or Assembler language code
- TotalView built-in variables ($tid, $pid, $systid ...)
- TotalView built-in statements ($stop, $hold, $stopall ...)

Memory Debugging

Beginning with TotalView version 8.7, the memory debugging functions of TotalView are packaged as a separate, but integrated, client called **MemoryScape**.
Prior to 8.7, the memory functionality was launched from an integrated Memory Debugging Window.

Key features include:

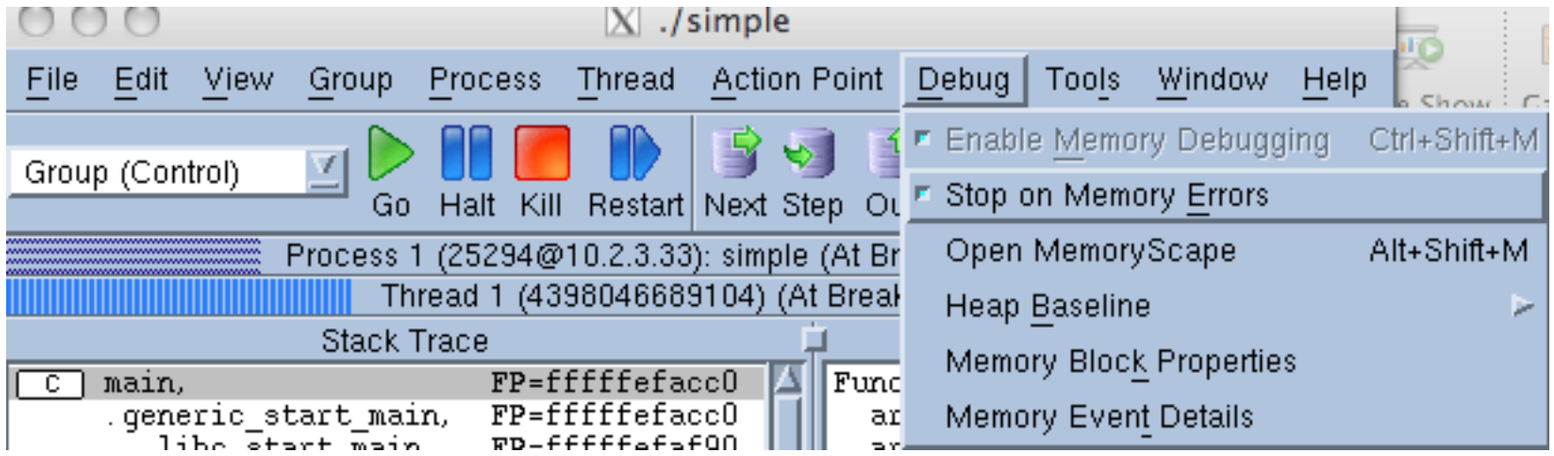
- Memory usage reports
- Leak detection
- Heap status
- Corrupted memory detection
- Dangling pointers

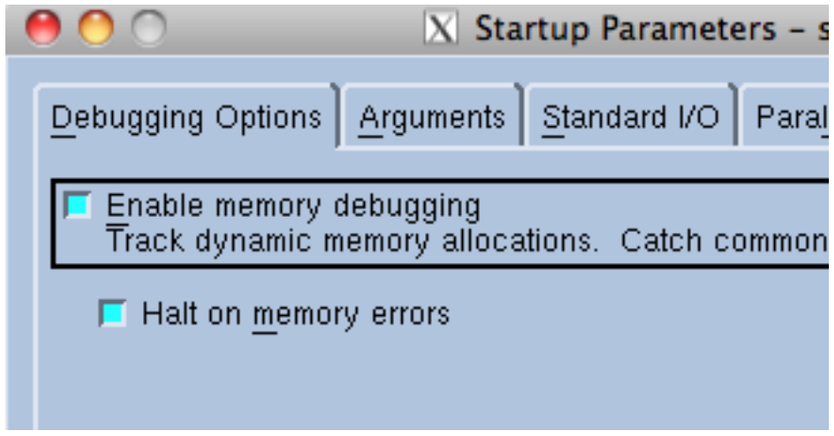
MemoryScape would require a separate tutorial

Memory Debugging

MemoryScape can be launched as a standalone application or from within TotalView

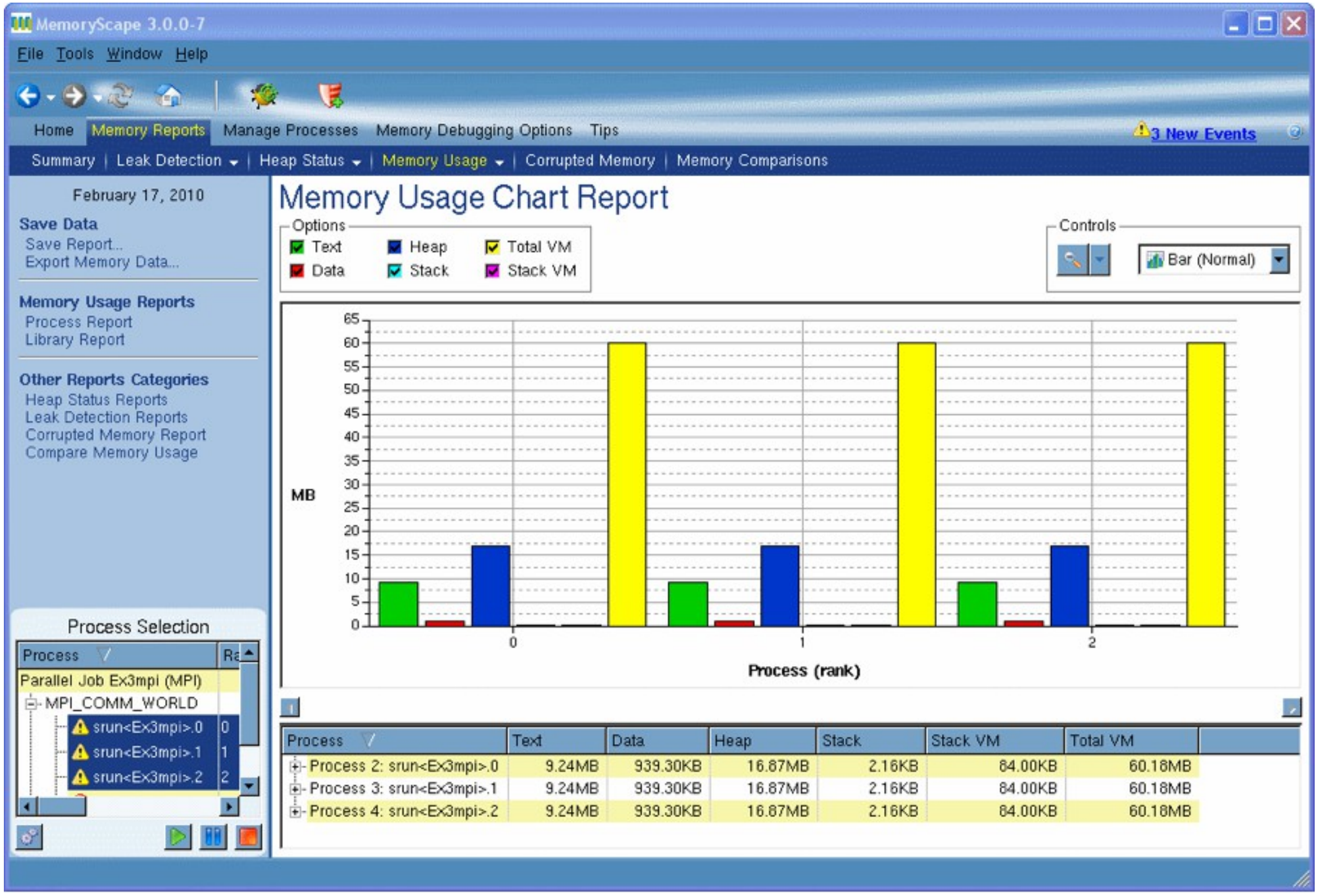
→ It is important to click on the checkbox “Enabling Memory Debugging” when TVD is started.



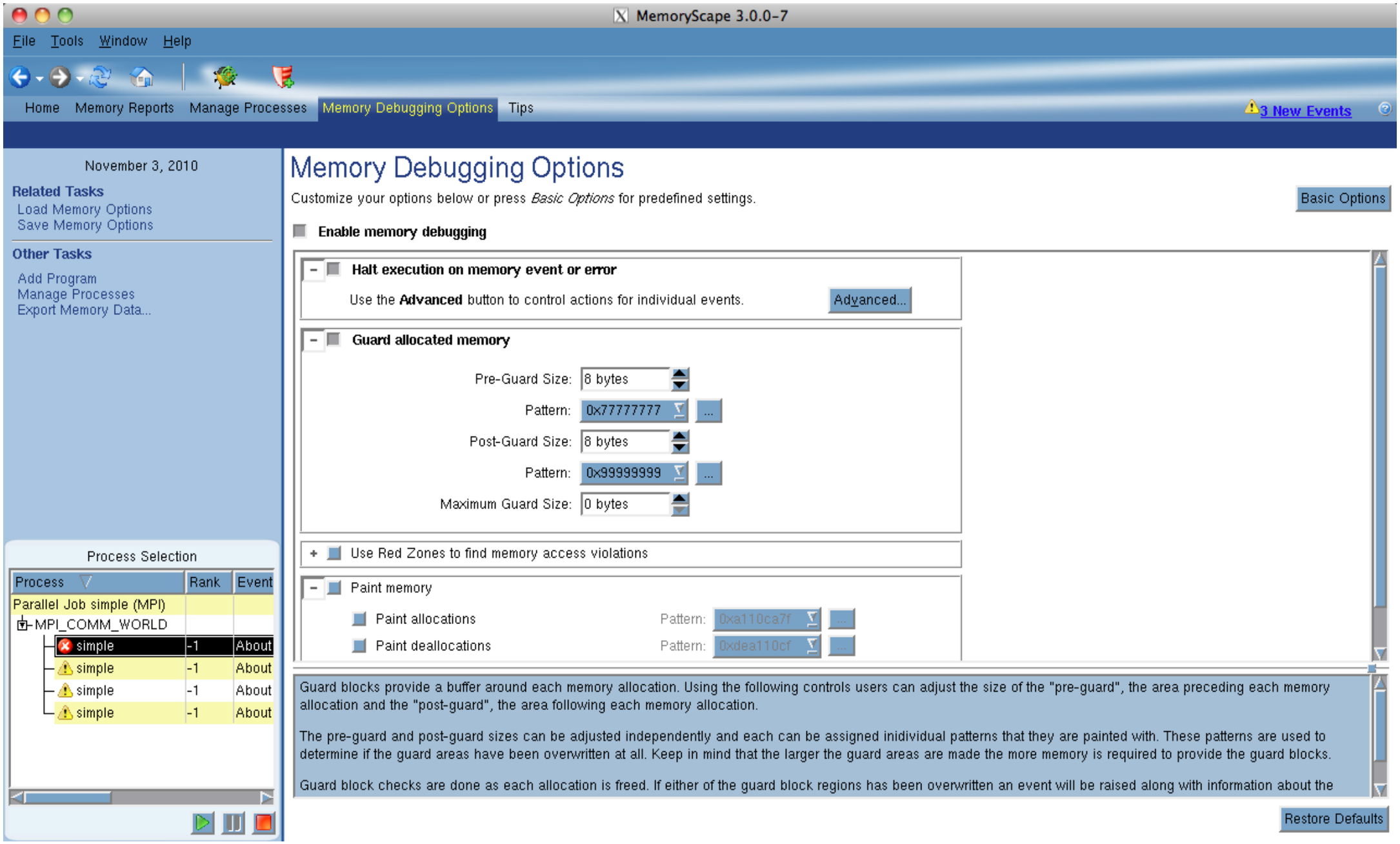


Introduction to TotalView

Memory Debugging

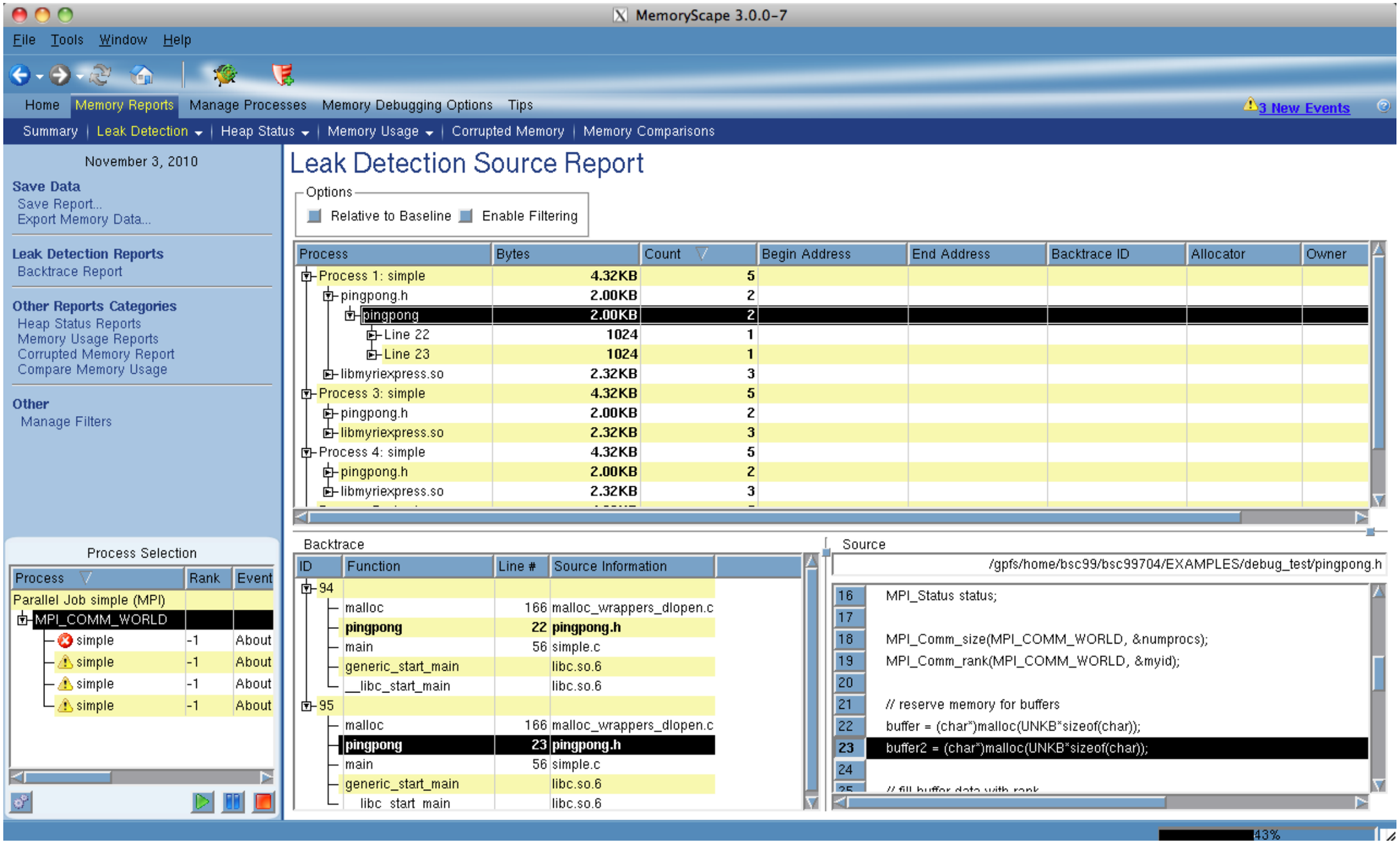


Memory Debugging



Introduction to TotalView

Memory Debugging



Remote Display

This feature is not currently available in MareNostrum but it is worth to mention it here

TotalView Remote Display lets you start and then view TotalView as it exe- cutes on another system.

For example, debugging in MS Windows from home in a PC which is outside a firewall

Some notes on the CLI

One interesting TotalView feature is the CLI (Command Line Interpreter)

- The TotalView Command Line Interpreter (CLI) provides a command line debugger interface
- CLI commands can be integrated into user-written Tcl programs/scripts for "automated" debugging (Advanced)

CLI is useful when:

- a program takes several days to execute
- the program must be run under a batch scheduling system or network conditions that inhibit GUI interaction.
- network traffic between the executing program and the person debugging is not permitted or limits the use of the GUI.

For details see the TotalView documentation located at
www.totalviewtech.com

Hands on

The End

Thanks for your attention

RED ESPAÑOLA DE SUPERCOMPUTACIÓN