



# Troubleshooting complex codes with TotalView (now with a new UI)



# Agenda

---

- Introduction to Rogue Wave
- TotalView Overview
- Debugging is challenging – how does TotalView help?
  - Getting Started
  - Navigating your debugging session
  - Viewing your data
  - Parallel debugging adds complexity
  - Tricky memory issues
  - Batch can be helpful
- New TotalView UI
- Q&A

# Introduction to Rogue Wave

# What we do

---

Rogue Wave helps organizations **simplify** complex software development, **improve** code quality, and **shorten** cycle times



# Company snapshot

---

We are the largest independent provider of cross-platform software development tools and embedded components

<b>Founded:</b> 1989	<b>Headquarters:</b> Louisville, CO	<b>Employees:</b> 350	<b>Offices Worldwide:</b> 11
-------------------------	--	--------------------------	---------------------------------

Our capabilities cover different languages, code bases, and platforms.  
We meet development where – and how – it happens.

# Company timeline

## Technology Timeline

<b>1989</b> Cross-platform commercial math & statistics libraries for C++	<b>1994</b> First commercial database library for C++	<b>2005</b> First enterprise focused open source management and governance tool	<b>2008</b> First graphical reverse debugger for C, C++, and Fortran on Linux	<b>2010</b> First GPU enabled commercial analytics in Fortran	<b>2011</b> First Infiniband Cluster-capable reverse debugger  First cache optimization product to market	<b>2013</b> Noise Reduction patent awarded to OpenLogic	<b>2015</b> PHP 7 released
						<b>2014</b> Klocwork awarded Red Herring Top 100	

1989

2016

<b>1989</b> Rogue Wave established Tools.h++	<b>1996</b> Rogue Wave publicly listed on NASDAQ	<b>2003</b> Rogue Wave acquired by Quovadx	<b>2007</b> Rogue Wave spun out of Quovadx	<b>2009</b> Rogue Wave acquires TotalView and Visual Numerics	<b>2010</b> Rogue Wave acquires Acumem	<b>2012</b> Rogue Wave acquires Visualizations for C++	<b>2013</b> Rogue Wave acquires OpenLogic & Klocwork	<b>2014</b> Rogue Wave acquires JViews, Elixir, Visualizations for .NET	<b>2015</b> Rogue Wave acquires Zend
---	---	---	---	--	---	---	---	--	---

## Corporate Timeline

# Our products



## Tools

**Klocwork** On-the-fly static code analysis for app security

**CodeDynamics** Commercial dynamic analysis

**Open Logic Support** Enterprise-grade SLA support for 850+ packages

**OpenLogic Audits** Detailed open source license and security risk guidance

**TotalView for HPC** Scalable parallel debugging

**Zend Server** Enterprise PHP app server

**Zend Studio** PHP IDE

**Zend Guard** PHP encoding and obfuscation



## Libraries

**SourcePro** OS, database, network, and analysis abstraction for C++

**Visualization** Real-time data visualization at scale

**PV-WAVE** Visual data analysis

**IMSL Numerical Libraries** Scalable math and statistics algorithms

**HydraExpress** SOA/C++ modernization framework

**HostAccess** Terminal emulation for Windows

**Stingray** MFC GUI components

# We enable mission-critical workloads

Used by 3,000 customers in over 57 countries across diverse industries to develop mission-critical applications and software



Financial Services



Telecom



Gov't / Defense



Technology



Other Verticals





# Our vision

---

Total commitment to the highest quality code

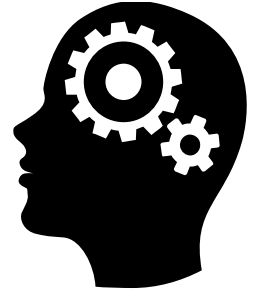
Leveraging the value of code across multiple platforms

Secure code that meets compliance and regulatory standards

Accelerating cycle times

Embracing change and open source in the enterprise

Simplifying the complex – solving industry challenges



# How we see software today

---

**Blending** languages, commercial, open source, and custom software and platforms is the norm

**Demands for faster, better code** in enterprise, embedded and web development rises every day

The mix of languages, components, frameworks, and governance add to the **complexity** of building mission-critical, enterprise systems

Software **development and deployment is fragmented** – tools, languages, frameworks, and environments

**Developers are empowered** to use the tools they want

# TotalView Overview

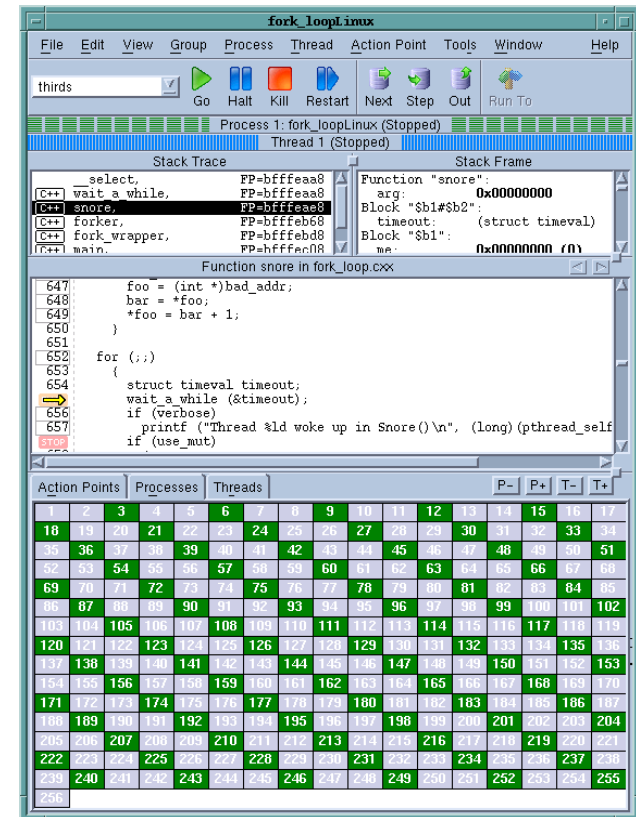
# What is TotalView®?

## Application Analysis and Debugging Tool: Code Confidently

- Debug and Analyse C/C++ and Fortran on Linux™, Unix or Mac OS X
- Laptops to supercomputers
- Makes developing, maintaining, and supporting critical apps easier and less risky

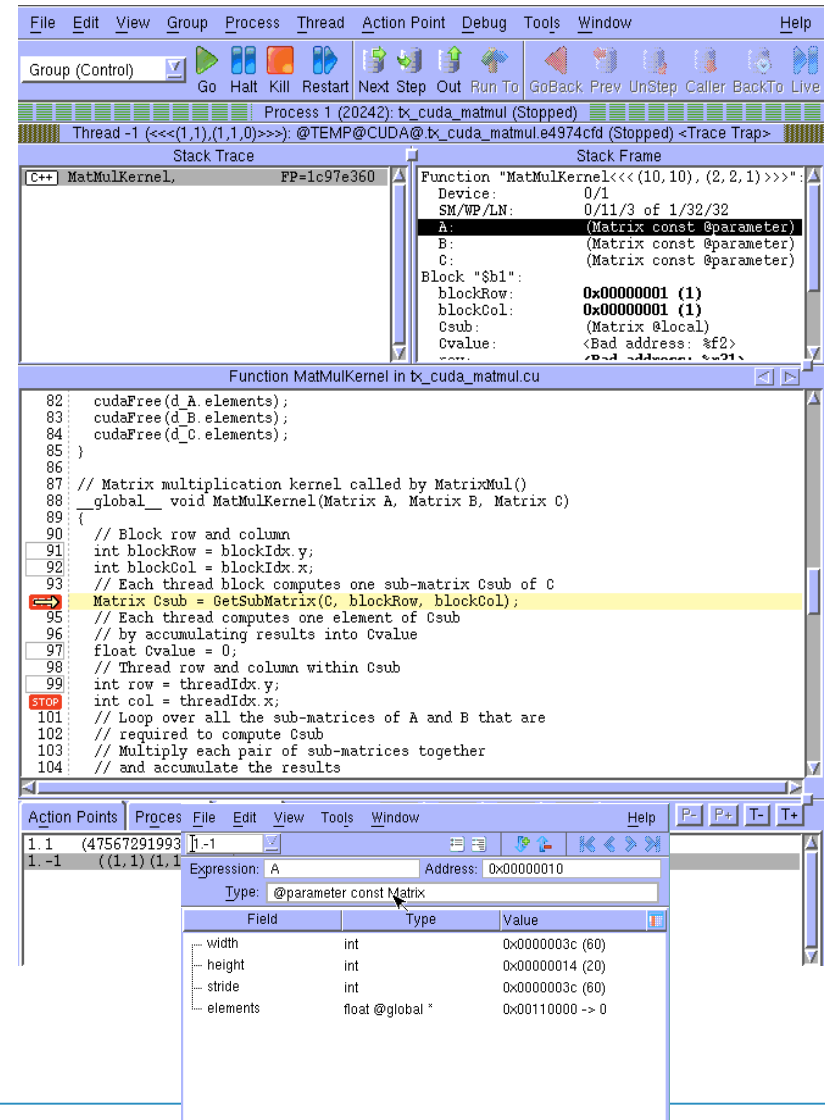
## Major Features

- Easy to learn **graphical user interface** with data visualization
- **Parallel Debugging**
  - MPI, Pthreads, OpenMP™
  - CUDA™, OpenACC®, and Intel® Xeon Phi™ coprocessor
- **Low** tool overhead resource usage
- Includes a **Remote Display Client** which frees you to work from anywhere
- **Memory Debugging** with MemoryScape™
- Deterministic **Replay Capability** Included on Linux/x86-64
- Non-interactive **Batch Debugging** with TVScript and the CLI
- **TTF & C++View** to transform user defined objects



# TotalView for the NVIDIA® GPU Accelerator

- How do you debug software running on CUDA today?
- It is hard because of its specific architecture
- TotalView supports the latest and greatest
- CUDA 6.0, 6.5, 7.0
- Cray CCE OpenACC
- Features and capabilities include
  - MPI-based clusters
  - Multi-card configurations
  - Flexible Display and Navigation
    - Physical (device, SM, Warp, Lane)
    - Logical (Grid, Block) tuples
  - Shows what is running where
  - Support for separate memory address spaces
  - Leverages CUDA memcheck
  - **Supports CUDA dynamic parallelism**



# TotalView for Intel® Xeon Phi™

Supports all major Intel Xeon Phi Coprocessor configurations

- Native Mode
  - With or without MPI
- Offload Directives
  - Incremental adoption, similar to GPU
- Symmetric Mode
  - Host and Coprocessor
- Multi-device, Multi-node
- Clusters

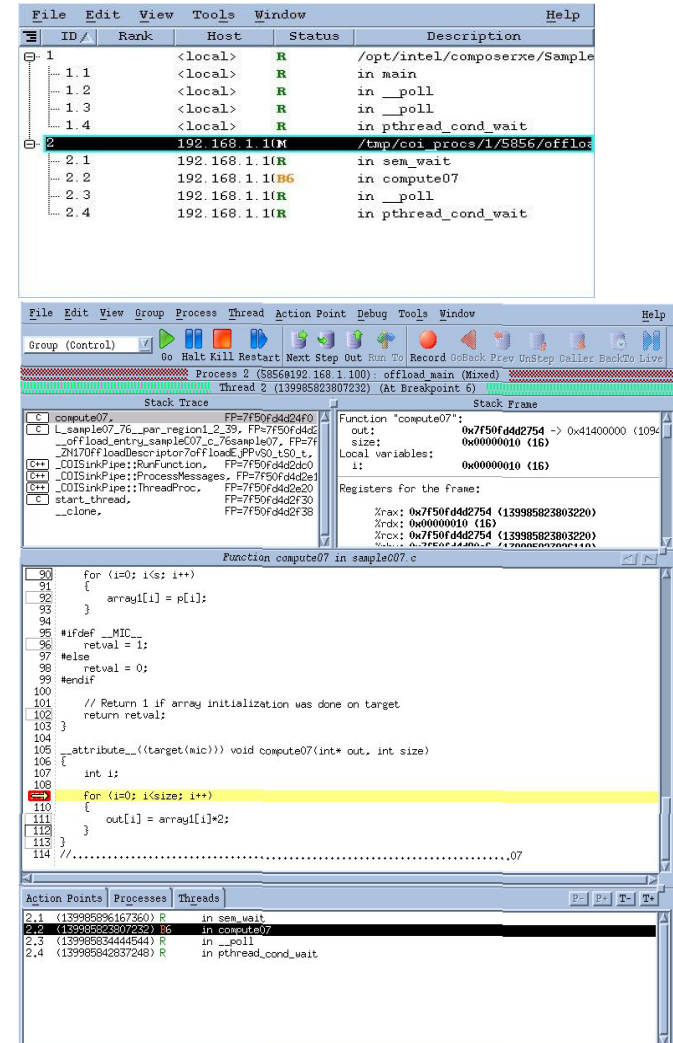
## User Interface

- MPI Debugging Features
  - Process Control, View Across, Shared Breakpoints
- Heterogeneous Debugging
  - Debug Both Xeon and Intel Xeon Phi Processes

## Memory Debugging

- Both native and symmetric mode

Set up well for supporting KNL when it is available

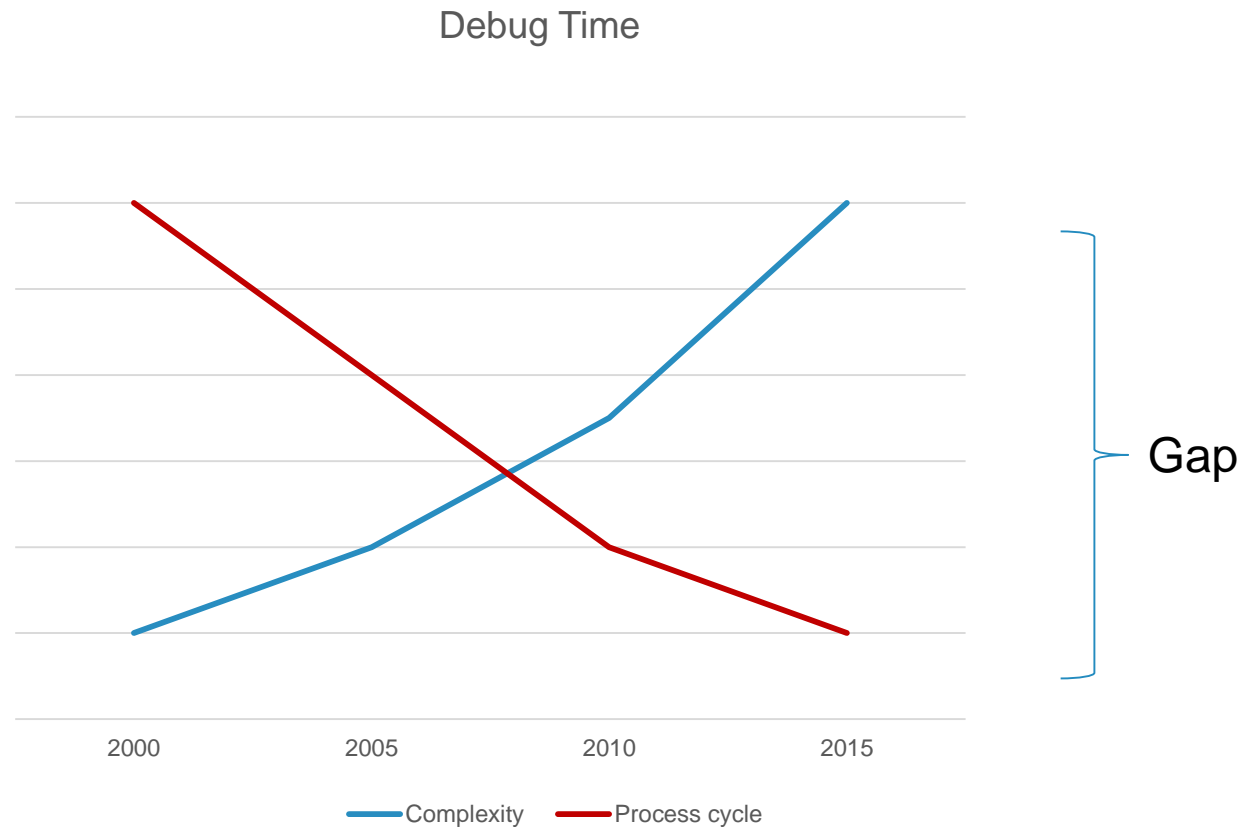


**Debugging is challenging**

**How does TotalView help?**

# What is the problem?

- Complexity
  - Threading, Cores
  - Data types
  - LOC
  - Memory Usage
- Iteration time
  - Growth in agile



Gap needs closure or majority of development spent fixing defects



# Defect analysis is a bottleneck

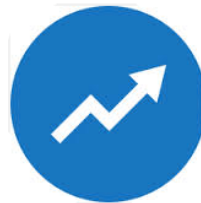
---

- Code Correctness, Bug troubleshooting, Memory Analysis
  - Some defects can take weeks to analyze
- Techniques include
  - Logging, Print statements, Debuggers
- Bugs are costly
  - The pressure is high
  - Releases slip
  - Customers get frustrated

# The solution



Reduce complexity



Increase productivity



Address new technologies

# How does TotalView help?

---

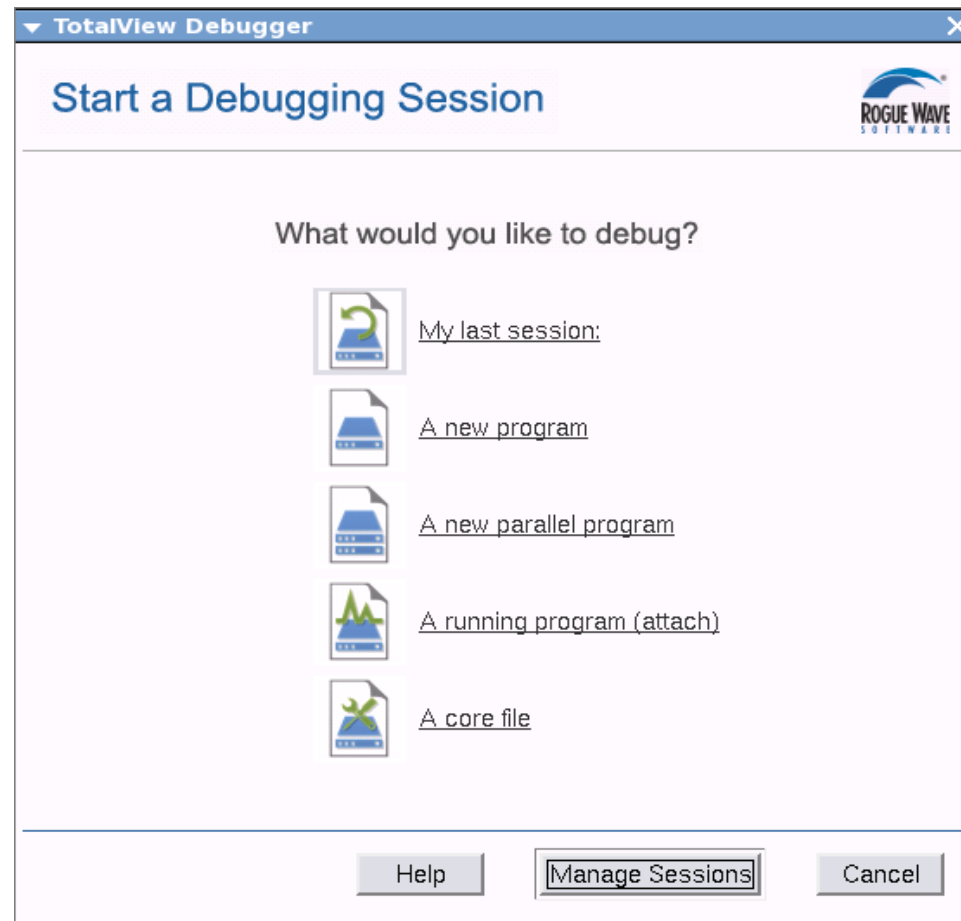
## TotalView debugger

- Troubleshooting and analysis tool
  - Visibility into applications
  - Control over applications
- Scalability
- Usability
- Support for HPC platforms and languages

# Getting Started

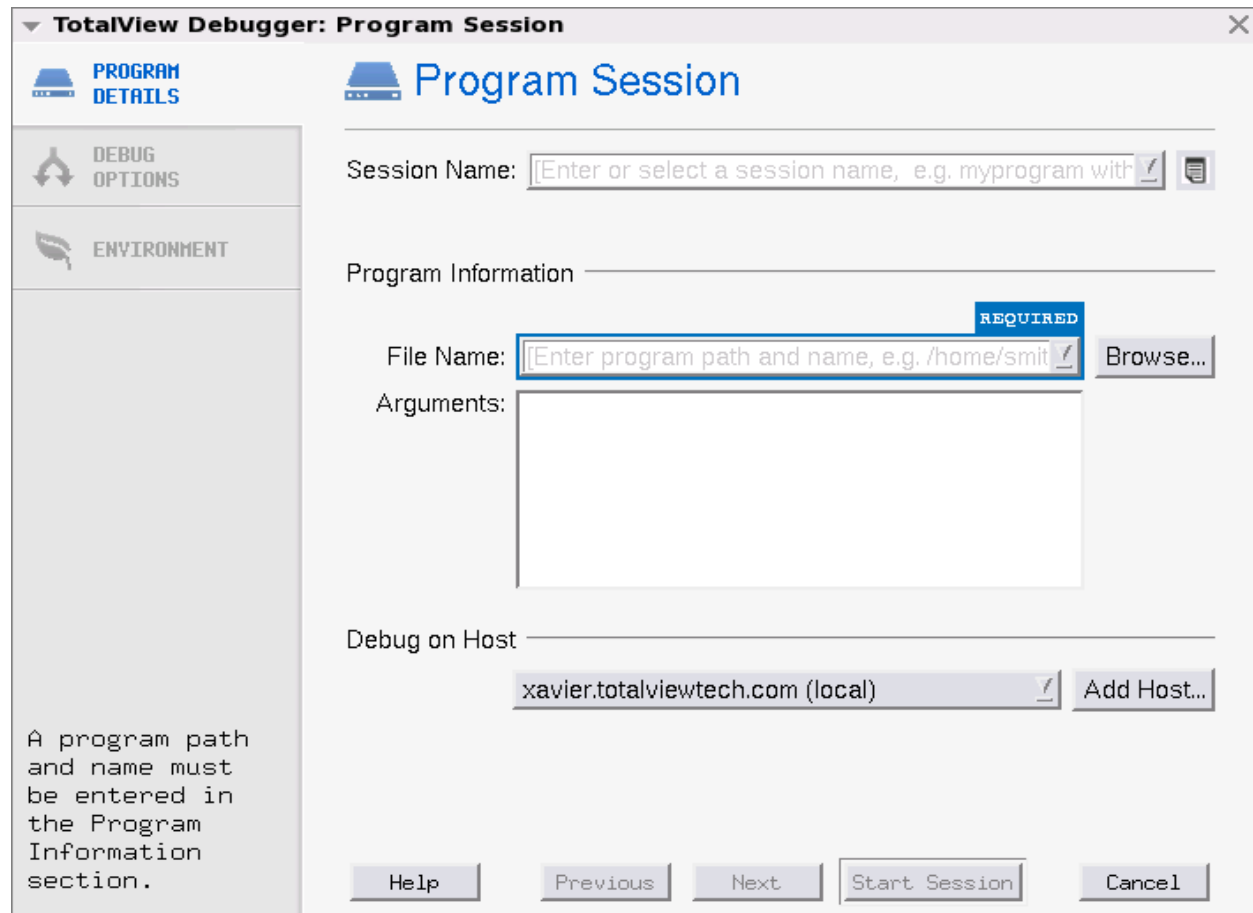
# Starting TotalView

## Sessions Manager



# Starting TotalView

## Debugging a new Program



The image shows the 'TotalView Debugger: Program Session' dialog box. It has a sidebar on the left with three tabs: 'PROGRAM DETAILS' (selected), 'DEBUG OPTIONS', and 'ENVIRONMENT'. The main area is titled 'Program Session' and contains the following fields:

- Session Name:** A text field with a placeholder '[Enter or select a session name, e.g. myprogram with]'. There is a small icon to the right of the field.
- Program Information:** A section header.
- File Name:** A text field with a placeholder '[Enter program path and name, e.g. /home/smit]'. A red 'REQUIRED' label is above the field. To the right is a 'Browse...' button.
- Arguments:** A large empty text area.
- Debug on Host:** A section header.
- Host:** A dropdown menu showing 'xavier.totalviewtech.com (local)'. To the right is an 'Add Host...' button.

At the bottom of the dialog are five buttons: 'Help', 'Previous', 'Next', 'Start Session', and 'Cancel'.

A program path and name must be entered in the Program Information section.

# Starting TotalView

## Debugging a new Program

The screenshot shows the 'TotalView Debugger: Program Session' dialog box. On the left is a sidebar with three tabs: 'PROGRAM DETAILS' (selected), 'DEBUG OPTIONS', and 'ENVIRONMENT'. Below these tabs is a large text area with the instruction: 'Press Start Session to start the debugging session.' The main area of the dialog is titled 'Program Session' and contains the following fields and controls:

- Session Name:** A text field containing 'Wave'.
- Program Information:**
  - File Name:** A text field containing 'talview.8X.12.0-0/linux-x86-64/examples/wave'. A blue 'REQUIRED' label is positioned above the field. To the right is a 'Browse...' button.
  - Arguments:** A large text area containing the placeholder text '[Enter any program arguments. Ex. -option foo]'.
- Debug on Host:** A text field containing 'xavier.totalviewtech.com (local)'. To the right is an 'Add Host...' button.

At the bottom of the dialog are five buttons: 'Help', 'Previous', 'Next', 'Start Session' (highlighted in green), and 'Cancel'.

# Starting TotalView

## Adding a remote host

**PROGRAM DETAILS**

**DEBUG OPTIONS**

**ENVIRONMENT**

**Program Session**

Session Name: [Enter or select a session name, e.g. myprogram with

[Enter notes about this session.]

Program Information

File Name: REQUIRED [Enter program path and name, e.g. /home/smit Browse...

Arguments: [Enter any program arguments. Ex. -option foo ]

Debug on Host

xavier.totalviewtech.com (local) Add Host...

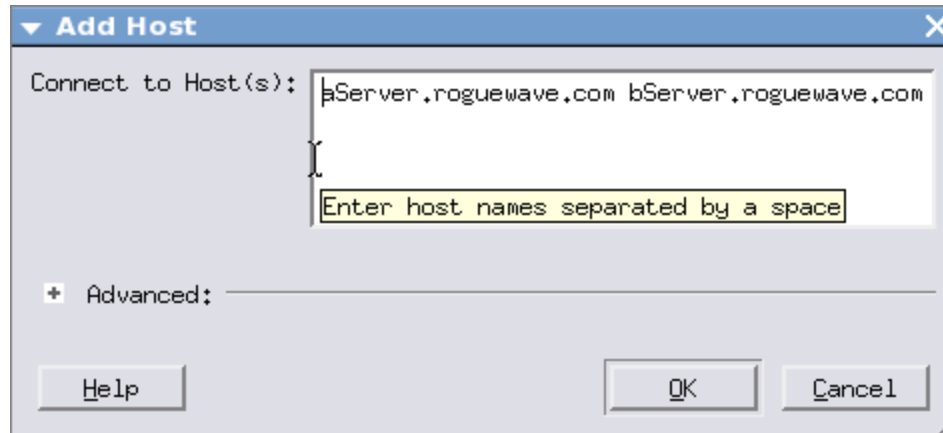
Add host(s) to the host list

A program path and name must be entered in the File Name field.



# Starting TotalView

## Adding a remote host



# Starting TotalView

## Attach to running program(s)

**TotalView Debugger: Attach to running program(s)**

**ATTACH DETAILS**

**DEBUG OPTIONS**

Session Name:

Processes

Host:   User:

Program	State	PID	PPID	Host	Path
ssh	S	14518	14516	visor.totalviewtech.com	/usr/sbin/
└ tcsh	S	14519	14518	visor.totalviewtech.com	/bin/
tcsh	S	18329	18328	visor.totalviewtech.com	/bin/
└ tvdsvrmain	R	18351	18329	visor.totalviewtech.com	/nfs/san0/user

PID & Program

PID:  **REQUIRED**


File Name:  **REQUIRED**


Select process(es) or enter a PID. PID & Program file are required.

# Starting TotalView

## Open a Core File

▼ TotalView Debugger: Core File Session

 PROGRAM DETAILS

 Core File Session

Session Name:

Core File

File Name:  REQUIRED

Program Information


File Name:  REQUIRED


Debug on Host


Press Start Session to start the debugging session.


# Starting TotalView

## Debug options

 PROGRAM  
DETAILS

 DEBUG  
OPTIONS

 ENVIRONMENT

 Program Session

---

Reverse Debugging

---

*Step and debug in reverse from any point during execution.*

☐ Enable reverse debugging with ReplayEngine

Memory Debugging

---

*Dynamically track memory allocations, catch memory errors and view memory analysis reports.*

☐ Enable memory debugging

☐ Suppress memory error notifications

CUDA Debugging

---

*Detect global memory addressing violations and misaligned memory accesses for CUDA based programs.*

☐ Enable CUDA memory checking

# Starting TotalView

## Environment

TotalView Session: Filter (on xavier)

**PROGRAM DETAILS**

**DEBUG OPTIONS**

**ENVIRONMENT**

Enter the environment variables that the process needs to run.

Press Start Session to start the debugging session.

### Program Session

Program Environment

Enter environment variables to add to your program's environment:

[Enter comma or line-separated NAME=VALUE pairs]

Input Processing

Read standard input from file: [Enter input file name and path] **Browse...**

Standard and Error Output Processing

- ☒ Write to terminal
- ☒ Write all output to the same file
  - Output file: [Enter output file name] **Browse...** ☐ Append
- ☒ Write standard and/or error output to separate files
  - Standard output file: [Enter output file name] **Browse...** ☐ Append
  - Standard error file: [Enter error file name] **Browse...** ☐ Append

**Help** **Previous** **Next** **Start Session** **Cancel**

# Starting TotalView

## Via Command Line

### Normal

```
totalview [ tv_args ] prog_name [-a prog_args ]
```

### Attach to running program

```
totalview [ tv_args ] prog_name -pid PID# [-a prog_args ]
```

### Attach to remote process

```
totalview [ tv_args ] prog_name -remote name [-a prog_args ]
```

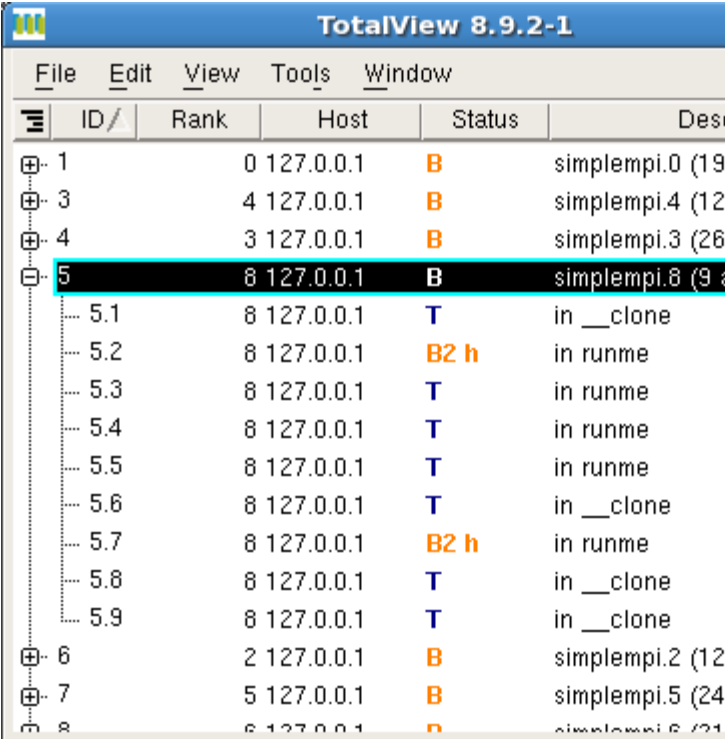
### Attach to a core file

```
totalview [ tv_args ] prog_name corefile_name [ -a prog_args ]
```

# Navigating your debugging session

# TotalView Root Window

- State of all processes being debugged
- Process and Thread status
- Instant navigation access
- Sort and aggregate by status



The screenshot shows the TotalView 8.9.2-1 Root Window. It contains a table with columns: ID/, Rank, Host, Status, and Description. The table lists several processes and their threads. Process 5 is highlighted in blue. The status column uses color-coded letters: B (Breakpoint), T (Stopped), E (Error), W (Watchpoint), R (Running), M (Mixed), and H (Held).

ID/	Rank	Host	Status	Description
1	0	127.0.0.1	B	simplempi.0 (19)
3	4	127.0.0.1	B	simplempi.4 (12)
4	3	127.0.0.1	B	simplempi.3 (26)
5	8	127.0.0.1	B	simplempi.8 (9)
5.1	8	127.0.0.1	T	in __clone
5.2	8	127.0.0.1	B2 h	in runme
5.3	8	127.0.0.1	T	in runme
5.4	8	127.0.0.1	T	in runme
5.5	8	127.0.0.1	T	in runme
5.6	8	127.0.0.1	T	in __clone
5.7	8	127.0.0.1	B2 h	in runme
5.8	8	127.0.0.1	T	in __clone
5.9	8	127.0.0.1	T	in __clone
6	2	127.0.0.1	B	simplempi.2 (12)
7	5	127.0.0.1	B	simplempi.5 (24)
8	6	127.0.0.1	B	simplempi.6 (21)

## ➤ Status Info

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



# TotalView Root Window

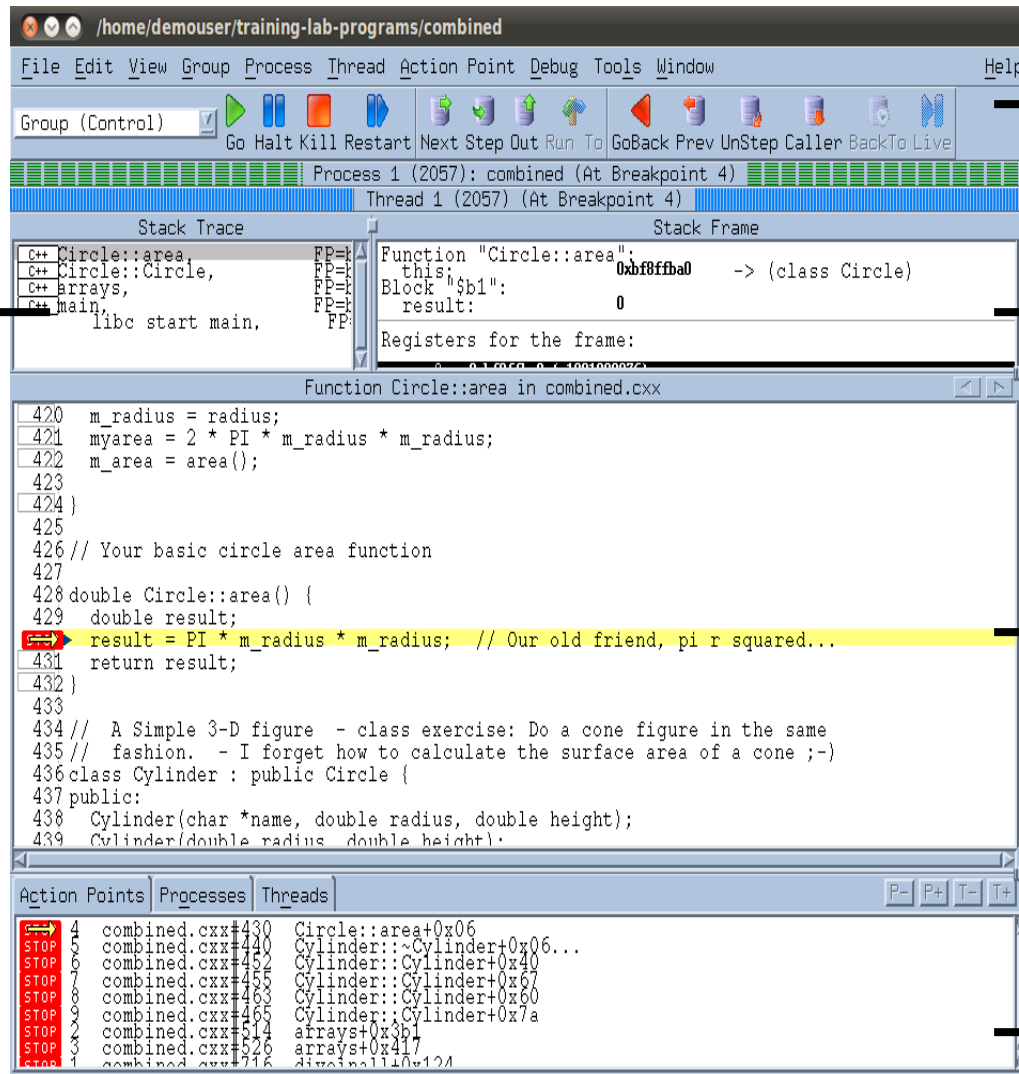
The screenshot shows the TotalView 8.9.2-1 Root Window. The window has a menu bar (File, Edit, View, Tools, Window, Help) and a table with columns: ID/, Rank, Host, Status, and Description. The table lists MPI processes and their threads. Annotations point to various UI elements:

- Host name:** Points to the 'Host' column.
- Hierarchical/Linear Toggle:** Points to the expand/collapse icons in the left margin.
- Rank # (if MPI program):** Points to the 'Rank' column.
- TotalView Thread ID #:** Points to the 'ID/' column.
- Expand - Collapse Toggle:** Points to the expand/collapse icons in the left margin.
- Process Status:** Points to the 'Status' column.
- Action Point ID number:** Points to the 'ID/' column.

ID/	Rank	Host	Status	Description
1	0	127.0.0.1	B	simplempi.0 (19 active threads)
3	4	127.0.0.1	B	simplempi.4 (12 active threads)
4	3	127.0.0.1	B	simplempi.3 (26 active threads)
5	8	127.0.0.1	B	simplempi.8 (9 active threads)
5.1	8	127.0.0.1	T	in __clone
5.2	8	127.0.0.1	B2 h	in runme
5.3	8	127.0.0.1	T	in runme
5.4	8	127.0.0.1	T	in runme
5.5	8	127.0.0.1	T	in runme
5.6	8	127.0.0.1	T	in __clone
5.7	8	127.0.0.1	B2 h	in runme
5.8	8	127.0.0.1	T	in __clone
5.9	8	127.0.0.1	T	in __clone
6	2	127.0.0.1	B	simplempi.2 (12 active threads)
7	5	127.0.0.1	B	simplempi.5 (24 active threads)
8	6	127.0.0.1	B	simplempi.6 (21 active threads)

- Dive to refocus
- Dive in new window to get a second process window

# Process Window Overview



Toolbar

Stack Trace Pane

Stack Frame Pane

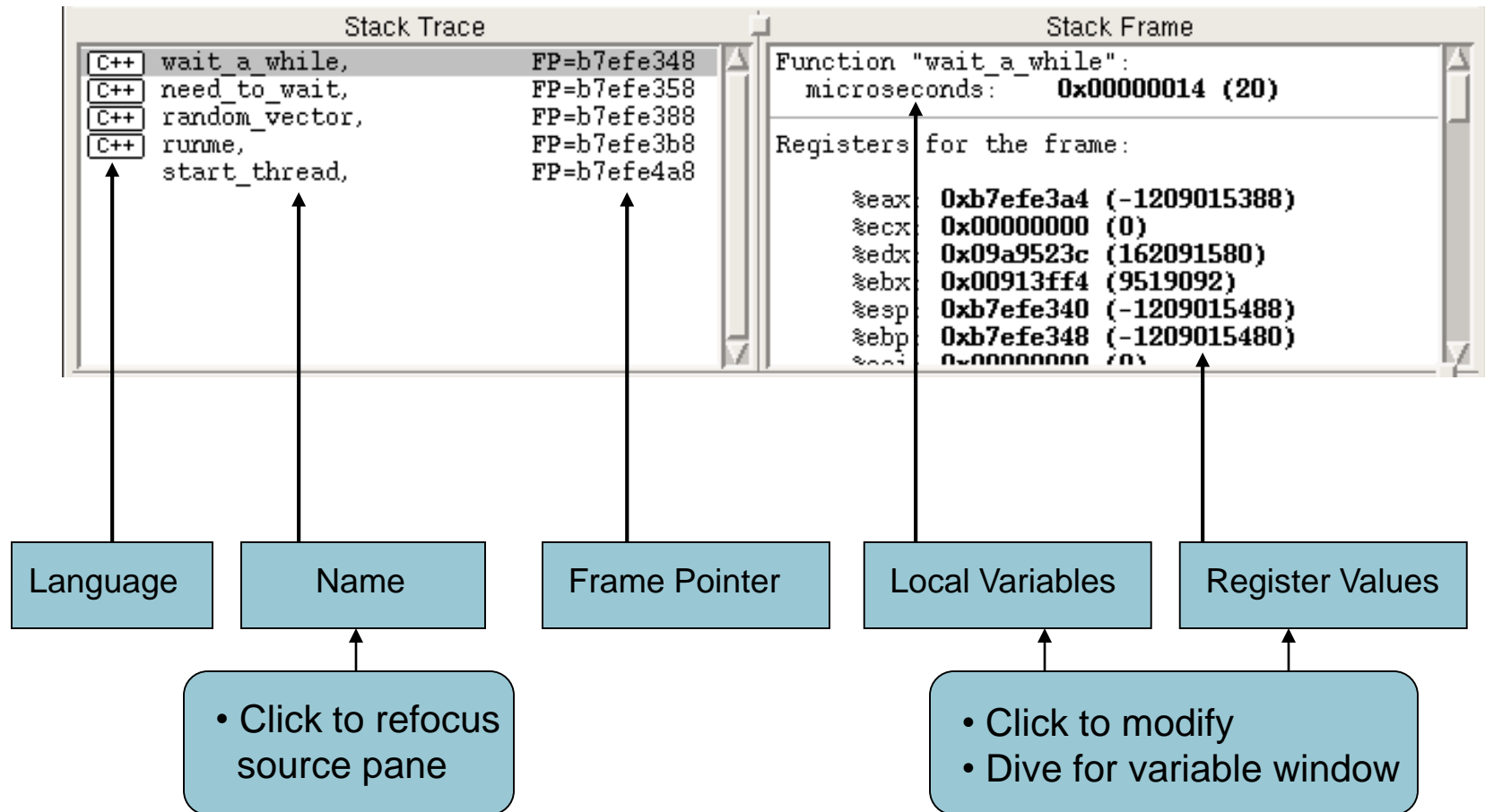
Source Pane

Tabbed Area

Provides detailed  
state of one process,  
or a single thread  
within a process

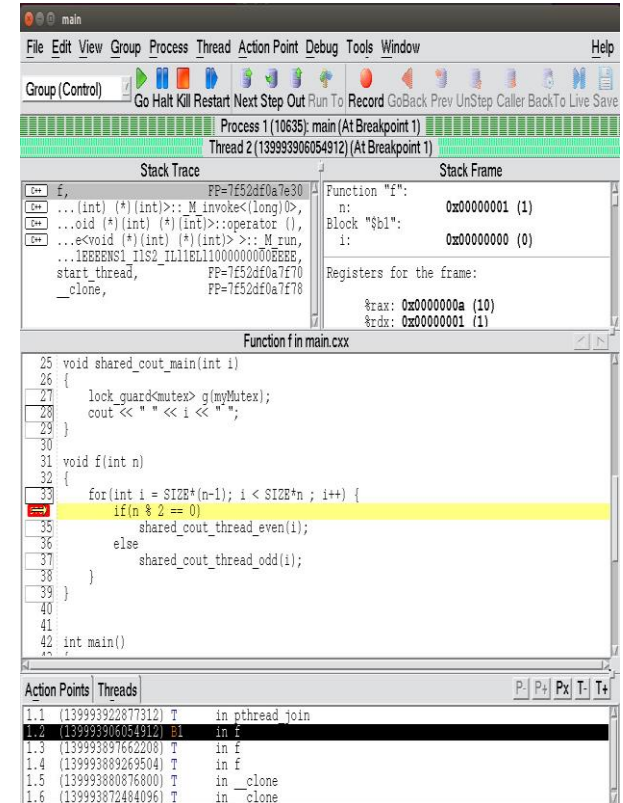
A single point of  
control for the  
process and other  
related processes

# Stack Trace and Stack Frame Panes

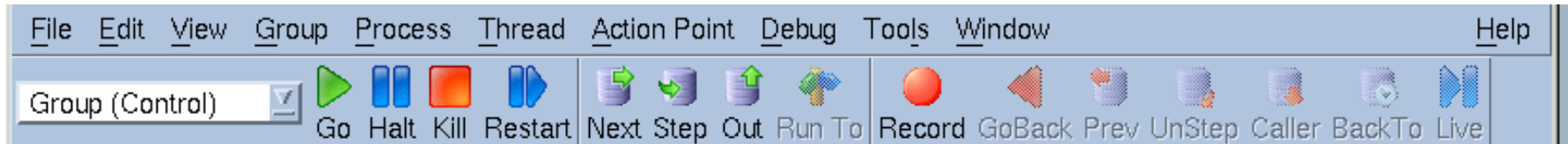


# Multi-threading made easier

- How do you debug a problem that occurs in 1 thread in a 50 thread application?
  - Without TotalView,
    - Set a breakpoint in code
    - Run and hope you hit the right thread
  - With TotalView
    - set thread specific breakpoints
- Better multithreaded debugger
  - Understand the state of all of your threads
  - Focus on specific threads
    - View stack and data
- Built to scale to HPC and leveraging that in mainstream commercial envs



# Deterministic Replay Debugging



- Reverse Debugging: Radically simplify your debugging
  - Captures and Deterministically Replays Execution
    - Not just “checkpoint and restart”
  - Eliminate the Restart Cycle and Hard-to-Reproduce Bugs
  - Step Back and Forward by Function, Line, or Instruction
- Specifications
  - A feature included in TotalView on Linux x86 and x86-64
    - No recompilation or instrumentation
    - Explore data and state in the past just like in a live process, including C++View transformations
  - Replay on Demand: enable it when you want it
  - Supports MPI on Ethernet, Infiniband, Cray XE Gemini
  - Supports Pthreads, and OpenMP
  - **New: Save / Load Replay Information**

```
40
41
42  int  funcB(int
43  int  c;
44  int  i;
45  int  v[MAXDEPT
46  int  *p;
    → c=b+2;
48  p=&c;
49  if(c<MAXDEPTH
50      c=funcA(c);
51  for (i=array1
52      v[i]=*p;
```

# Remote Display Client

- Offers users the ability to easily set up and operate a TotalView debug session that is running on another system
- Consists of two components
  - Client – runs on local machine
  - Server – runs on any system supported by TotalView and “invisibly” manages the secure connection between host and client
- Remote Display Client is available for:
  - Linux x86, x86-64
  - Windows XP, Vista, 7
  - Mac OS X

The screenshot shows the 'TotalView Remote Display Client' window. On the left is a 'Session Profiles' list with 'nvidia5' selected. The main area contains four numbered steps for configuring a debug session:

- 1. Enter the Remote Host to run your debug session:**  
Remote Host:  User Name:  Advanced Options
- 2. As needed, enter hosts in access order to reach the Remote Host:**  
A table with columns: Host, Access By, Access Value, Commands.

Host	Access By	Access Value	Commands
1 prodigy.totalviewtech.c...	User Name	petert	
2	User Name		
- 3. Enter settings for the debug session on the Remote Host :**  
TotalView | MemoryScope |  
Path to TotalView on Remote Host:   
Arguments for TotalView:   
Your Executable (path & name):   
Arguments for Your Executable:   
Submit Job to Batch Queueing System:
- 4. Enter batch submission settings for the Remote Host :**  
LoadLeveler Submit Command:   
TotalView LoadLeveler Script to Run:   
Additional LoadLeveler Options:

At the bottom right is a 'Launch Debug Session' button. The status bar at the bottom says 'No session running'.

# Viewing your data

# Breakpoints

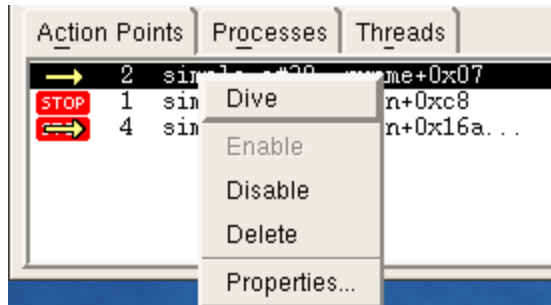


***Breakpoints***

***Barrier Breakpoints***

***Conditional Breakpoints***

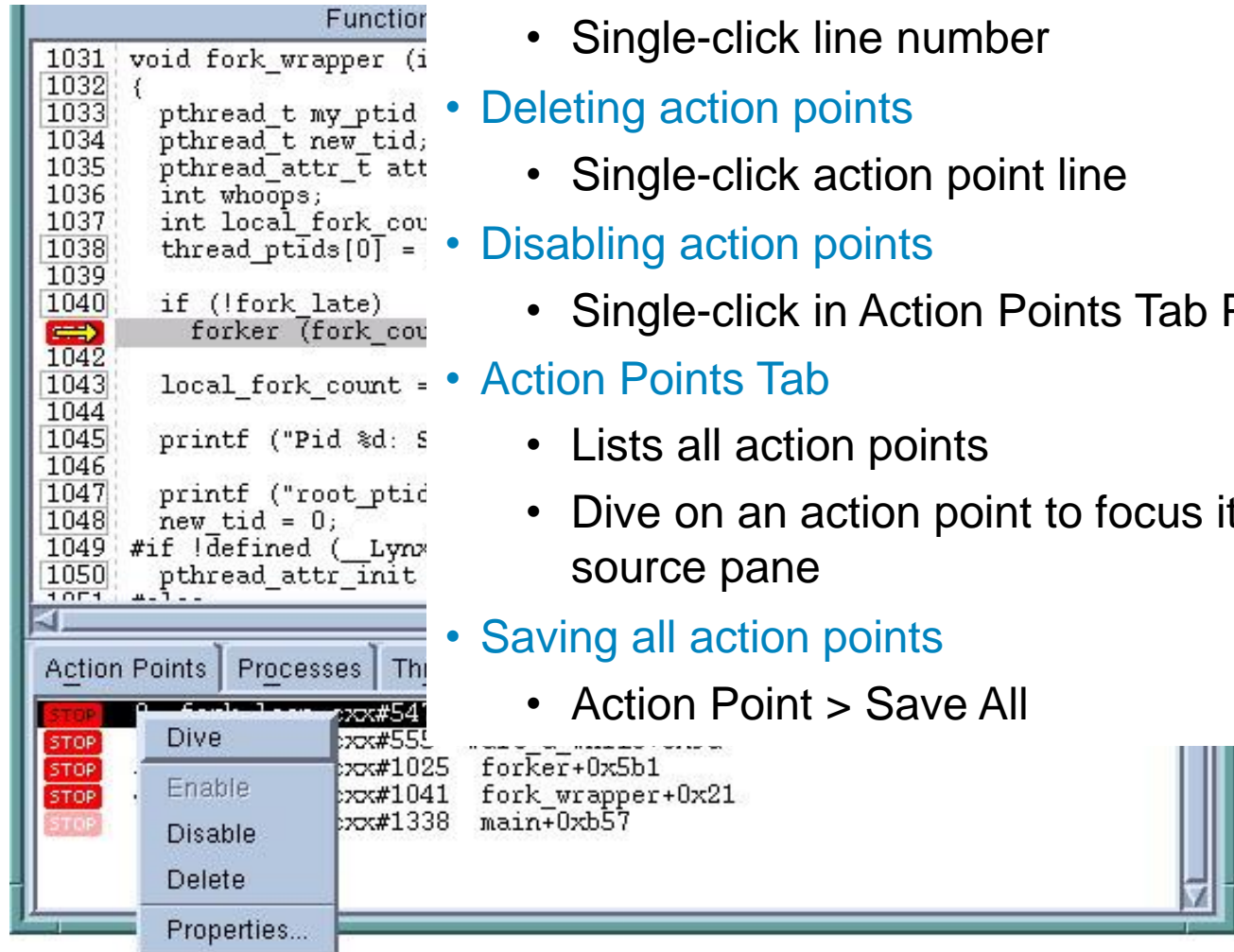
***Evaluation Breakpoints***



***Watchpoints***



# Setting Breakpoints

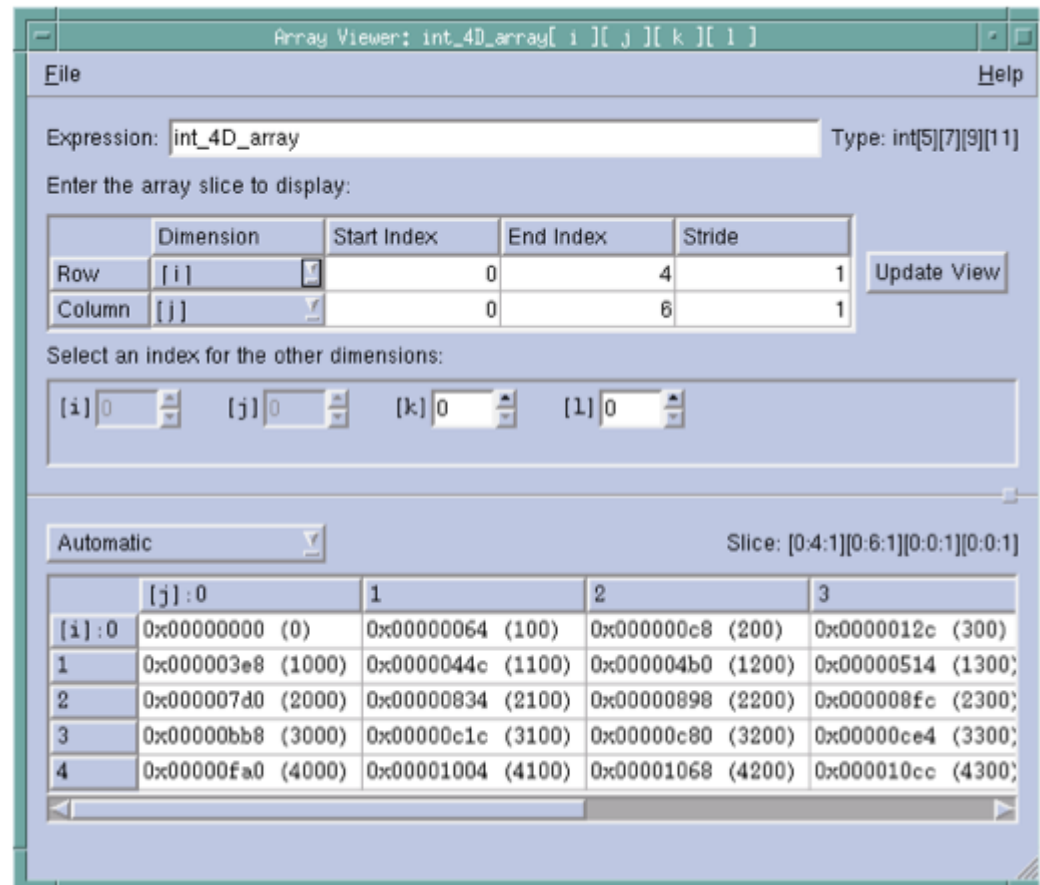


The screenshot displays a debugger window with a source code pane on the left and an 'Action Points' pane on the right. The source code is for a function named `fork_wrapper` and includes lines 1031 through 1051. A red arrow icon is positioned next to line 1041, which contains the code `forker(fork_count);`. The 'Action Points' pane on the right lists several action points, each with a red 'STOP' button and a memory address. A context menu is open over the first action point, showing options: 'Dive', 'Enable', 'Disable', 'Delete', and 'Properties...'. The memory addresses listed are `xxx#559`, `xxx#1025`, `xxx#1041`, and `xxx#1338`. The corresponding code snippets are `forker+0x5b1`, `fork_wrapper+0x21`, and `main+0xb57`.

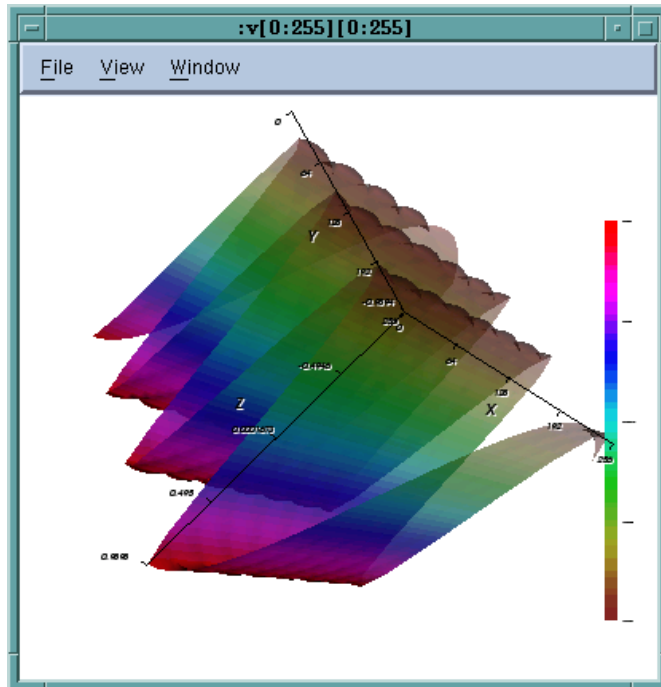
- Setting action points
  - Single-click line number
- Deleting action points
  - Single-click action point line
- Disabling action points
  - Single-click in Action Points Tab Pane
- Action Points Tab
  - Lists all action points
  - Dive on an action point to focus it in source pane
- Saving all action points
  - Action Point > Save All

# Multi-dimensional array viewer

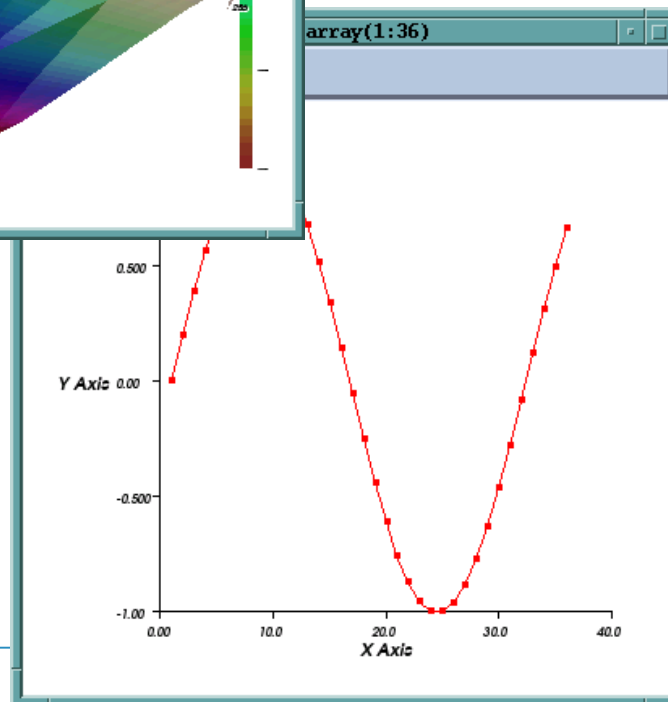
- See your arrays on a Grid display
- 2-D, 3-D, ...N-D
- Arbitrary slices
- Specify data representation
- Windowed data access – Fast



# Visualizing Arrays



- Visualize array data using Tools > Visualize from the Variable Window
- Large arrays can be sliced down to a reasonable size first
- Visualize is a standalone program
- Data can be piped out to other visualization tools

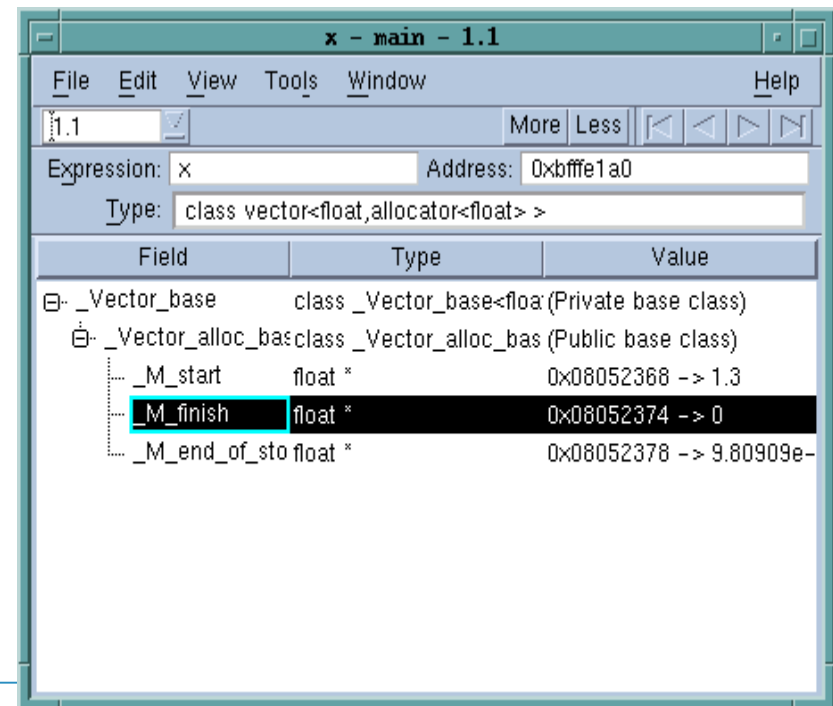
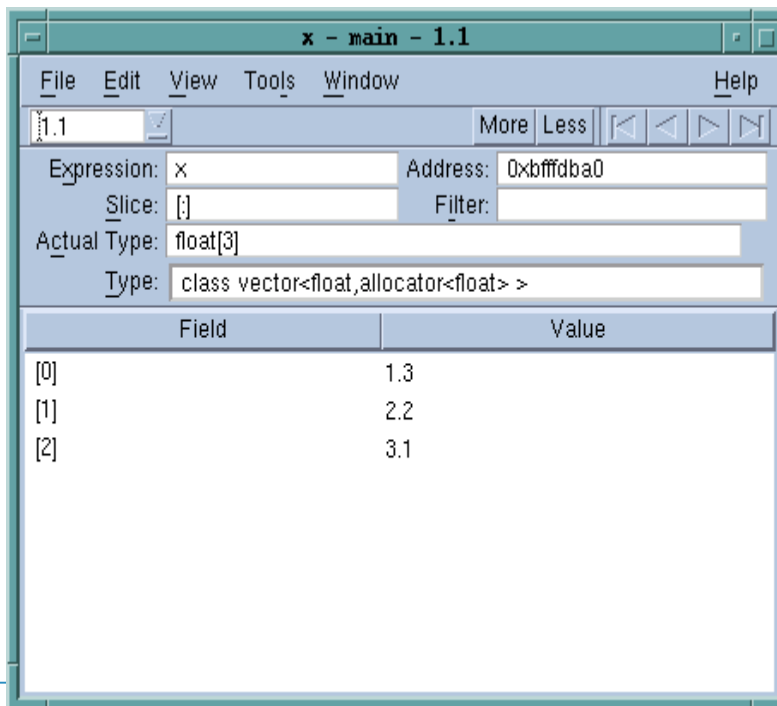


- Visualize allows to spin, zoom, etc.
- Data is not updated with Variable Window; You must revisualize
- `$visualize()` is a directive in the expression system, and can be used in evaluation point expressions.

# STLView

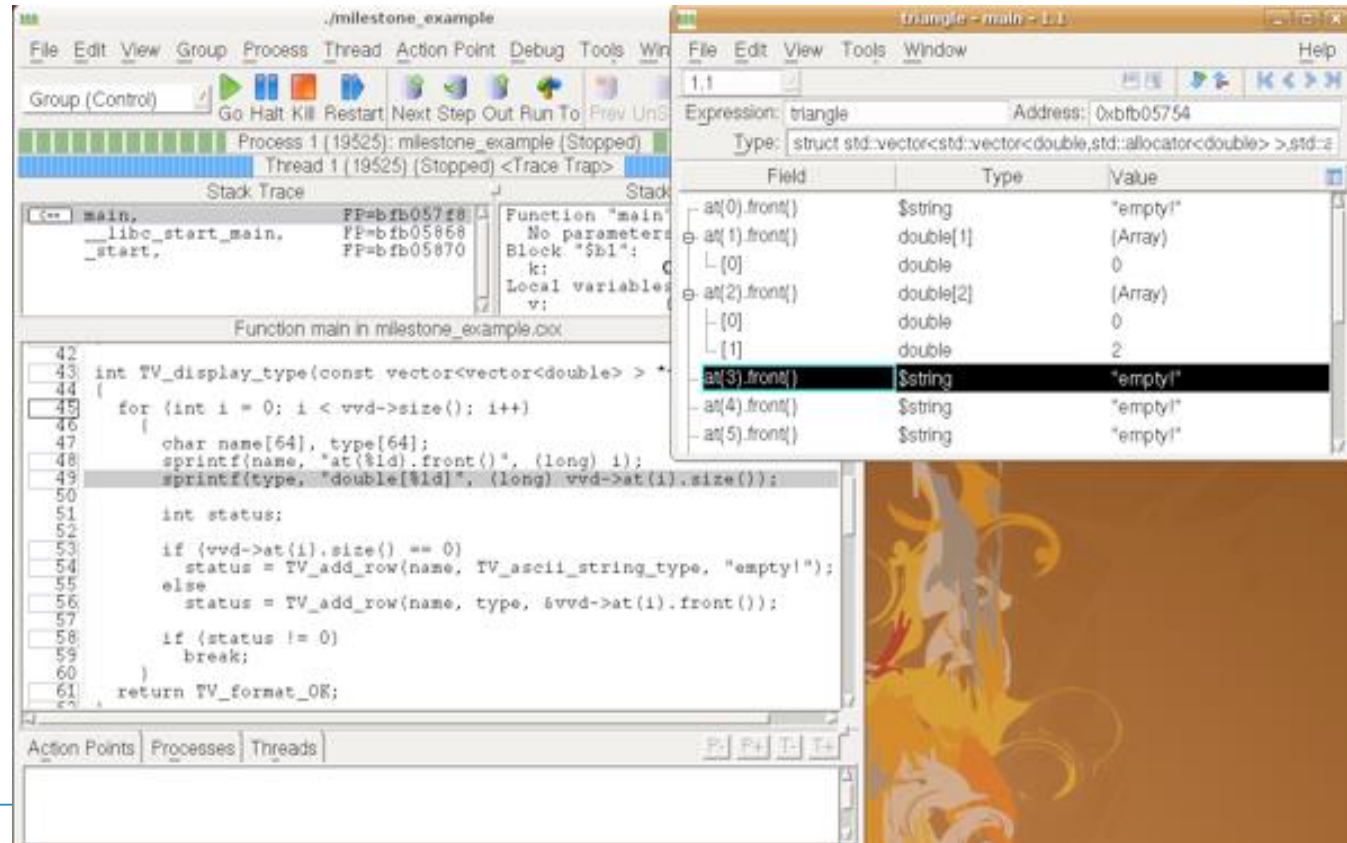
STLView transforms templates into readable and understandable information

vector, list, map, string, set, multiset and multimap



# C++View

- C++View is a simple way for you to define type transformations
  - Simplify complex data
  - Aggregate and summarize
  - Check validity
- Transforms
  - Type-based
  - Compose-able
  - Automatically visible
- Code
  - C++
  - Easy to write
  - Resides in target
  - Only called by TotalView



# C++ View API

```
#include "tv_data_display.h"
```

```
int TV_ttf_display_type ( const T * );
```

```
int TV_ttf_add_row (  
    const char *field_name,  
    const char *type_name,  
    const char *address );
```

# C++ View Example

```
class A {
    int i;
    char *s;
};

class B {
    A a;
    double d;
};

int TV_ttf_display_type ( const A *a )
{
    (void) TV_ttf_add_row ( "i", TV_ttf_type_int, &(a->i) );
    (void) TV_ttf_add_row ( "s", TV_ttf_type_ascii_string, a->s );
}

int TV_ttf_display_type ( const B *b )
{
    (void) TV_ttf_add_row ( "a", "A", &(b->a) );
    (void) TV_ttf_add_row ( "d", "double", &(b->d) );
}
```

# Type Transformation Facility (TTF)

TTF is a TotalView subsystem that

- Allows you to transform the way information appears
- Doesn't require changes to your code
- TTF transforms can be stored in a .tvd file for referencing during a debugging session or at TotalView startup

QT Qstring  
class  
transform

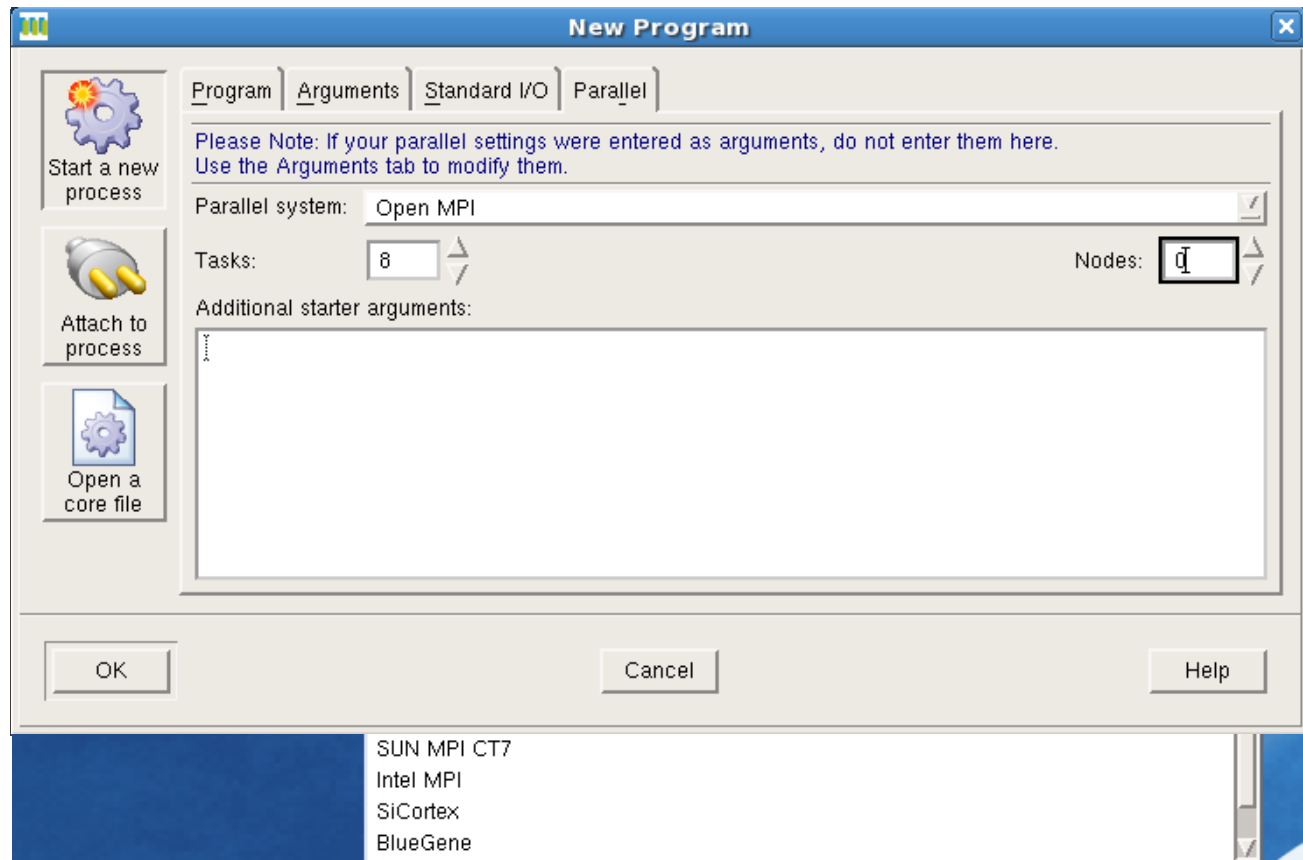
```
::TV::TTF::RTF::build_struct_transform {  
    name {^class QString$}  
    members {  
        { ascii { $wstring_u16 cast { * {d -> data} } } }  
    }  
}
```



**Parallel debugging adds  
complexity**

# TotalView Startup with MPI

In the Parallel tab, select:



your MPI preference, number of tasks, and number of nodes.  
... then add any additional starter arguments

# Command line MPI Startup

---

## Using OpenMPI

```
mpirun -tv args prog prog_args
```

As an alternative, you can invoke TotalView on mpirun.

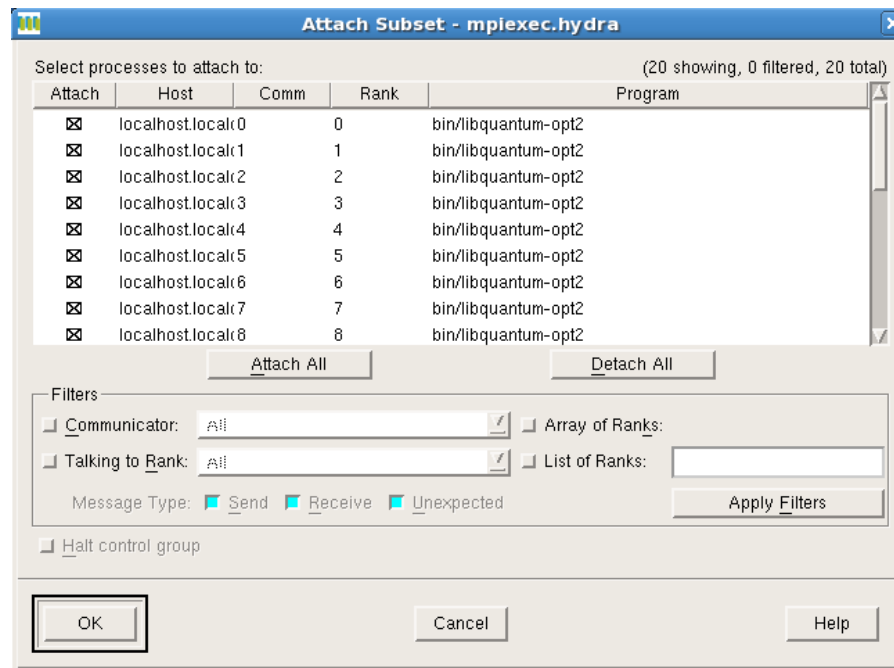
```
totalview -args mpirun args ./prog
```

For example, to start TotalView on a four-process MPI program:

```
totalview -args mpirun -np 4 ./mpi_program
```

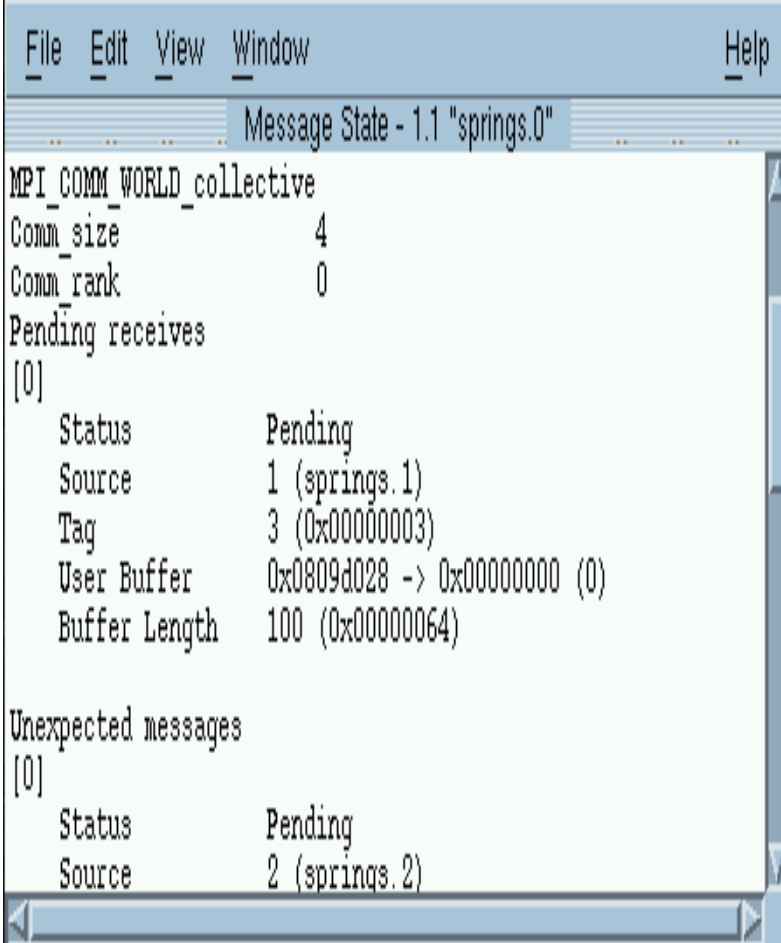
# Subset Attach

- Connecting to a subset of a job reduces tokens and overhead
- Can change this during a run
- Groups->Subset Attach



# View MPI Message Queues

- Information visible whenever MPI rank processes are halted
- Provides information from the MPI layer
  - Unexpected messages
  - Pending Sends
  - Pending Receives
- Use this info to debug
  - Deadlock situations
  - Load balancing
- May need to be enabled in the MPI library
  - --enable-debug



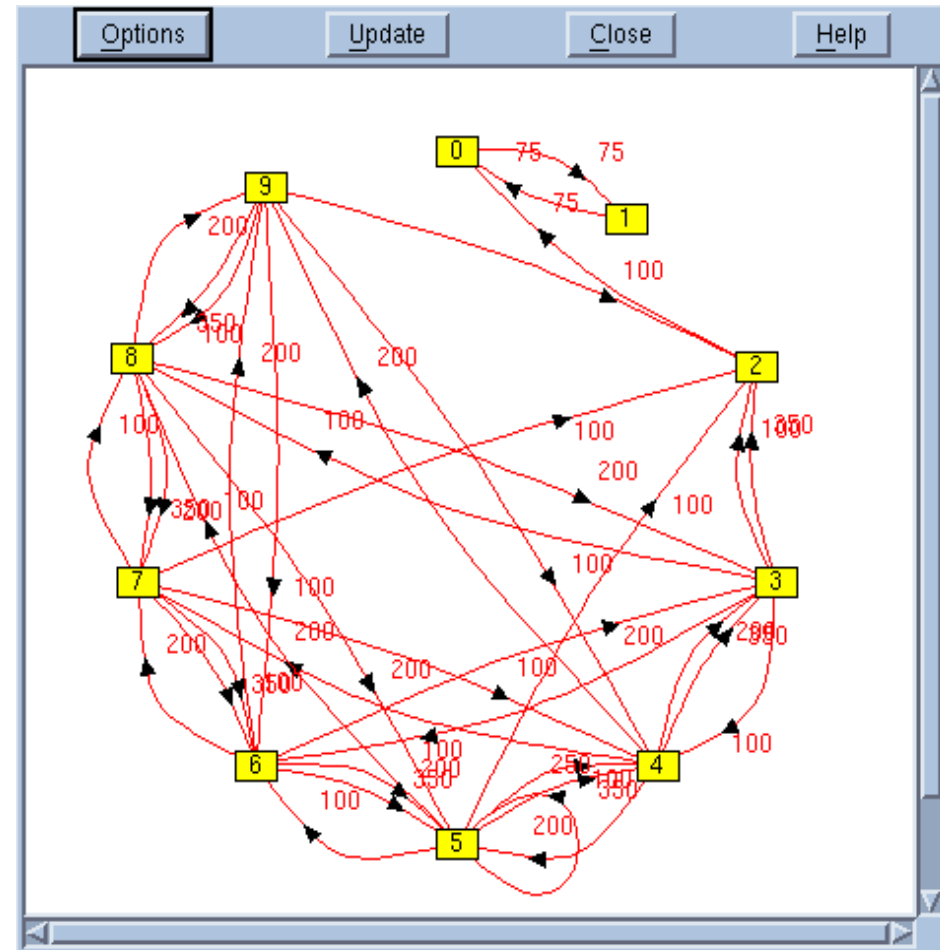
The screenshot shows a window titled "Message State - 1.1 \"springs.0\"". The window has a menu bar with "File", "Edit", "View", "Window", and "Help". The main content area displays the following information:

```
MPI_COMM_WORLD_collective
Comm_size          4
Comm_rank          0
Pending receives
[0]
  Status           Pending
  Source           1 (springs.1)
  Tag              3 (0x00000003)
  User Buffer       0x0809d028 -> 0x00000000 (0)
  Buffer Length     100 (0x00000064)

Unexpected messages
[0]
  Status           Pending
  Source           2 (springs.2)
```

# Message Queue Graph

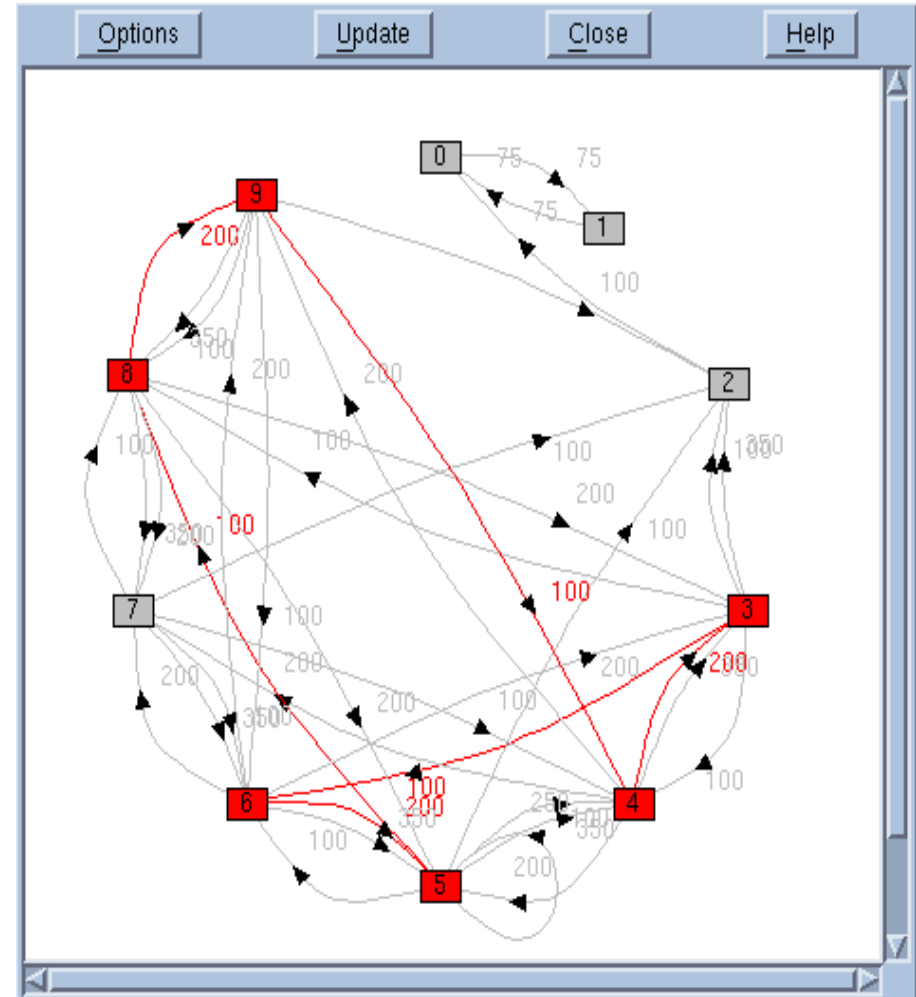
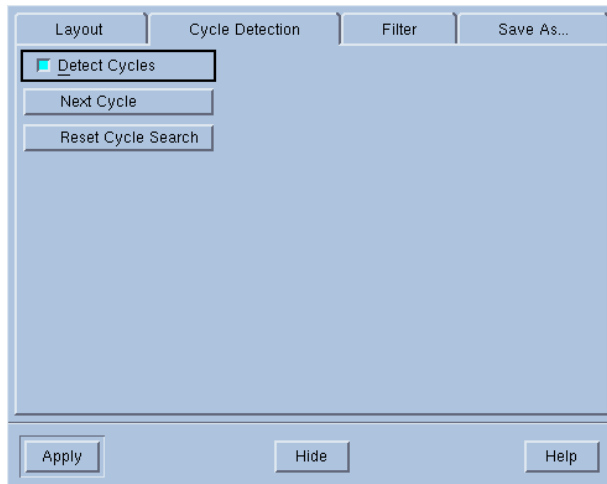
- Hangs & Deadlocks
- Pending Messages
  - Receives
  - Sends
  - Unexpected
- Inspect
  - Individual entries
- Patterns



# Message Queue Graph

## Message Queue Debugging

- **Filtering**
  - Tags
  - MPI Communicators
- **Cycle detection**
  - Find deadlocks



# Tricky memory issues



# Memory Analysis

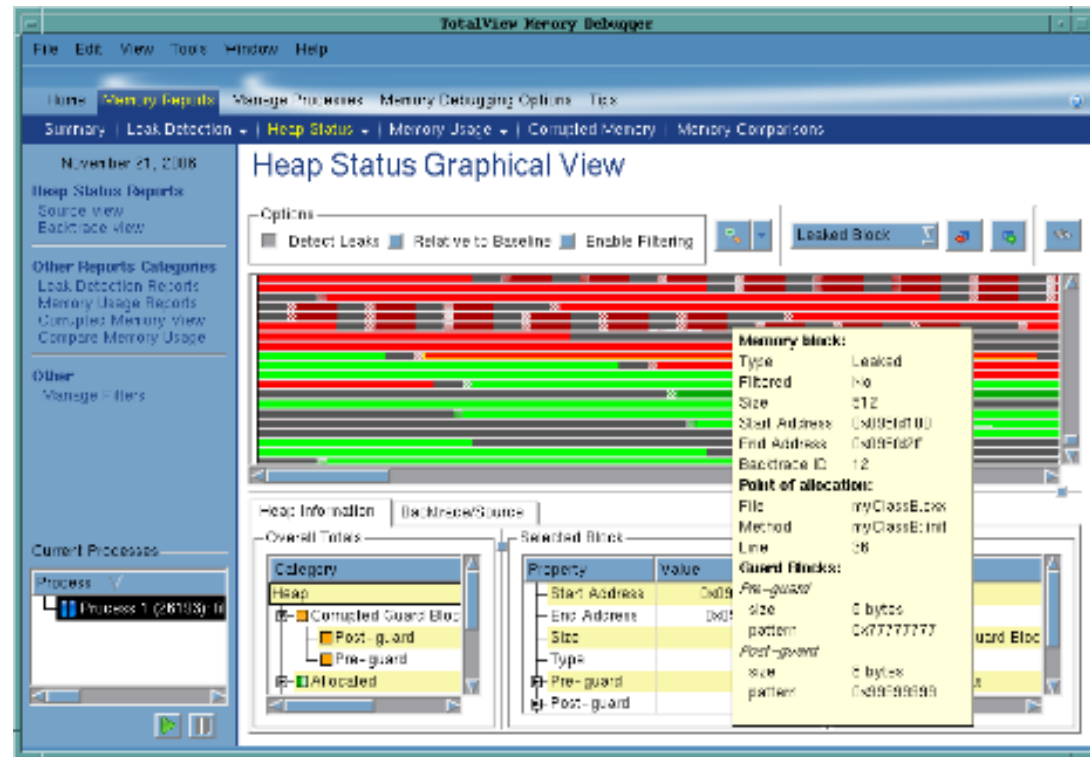
Eliminate memory errors and slowdown hackers from using them

## Runtime Memory Analysis : Eliminate Memory Errors

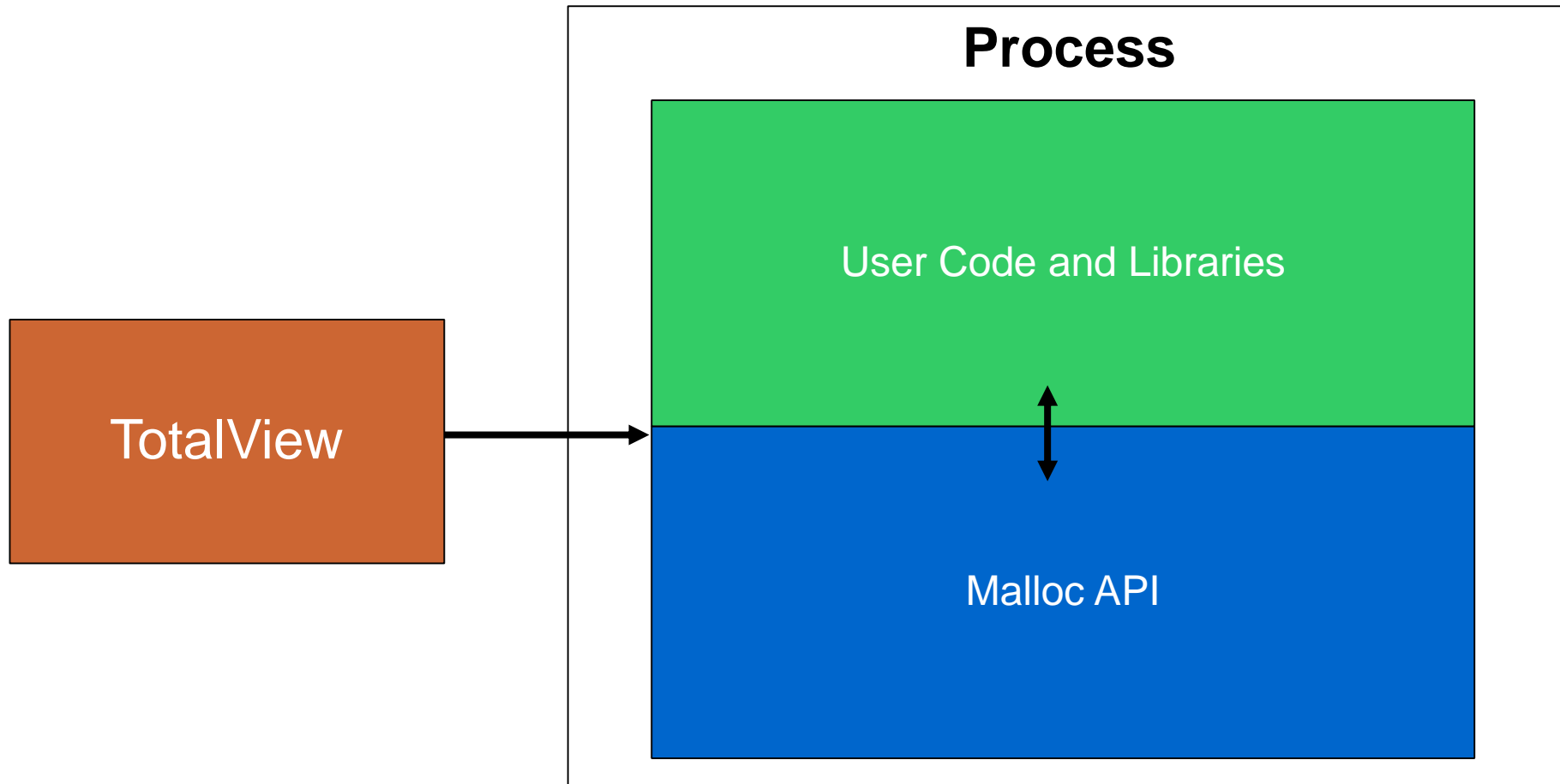
- Detects memory leaks *before* they are a problem
- Explore heap memory usage

## Features

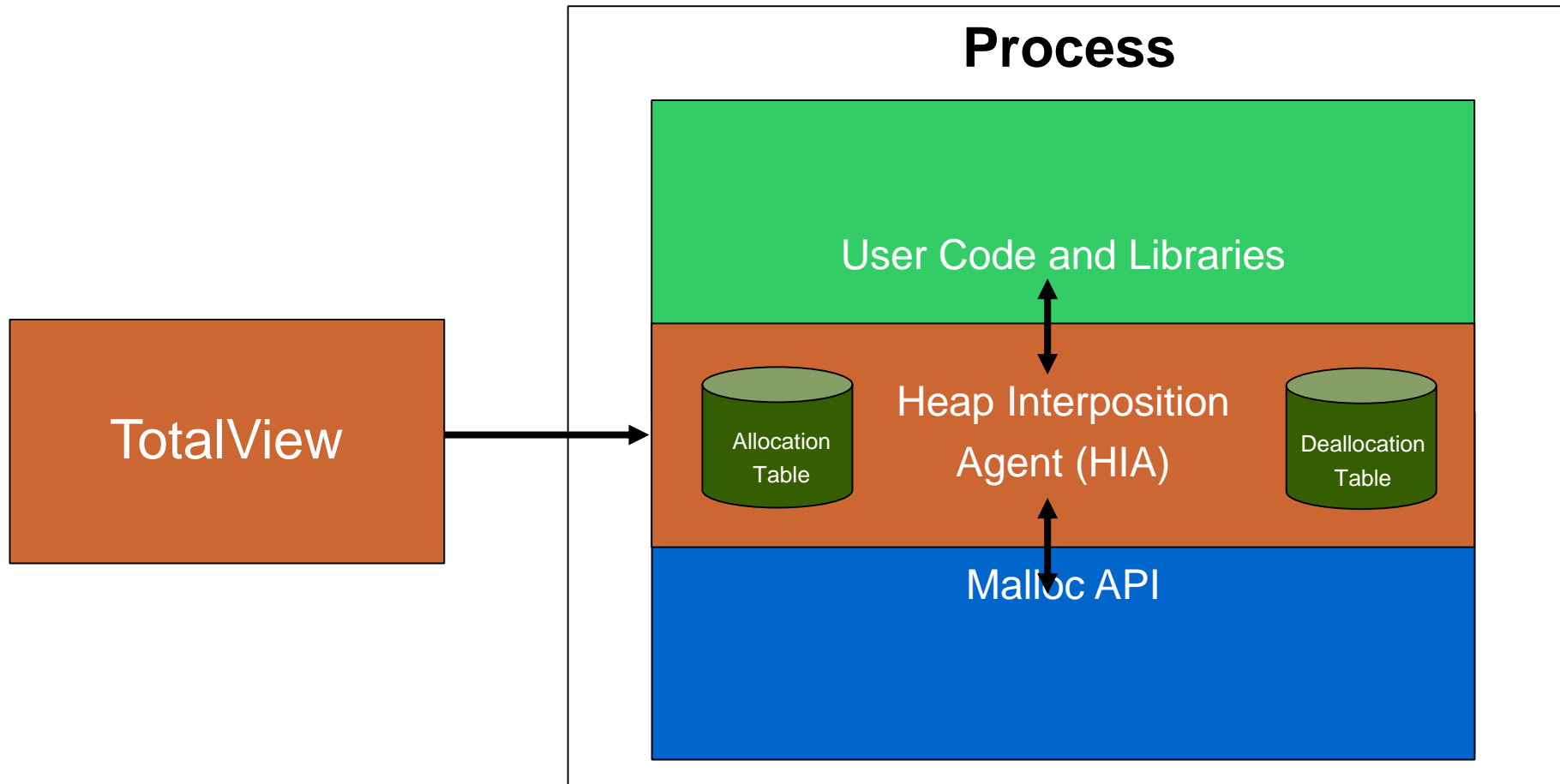
- Detects
  - Malloc API misuse
  - Memory leaks
  - Buffer overflows
- Low runtime overhead
- Easy to use
  - Works with vendor libraries
  - No recompilation
  - No instrumentation



# The Agent and Interposition



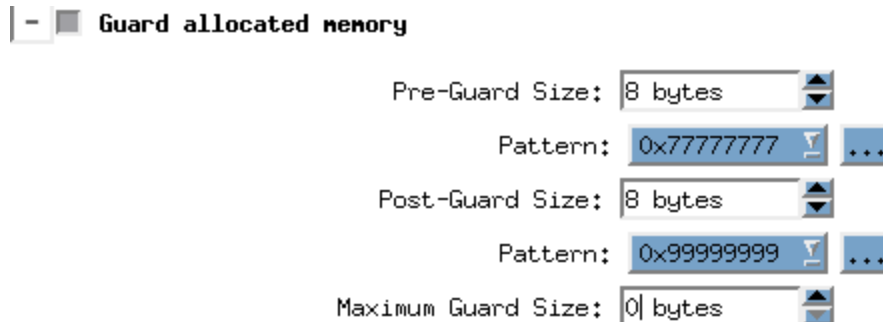
# The Agent and Interposition



# Guard Blocks

## Guard allocated memory

When selected, the Memory Debugger writes guard blocks before and after a memory block that your program allocates



### Pre-Guard and Post-Guard Size:

Sets the size in bytes of the block that the Memory Debugger places immediately before and after the memory block that your program allocates

### Pattern:

Indicates the pattern that the Memory Debugger writes into guard blocks. The default values are 0x77777777 and 0x99999999

# Red Zones

Red Zones : instant array bounds detection

- Red Zones provides:
  - Both read and write memory access violations.
  - Immediate detection of memory overruns.
  - Detection of access violations both before and after the bounds of allocated memory.
  - Detection of deallocated memory accesses.
- Red Zones events
  - MemoryScape will stop your programs execution and raise an event alerting you to the illegal access. You will be able to see exactly where your code over stepped the bounds.

# Heap Graphical View

MemoryScape 3.2.3-0

File Tools Window Help

Home **Memory Reports** Manage Processes Memory Debugging Options Tips

Summary | Leak Detection | **Heap Status** | Memory Usage | Corrupted Memory | Memory Comparisons

May 11, 2012

**Save Data**  
Export Memory Data

**Heap Status Reports**  
Source Report  
Backtrace Report

**Other Reports Category**  
Leak Detection Report  
Memory Usage Report  
Corrupted Memory Report  
Compare Memory Usage

**Other Tasks**  
Manage Filters

Process Selection

Process ▾

Parallel Job filter

▾ MPI\_COMM\_WORLD

▸ filterapp-mpi

⚠ filterapp-mpi

▸ filterapp-mpi

▸ filterapp-mpi

**Heap Status Graphical Report**

Options

☐ Detect Leaks ☐ Relative to Baseline ☐ Enable Filtering

Leaked Block

Process 1: filterapp-mpi.1

0x0949d058 - 0x094d2c00 (214.91KB)

Heap Information | Backtrace/Source | Memory Content

Overall Totals

Category	Bytes
Heap	
▣ Allocated	81.55KB
▣ Deallocated	129.88KB
▣ Hoarded	0
▣ Leaked	Unknown
▣ Red Zones	0

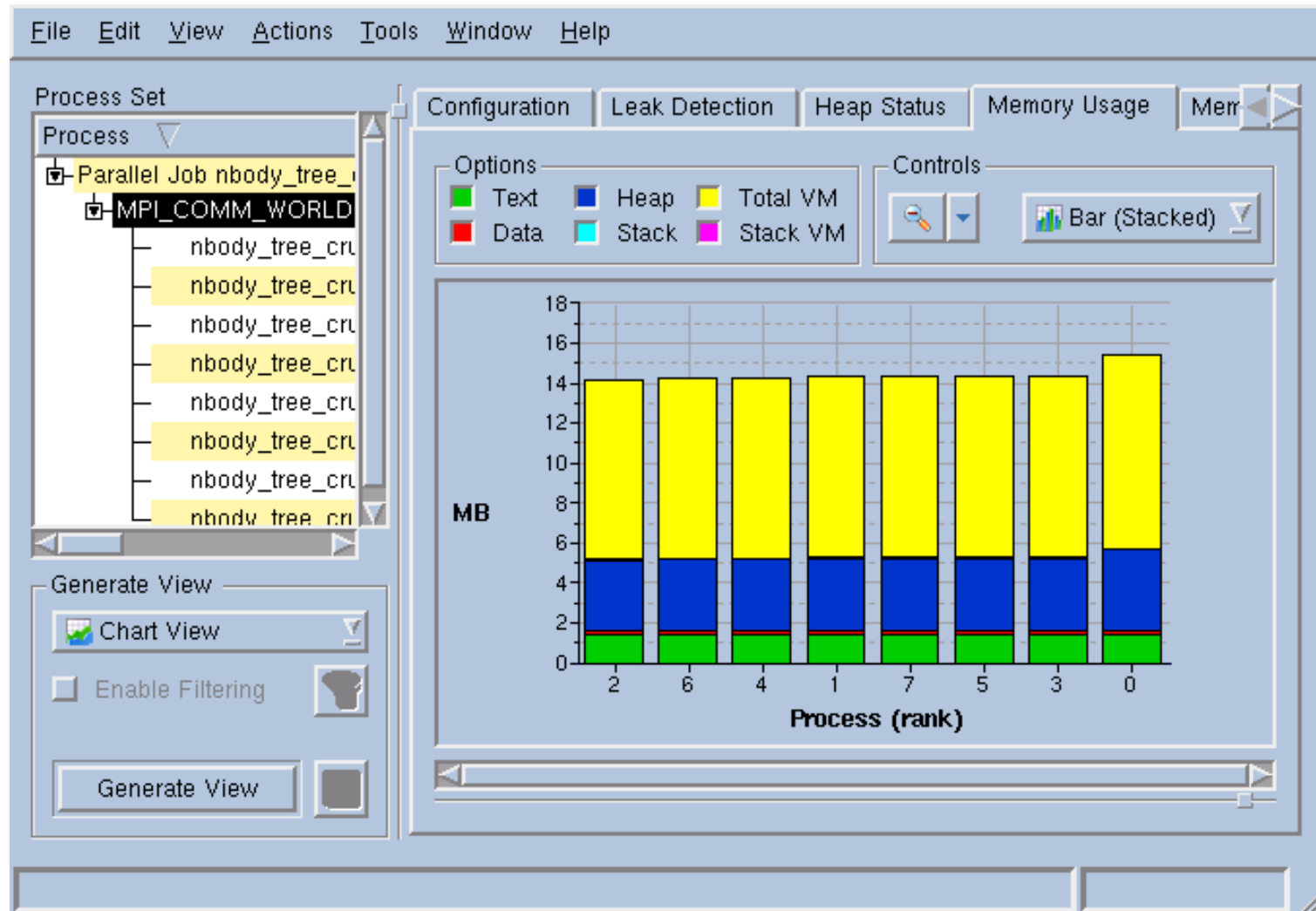
Selected Block

Property	Value
Start Address	0x0949d098
End Address	0x0949d0bb
Size	36
Type	Allocated
Filtered	No
Backtrace ID	3
Allocator	C
Owner	C

Related Blocks

Category
Backtrace ID 3
▣ Allocated
▣ Corrupted Guard Block
▣ Deallocated
▣ Guard Blocks
▣ Hoarded
▣ Leaked
▣ Red Zones

# Memory Usage Statistics



# Memory Comparisons

- “Diff” live processes
  - Compare processes across cluster
- Compare with baseline
  - See changes between point A and point B
- Compare with saved session
  - Provides memory usage change from last run

The screenshot displays the MemoryScape 3.2.3-0 application window. The main menu includes File, Tools, Window, and Help. The toolbar contains various icons for navigation and analysis. The status bar at the top right indicates '1 New Event'. The main window is titled 'Memory Comparison Report' and shows a comparison between two sessions: Session 1 (filterapp-mpi.0) and Session 2 (filterapp-mpi.1). The report is dated May 11, 2012.

The report includes a table with the following columns: Process, Bytes Session 2, Bytes Session 1, Bytes Difference, and Count Session 2. The data is as follows:

Process	Bytes Session 2	Bytes Session 1	Bytes Difference	Count Session 2
filterapp-mpi.0/filterap...	69.36KB	194	69.17KB	149
filterapp-mpi	69.36KB	194	69.17KB	149
myClassB.cxx	66.01KB	0	66.01KB	131
myClassA.cxx	2.00KB	0	2.00KB	4
new_allocator.h	1024	0	1024	1
main.cxx	192	0	192	7
stl_algobase.h	162	194	-32	6

The interface also includes a 'Process Selection' panel on the left, showing a tree view of processes. The bottom of the window has tabs for 'Session 1 Source' and 'Session 2 Source'.



**Batch debugging can help**

# Regression Testing

- How do you make sure a bug you fixed never returns?
  - Build a regression test
  - Issue is it typically is time consuming
- What is the method to build a regression test?
  - Use the tools that helped you find it
- How do you run a regression test?
  - Invoke it during your build process
- Enter TotalView scripts
  - Command line driven
  - Access to application internals
  - Same commands as in the debugger

```
•!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!
•! Print
•!
•! Process:
•!   ./server (Debugger Process
ID: 1, System ID: 12110)
•! Thread:
•!   Debugger ID: 1.1, System
ID: 3083946656
•! Time Stamp:
•!   06-26-2008 14:04:09
•! Triggered from event:
•!   actionpoint
•! Results:
•!   foreign_addr = {
•!       sin_family = 0x0002 (2)
•!       sin_port = 0x1fb6 (8118)
•!       sin_addr = {
•!           s_addr = 0x6658a8c0
(1717086400)
•!       }
•!       sin_zero = ""
•!   }
•!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!
```

# TVScript Overview

---

- Gives you non-interactive access to TotalView's capabilities
- Useful for
  - Debugging in batch environments
  - Watching for intermittent faults
  - Parametric studies
  - Automated testing and validation
- TVScript is a script (not a scripting language)
  - It runs your program to completion and performs debugger actions on it as you request
  - Results are written to an output file
  - No GUI
  - No interactive command line prompt

# TVScript Syntax

- tvscript syntax:
  - tvscript [ options ] [ filename ] [ -a program\_args ]
- Options express (“event”, “action”) pairs
  - Typical events
    - Action\_point
    - Any\_memory\_event
    - Guard\_corruption
    - error
  - Typical actions
    - Display\_backtrace
    - List\_leaks
    - Save\_memory
    - Print [-slice {slice\_exp} {variable | exp}]
- Example

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
.! Print
.!
.! Process:
.! ./server (Debugger Process ID: 1, System ID: 12110)
.! Thread:
.! Debugger ID: 1.1, System ID: 3083946656
.! Time Stamp:
.! 06-26-2008 14:04:09
.! Triggered from event:
.! actionpoint
.! Results:
.! foreign_addr = {
.!     sin_family = 0x0002 (2)
.!     sin_port = 0x1fb6 (8118)
.!     sin_addr = {
.!         s_addr = 0x6658a8c0 (1717086400)
.!     }
.!     sin_zero = ""
.! }
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
-create_actionpoint "#85=>print foreign_addr"
```

# New TotalView UI

# TotalView new UI

The screenshot displays the TotalView new UI interface for debugging a C++ program. The window title is "combined - Process 1, No current thread - TotalView for HPC 2016". The menu bar includes File, Edit, Group, Process, Thread, Action Points, Debug, Window, and Help. The toolbar contains various controls for group control, execution (play, pause, stop, step over, step into, step out), and the ReplayEngine.

**Processes & Threads**

Description	# P	# T	Members
combined (S3)	1	1	p1
Nonexis...	1	1	p1

**Code Editor** (combined.cxx)

```
16 using namespace std;
17 static int ARRAYSIZE = 20;
18
19 long bb;
20 char * str;
21 long cc;
22
23
24 int do_parallel = 1;
25
26 int main(int argc, char **argv)
27 {
28     str = (char *) malloc(100);
29     strcpy( str, "Hello World" );
30
31     // General features
32     arrays();
33     diveinall();
34     printf( "%s\n", str );
35
36     long arr_longs[7] = {1,2,3,4,5,16,32};
37     pthreads_loop();
38
39     // C++ features
40     derived_class();
41     stl_view();
42     test_templates();
43
44 }
```

**Call Stack**

No current thread

**Action Points**

ID	Type	Stop	File	Line
1	BP	Process	combined.cxx	717
2	BP	Process	combined.cxx	670
3	BP	Process	combined.cxx	608

**Data View**

Name	Type	Value
[Add New Expression]		

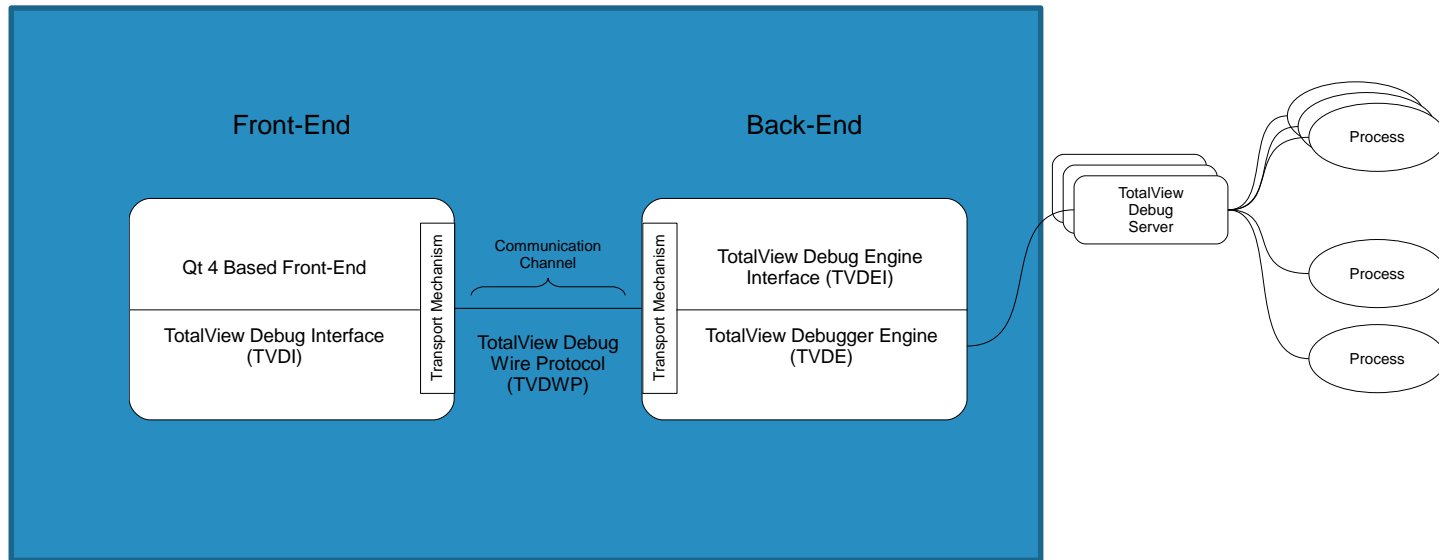
Process: combined (1) No current thread

dstewart@localhost:~ combined - Process 1, No current...

1 / 4

# New Architecture in building UI

Used to be one large application



# Why does the architecture matter

---

- This isn't short term thinking
  - No need for XWindows on target platforms
  - Performance
  - Scalability
  - Platforms
- More effective debugger on all platforms
- Easier 3rd party integrations



**Q&A**

# Thanks!

---

- Visit the website
  - <http://www.roguewave.com/products/totalview.aspx>
  - Videos
  - Documentation
  - Sign up for an evaluation
  - Contact customer support
  - Post on the user forum