TotalView Training

Dean Stewart

Rogue Wave Software

Cray XE6 Performance Workshop
July 12th, 2012

Agenda

- Introduction
- Startup
- Remote Display Debugging
- UI Navigation and Process Control
- Action Points
- Data Monitoring and Visualization
- Debugging for Parallel Applications

Agenda

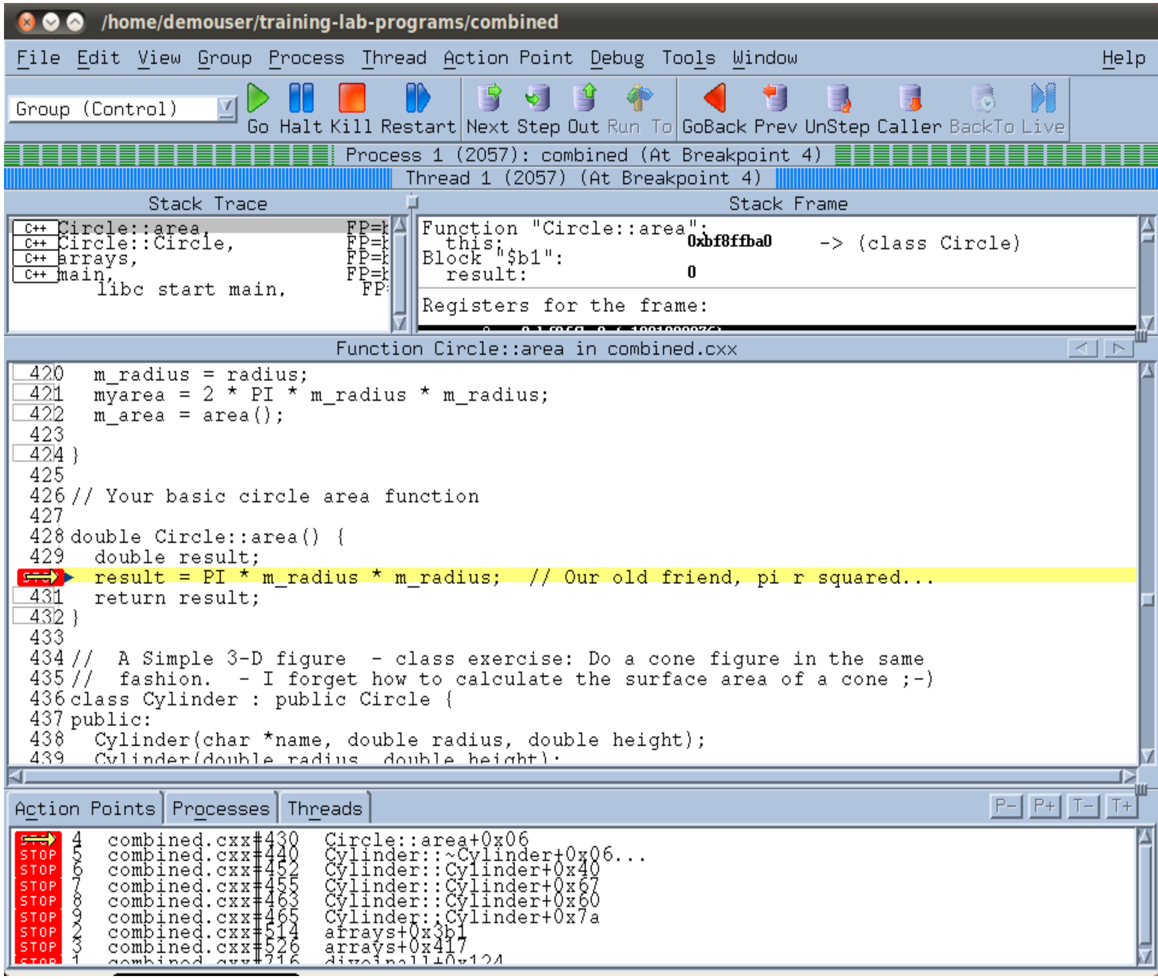
- Memory Debugging with MemoryScape
- Batch Debugging
- Reverse Debugging with ReplayEngine
- What’s New in TotalView 8.10
- Support and Questions

INTRODUCTION

What is TotalView?

A comprehensive debugging solution for demanding parallel and multi-core applications

- Wide compiler & platform support
  - C, C++, Fortran 77 & 90, UPC
  - Unix, Linux, OS X
- Handles Concurrency
  - Multi-threaded Debugging
  - Parallel Debugging
    - MPI, PVM, Others
  - Remote and Client/Server Debugging
- Integrated Memory Debugging
- Reverse Debugging
  - ReplayEngine
- Supports a Variety of Usage Models
  - Powerful and Easy GUI
    - Visualization
  - CLI for Scripting
  - Long Distance Remote Debugging
  - Unattended Batch Debugging



Process 1 (2057): combined (At Breakpoint 4)

Thread 1 (2057) (At Breakpoint 4)

```
C++ Circle::area,            FP=
C++ Circle::Circle,          FP=
C++ arrays,                  FP=
C++ main,                    FP=
       libc start main,      FP
```

```
Function "Circle::area":
  this:              0xbf8ffba0      -> (class Circle)
Block "$b1":
  result:            0
Registers for the frame:
```

Function Circle::area in combined.cxx

```
  m_radius = radius;
  myarea = 2 * PI * m_radius * m_radius;
  m_area = area();
}

// Your basic circle area function

double Circle::area() {
  double result;
  result = PI * m_radius * m_radius;   // Our old friend, pi r squared...
  return result;
}

//  A Simple 3-D figure  - class exercise: Do a cone figure in the same
//  fashion.  - I forget how to calculate the surface area of a cone ;-)
class Cylinder : public Circle {
public:
  Cylinder(char *name, double radius, double height);
  Cylinder(double radius, double height);
```

```
     4  combined.cxx#430  Circle::area+0x06
STOP 5  combined.cxx#440  Cylinder::~Cylinder+0x06...
STOP 6  combined.cxx#452  Cylinder::Cylinder+0x40
STOP 7  combined.cxx#455  Cylinder::Cylinder+0x67
STOP 8  combined.cxx#463  Cylinder::Cylinder+0x60
STOP 9  combined.cxx#465  Cylinder::Cylinder+0x7a
STOP 2  combined.cxx#514  arrays+0x3b1
STOP 3  combined.cxx#526  arrays+0x417
STOP 1  combined.cxx#716
```

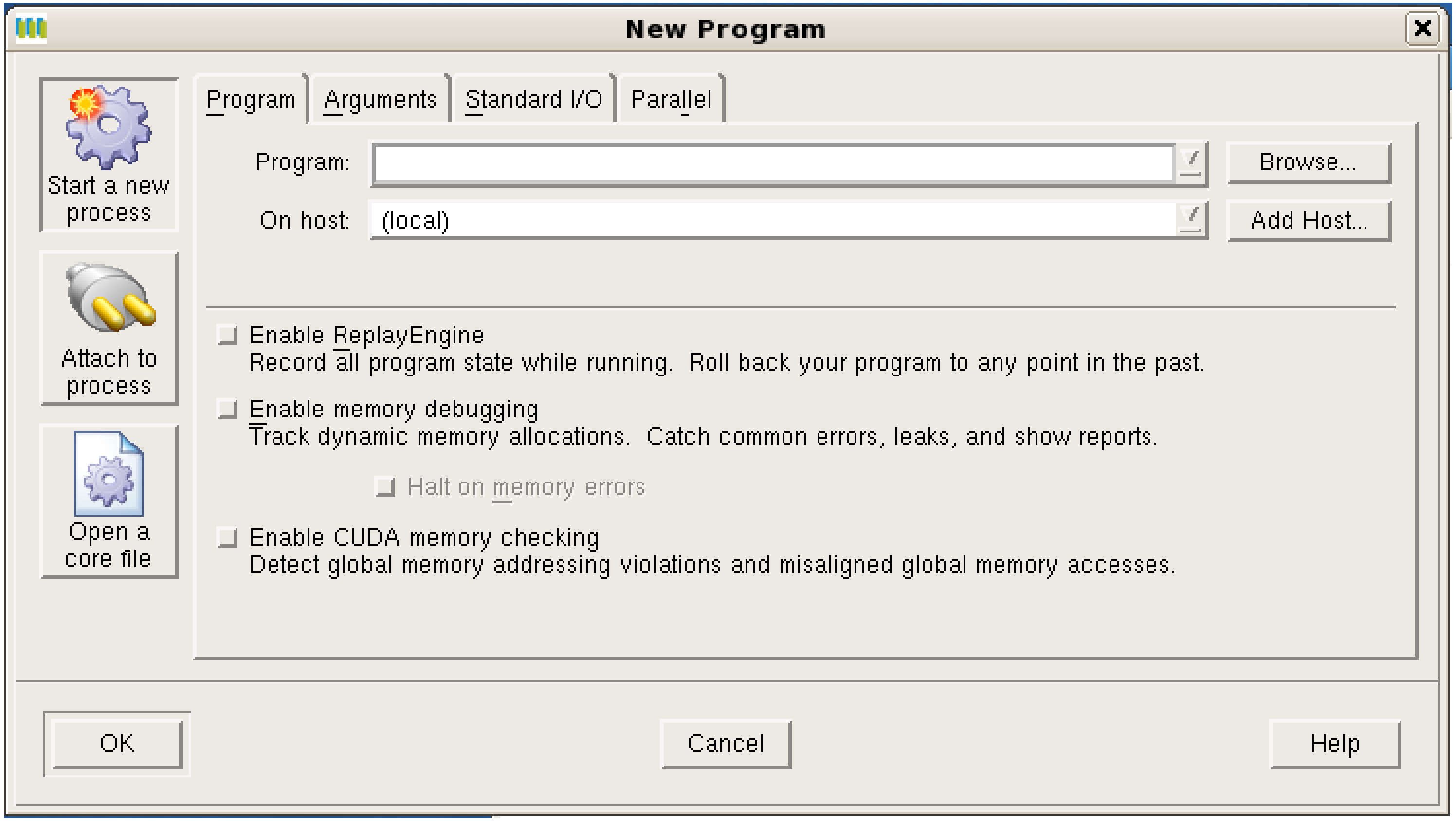
Supported Compilers and Architectures

- **Platform Support**
  - Linux x86, x86-64, ia64, Power
  - Mac Intel
  - Solaris Sparc and AMD64
  - AIX
  - Cray XT, XE, XK
  - IBM BGL, BGP
  - Cell
- **Languages / Compilers**
  - C/C++, Fortran, UPC, Assembly
  - Many Commercial & Open Source Compilers
- **Parallel Environments**
  - MPI
    - MPICH1& 2, Open MPI, Intel MPI, SGI MPT & Propack, SLURM, poe, MPT, Quadrics, MVAPICH1 & 2, Bullx MPI, & many others)
  - UPC

STARTUP

Starting TotalView

Start New Process



New Program

Program:

On host: (local)

□ Enable ReplayEngine
Record all program state while running. Roll back your program to any point in the past.

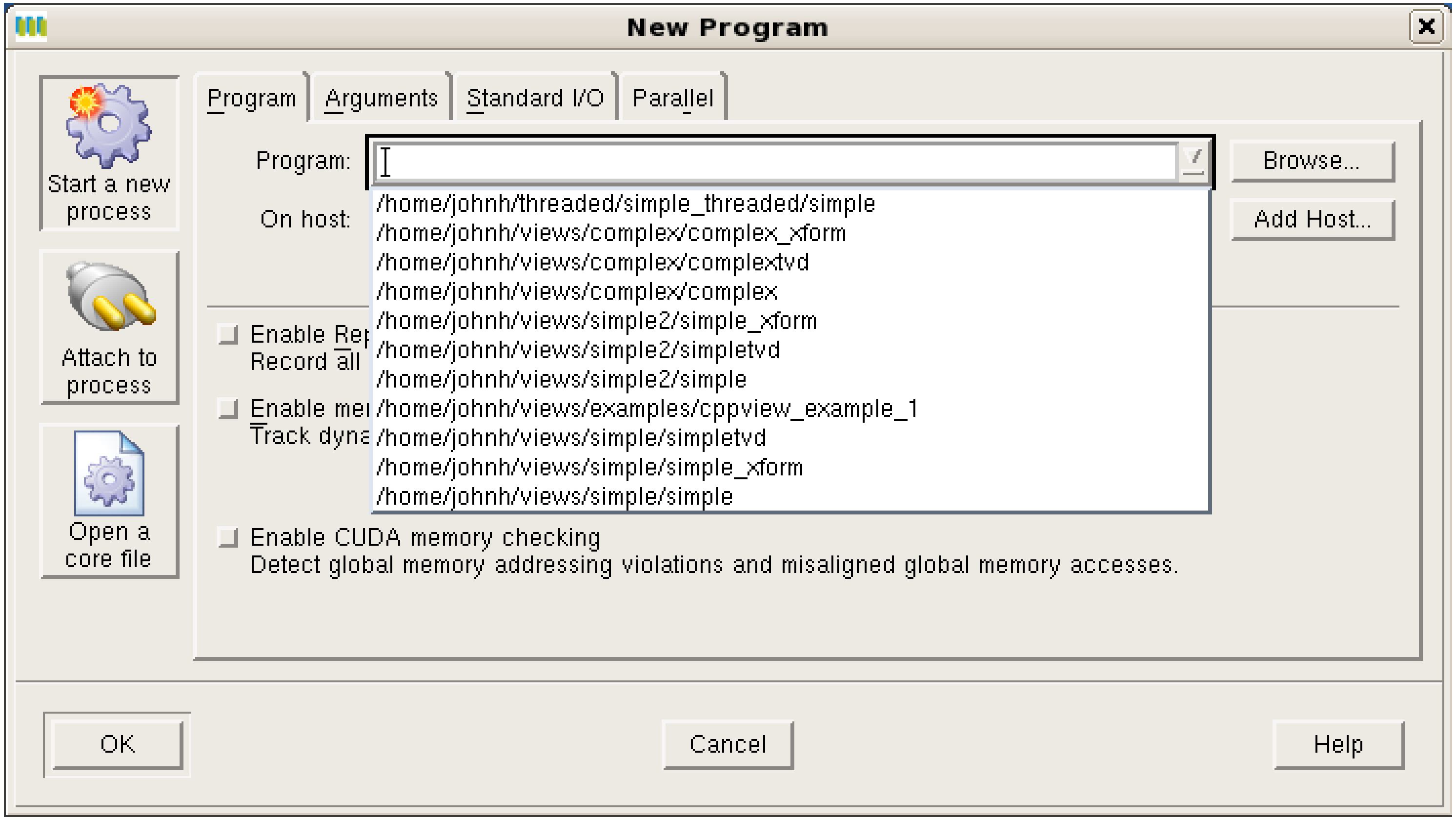
□ Enable memory debugging
Track dynamic memory allocations. Catch common errors, leaks, and show reports.

□ Halt on memory errors

□ Enable CUDA memory checking
Detect global memory addressing violations and misaligned global memory accesses.

Starting TotalView

Start New Process – Select a recent process



New Program

Program:

On host:

/home/johnh/threaded/simple_threaded/simple
/home/johnh/views/complex/complex_xform
/home/johnh/views/complex/complextvd
/home/johnh/views/complex/complex
/home/johnh/views/simple2/simple_xform
/home/johnh/views/simple2/simpletvd
/home/johnh/views/simple2/simple
/home/johnh/views/examples/cppview_example_1
/home/johnh/views/simple/simpletvd
/home/johnh/views/simple/simple_xform
/home/johnh/views/simple/simple

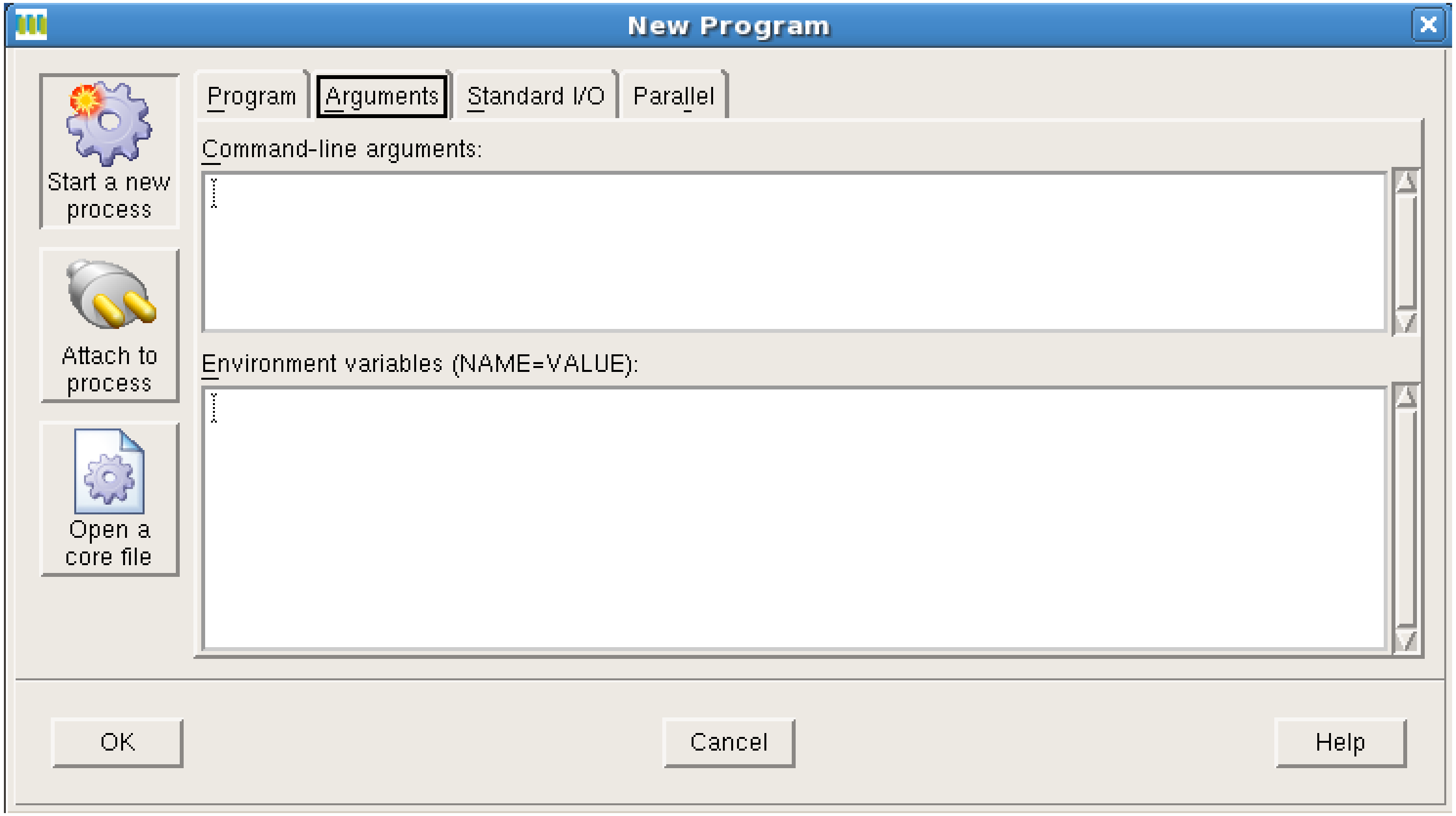
□ Enable Rep
Record all

□ Enable me
Track dyna

□ Enable CUDA memory checking
Detect global memory addressing violations and misaligned global memory accesses.

Starting TotalView

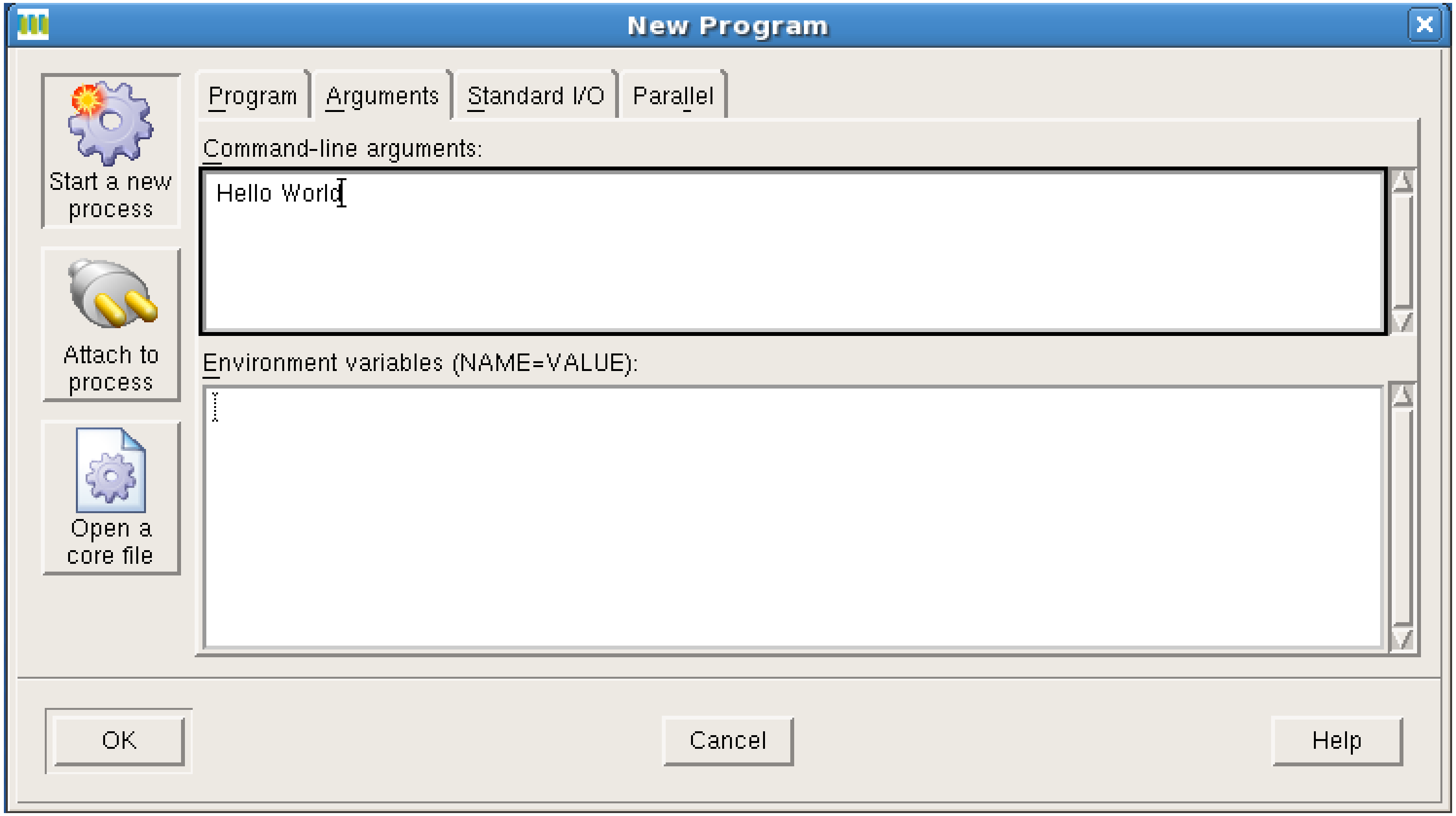
Start New Process – Arguments tab



Command-line arguments:

Environment variables (NAME=VALUE):

Start New Process – Command-line Args

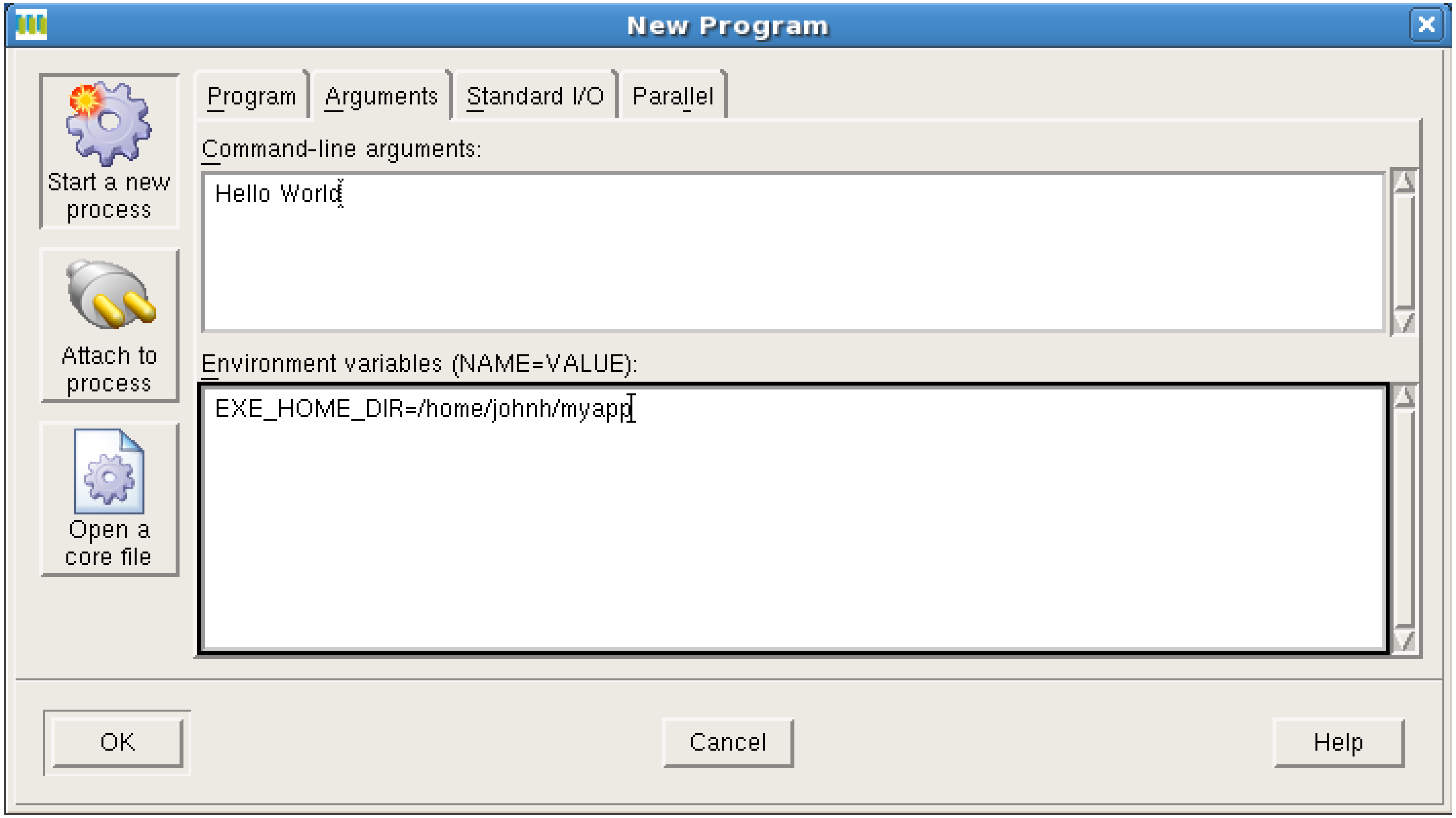


Command-line arguments:

Hello World

Environment variables (NAME=VALUE):

Start New Process – set environment variables



Command-line arguments:

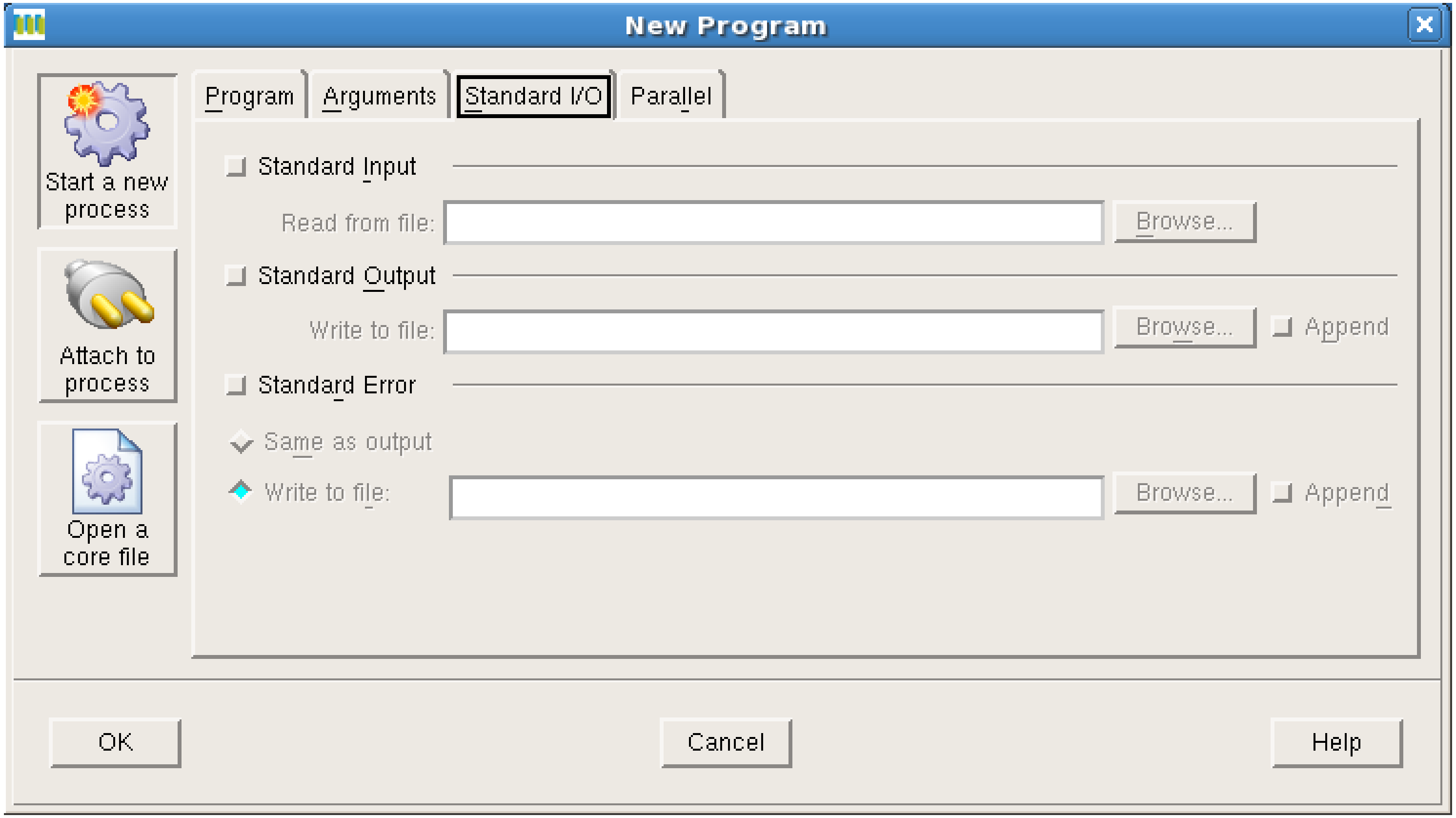
Hello World

Environment variables (NAME=VALUE):

EXE_HOME_DIR=/home/johnh/myapp

Starting TotalView

Start New Process – Standard I/O redirection



New Program

□ Standard Input

Read from file:

□ Standard Output

Write to file:

□ Append

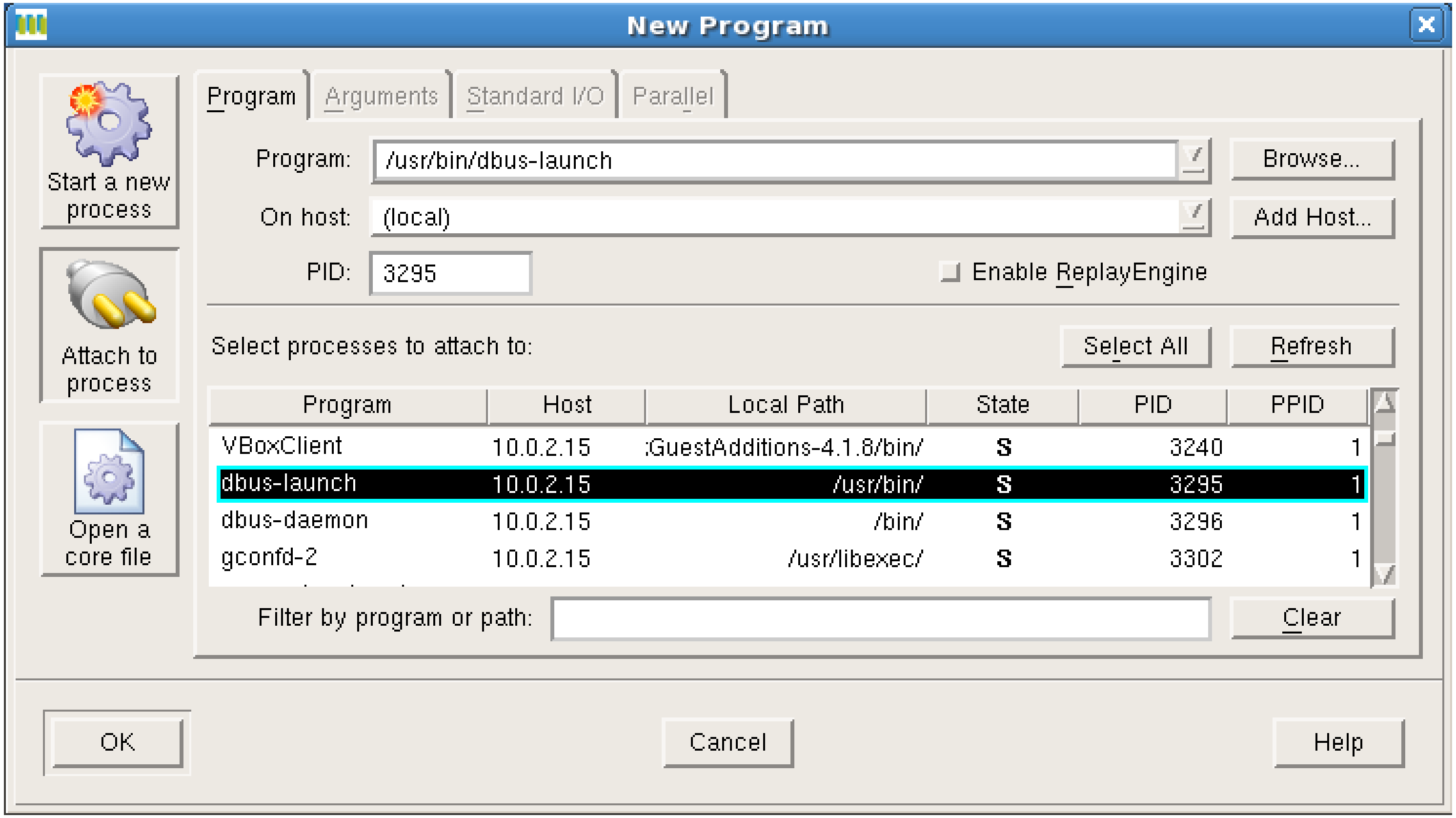
□ Standard Error

◇ Same as output

◆ Write to file:

□ Append

Attach to Process



Program: /usr/bin/dbus-launch

On host: (local)

PID: 3295

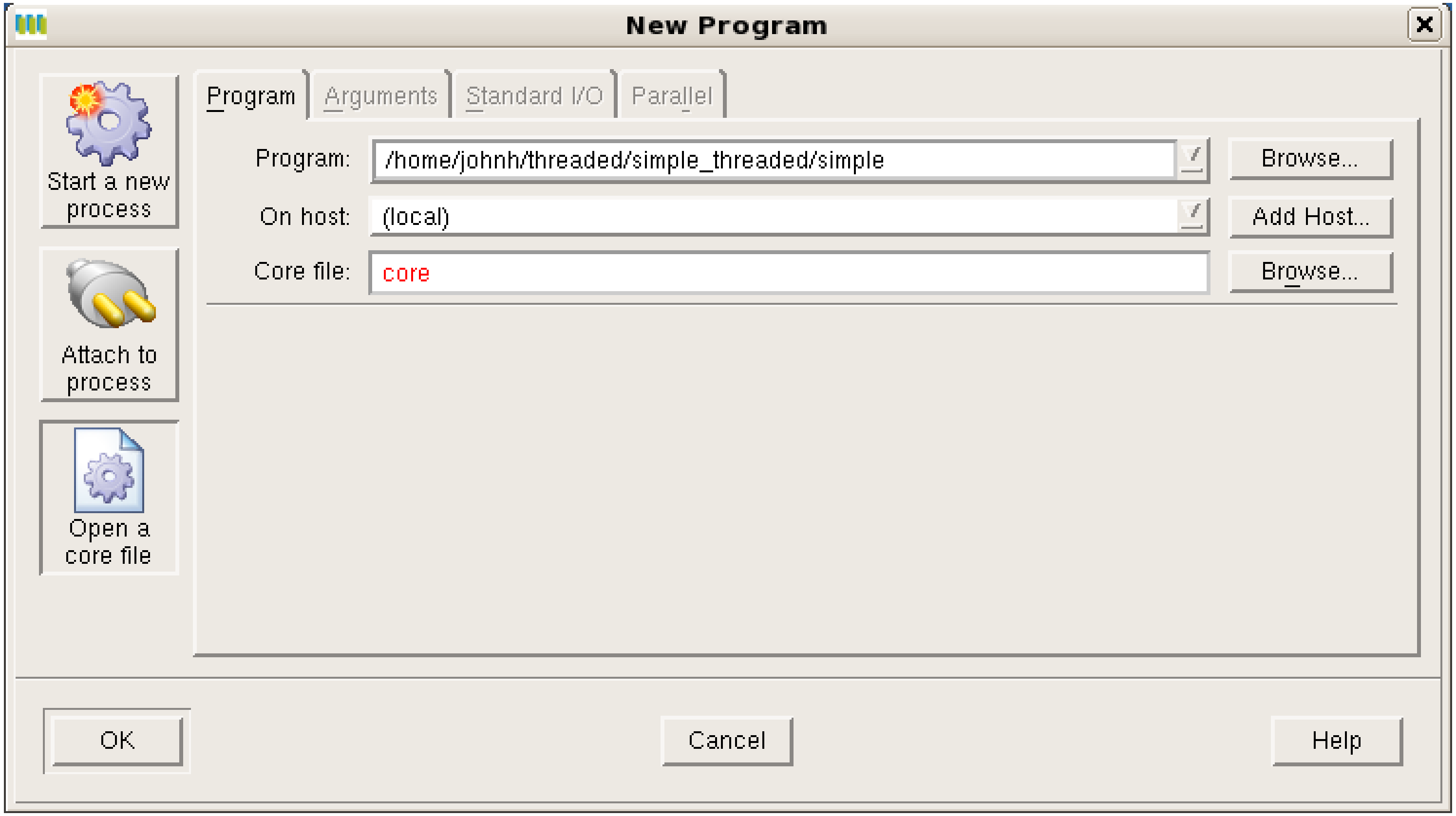
□ Enable ReplayEngine

Select processes to attach to:

Program	Host	Local Path	State	PID	PPID
VBoxClient	10.0.2.15	:GuestAdditions-4.1.8/bin/	S	3240	1
dbus-launch	10.0.2.15	/usr/bin/	S	3295	1
dbus-daemon	10.0.2.15	/bin/	S	3296	1
gconfd-2	10.0.2.15	/usr/libexec/	S	3302	1

Filter by program or path:

Open a Core File



Program: /home/johnh/threaded/simple_threaded/simple

On host: (local)

Core file: core

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 ]
```

REMOTE DISPLAY DEBUGGING

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

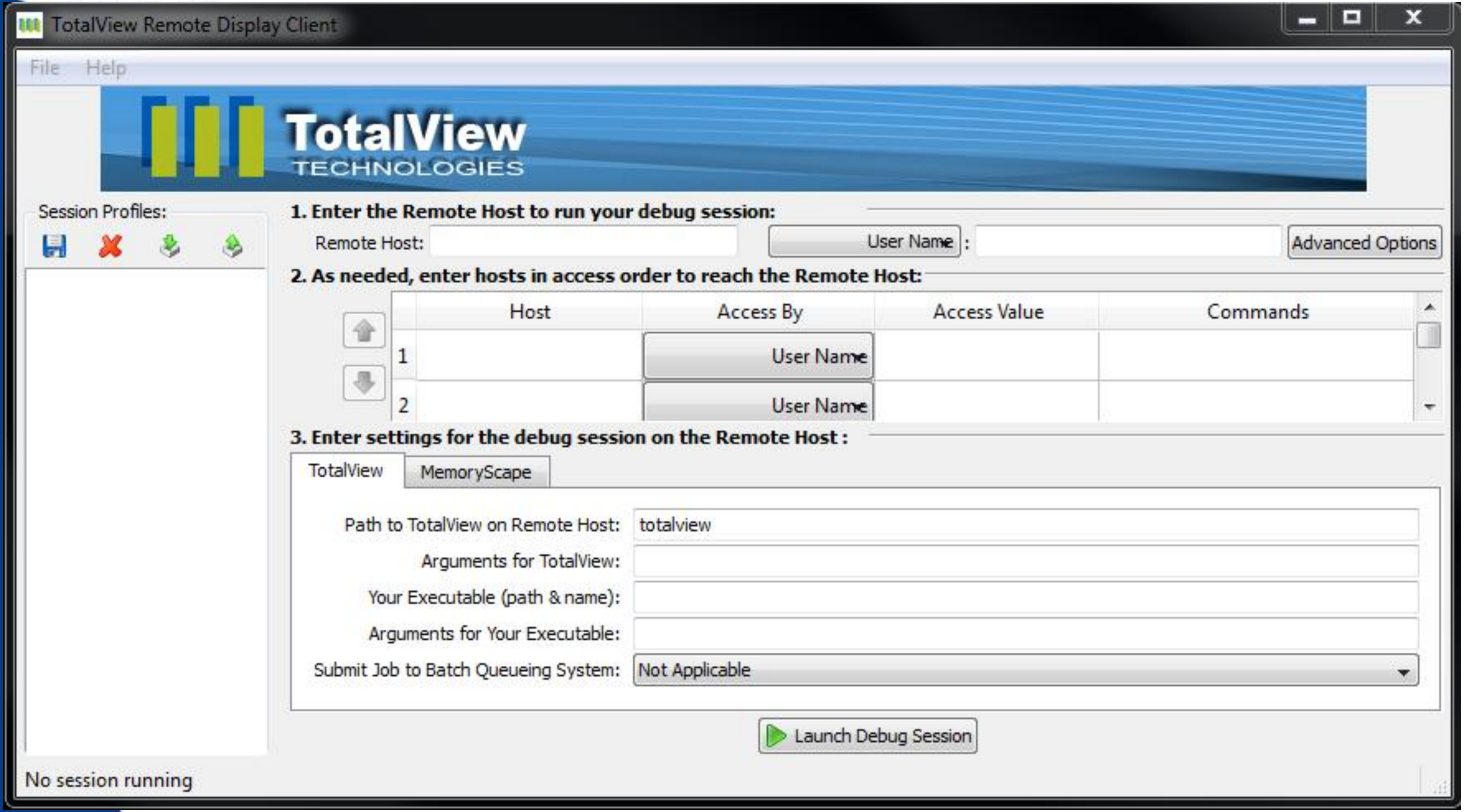
Remote Display Client

- **Free to install on as many clients as needed**
- **No license required to run the client**
  - Only the server running TotalView requires licenses. Must be version 8.6 or later of TotalView or version 2.4 or later of MemoryScape.
- **Presents a local window that displays TotalView or MemoryScape running on the remote machine**
- **Requires SSH and X Windows on Server**

Remote Display Client

- **User must provide information necessary to connect to remote host**
- **Connection info can be saved for reuse**
- **Information required includes:**
  - User name, public key file, other ssh information
  - Directory where TotalView/MemoryScape is located
  - Path and name of executable to be debugged
  - If using indirect connection with host jump, each host
    - Host name
    - Access type (User name, public key, other ssh information)
    - Access value
- **Client also allows for batch submission via PBS Pro or LoadLeveler**

Remote Display Client



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:

	Host	Access By	Access Value	Commands
1		User Name		
2		User Name		

3. Enter settings for the debug session on the Remote Host :

Path to TotalView on Remote Host: totalview

Arguments for TotalView:

Your Executable (path & name):

Arguments for Your Executable:

Submit Job to Batch Queueing System: Not Applicable

No session running

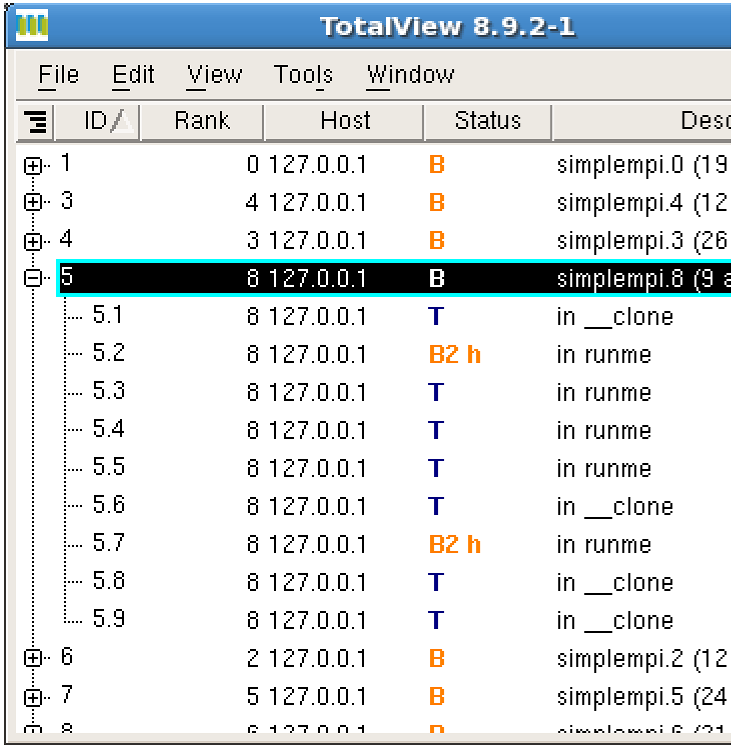
Session Profile Management

- Connection information can be saved as a profile, including all host jumping information
- Multiple profiles can be generated
- Profiles can be exported and shared
- Generated profiles can be imported for use by other users

UI NAVIGATION AND PROCESS CONTROL

Root Window

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

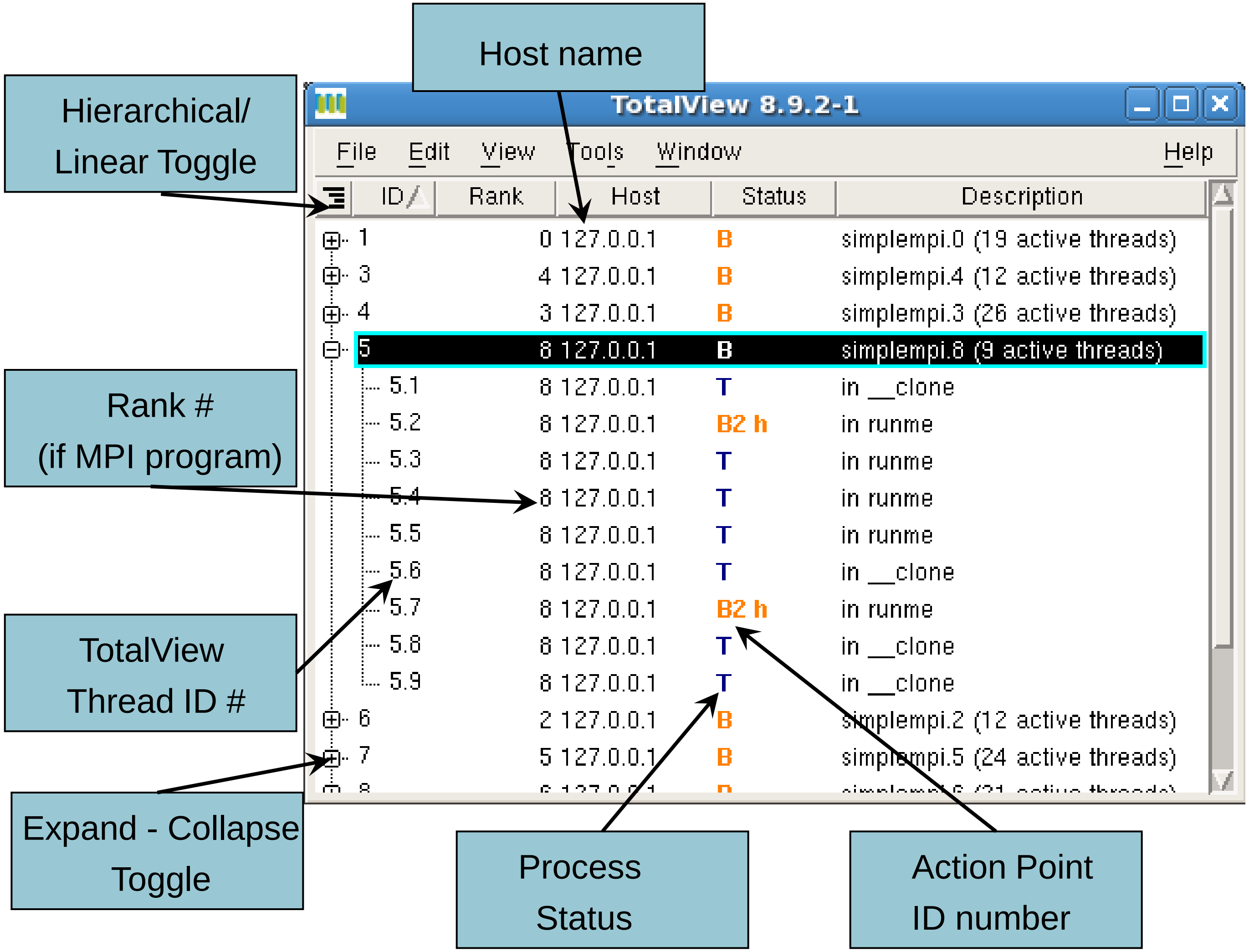


ID	Rank	Host	Status	Desc
1	0	127.0.0.1	B	simplempi.0 (19
3	4	127.0.0.1	B	simplempi.4 (12
4	3	127.0.0.1	B	simplempi.3 (26
5	8	127.0.0.1	B	simplempi.8 (9 a
5.1	8	127.0.0.1	T	in __clone
5.2	8	127.0.0.1	B2 h	in runme
5.3	8	127.0.0.1	T	in runme
5.4	8	127.0.0.1	T	in runme
5.5	8	127.0.0.1	T	in runme
5.6	8	127.0.0.1	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



Host name

Hierarchical/ Linear Toggle

Rank # (if MPI program)

TotalView Thread ID #

Expand - Collapse Toggle

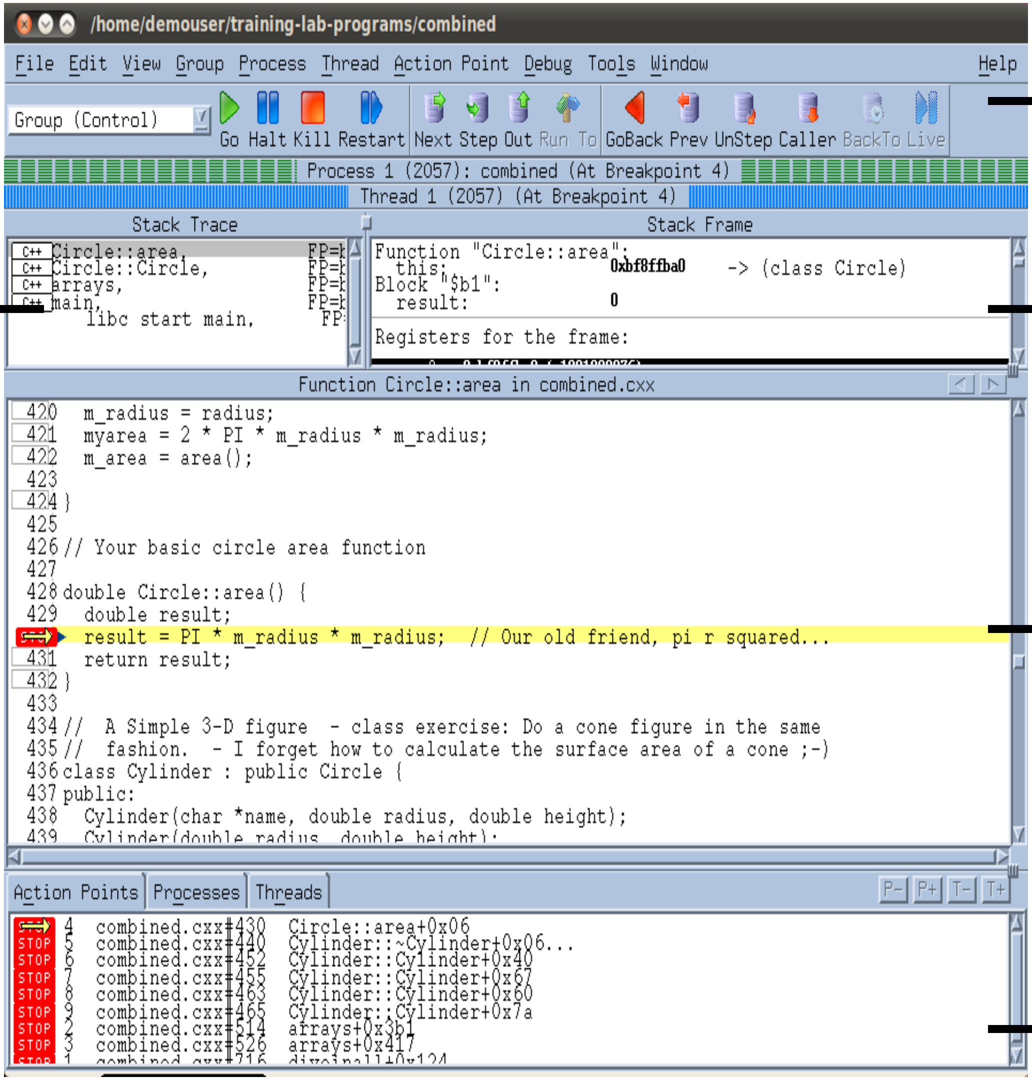
Process Status

Action Point ID number

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


File Edit View Group Process Thread Action Point Debug Tools Window Help
Group (Control)
Go Halt Kill Restart Next Step Out Run To GoBack Prev UnStep Caller BackTo Live
Stack Trace
Stack Frame
Action Points Processes Threads
P- P+ T- T+

/home/demouser/training-lab-programs/combined

Process 1 (2057): combined (At Breakpoint 4)

Thread 1 (2057) (At Breakpoint 4)

```
C++ Circle::area,          FP=b
C++ Circle::Circle,        FP=b
C++ arrays,                FP=b
C++ main,                  FP=b
      libc start main,      FP:
```

```
Function "Circle::area":
  this:          0xbf8ffba0     -> (class Circle)
Block "$b1":
  result:        0

Registers for the frame:
```

Function Circle::area in combined.cxx

```
  m_radius = radius;
  myarea = 2 * PI * m_radius * m_radius;
  m_area = area();

}

// Your basic circle area function

double Circle::area() {
  double result;
  result = PI * m_radius * m_radius;   // Our old friend, pi r squared...
  return result;
}

//  A Simple 3-D figure  - class exercise: Do a cone figure in the same
//  fashion.  - I forget how to calculate the surface area of a cone ;-)
class Cylinder : public Circle {
public:
  Cylinder(char *name, double radius, double height);
  Cylinder(double radius, double height);
```

```
     4  combined.cxx#430  Circle::area+0x06
STOP 5  combined.cxx#440  Cylinder::~Cylinder+0x06...
STOP 6  combined.cxx#452  Cylinder::Cylinder+0x40
STOP 7  combined.cxx#455  Cylinder::Cylinder+0x67
STOP 8  combined.cxx#463  Cylinder::Cylinder+0x60
STOP 9  combined.cxx#465  Cylinder::Cylinder+0x7a
STOP 2  combined.cxx#514  arrays+0x3b1
STOP 3  combined.cxx#526  arrays+0x417
STOP 1  combined.cxx#716  divoinall+0x124
```

Toolbar

Stack Trace Pane

Stack Frame Pane

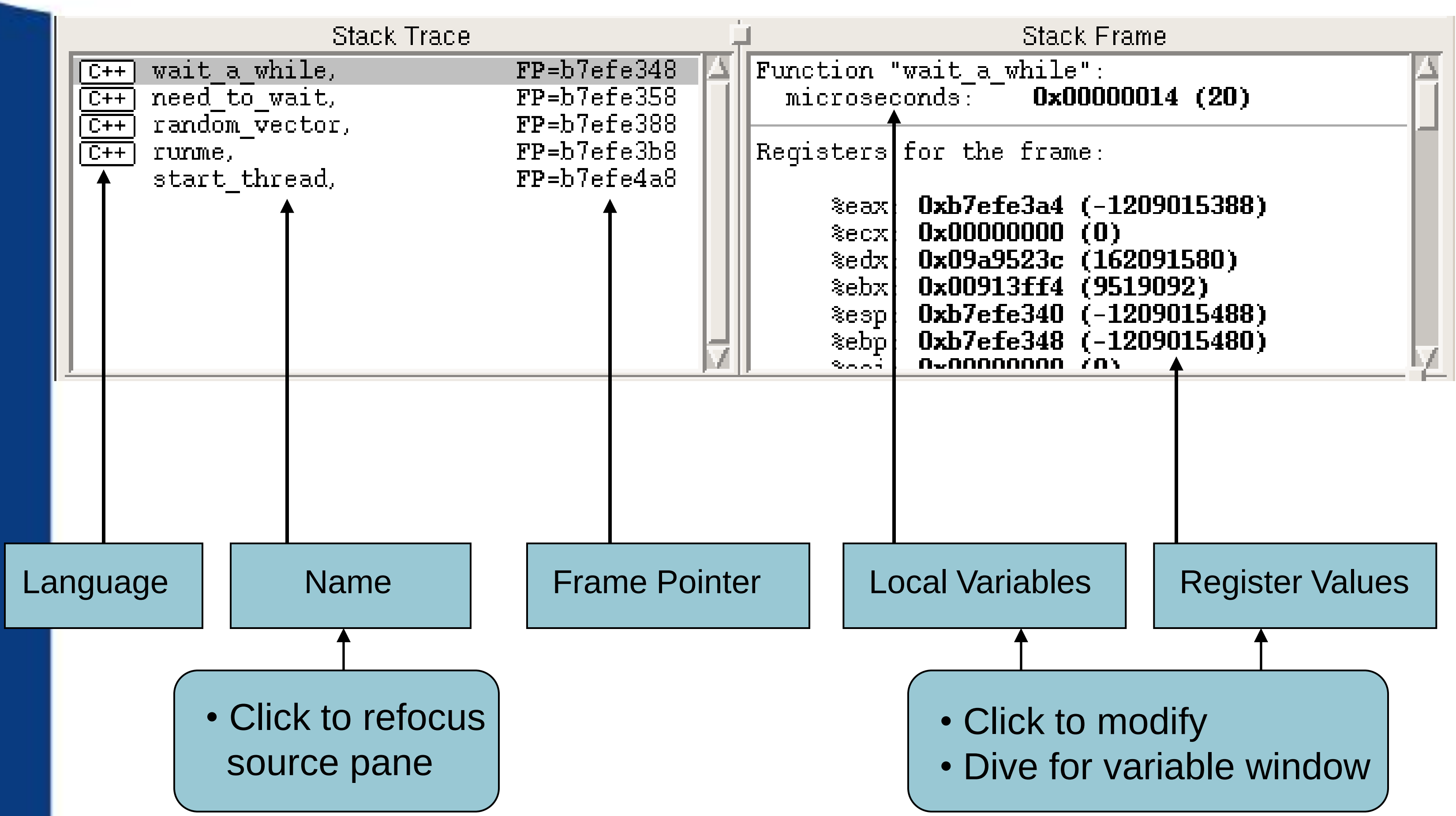
Source Pane

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



```
Stack Trace
C++  wait_a_while,          FP=b7efe348
C++  need_to_wait,          FP=b7efe358
C++  random_vector,         FP=b7efe388
C++  runme,                 FP=b7efe3b8
     start_thread,          FP=b7efe4a8
```

```
Stack Frame
Function "wait_a_while":
  microseconds:      0x00000014 (20)

Registers for the frame:

      %eax: 0xb7efe3a4 (-1209015388)
      %ecx: 0x00000000 (0)
      %edx: 0x09a9523c (162091580)
      %ebx: 0x00913ff4 (9519092)
      %esp: 0xb7efe340 (-1209015488)
      %ebp: 0xb7efe348 (-1209015480)
      %esi: 0x00000000 (0)
```

Language

Name

Frame Pointer

Local Variables

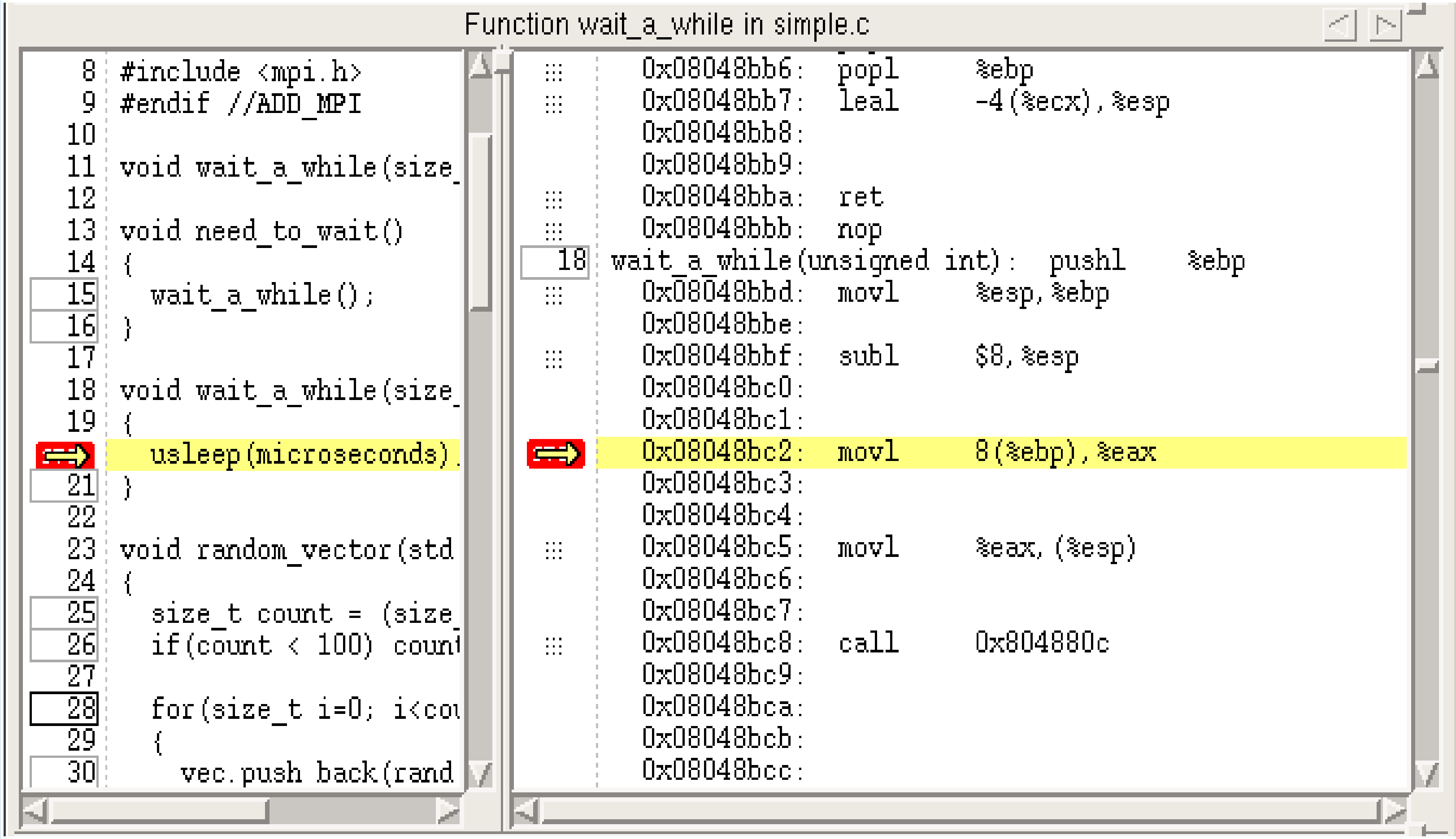
Register Values

- Click to refocus source pane

- Click to modify
- Dive for variable window

Source Code Pane

View as Source - or Assembly - or Both!

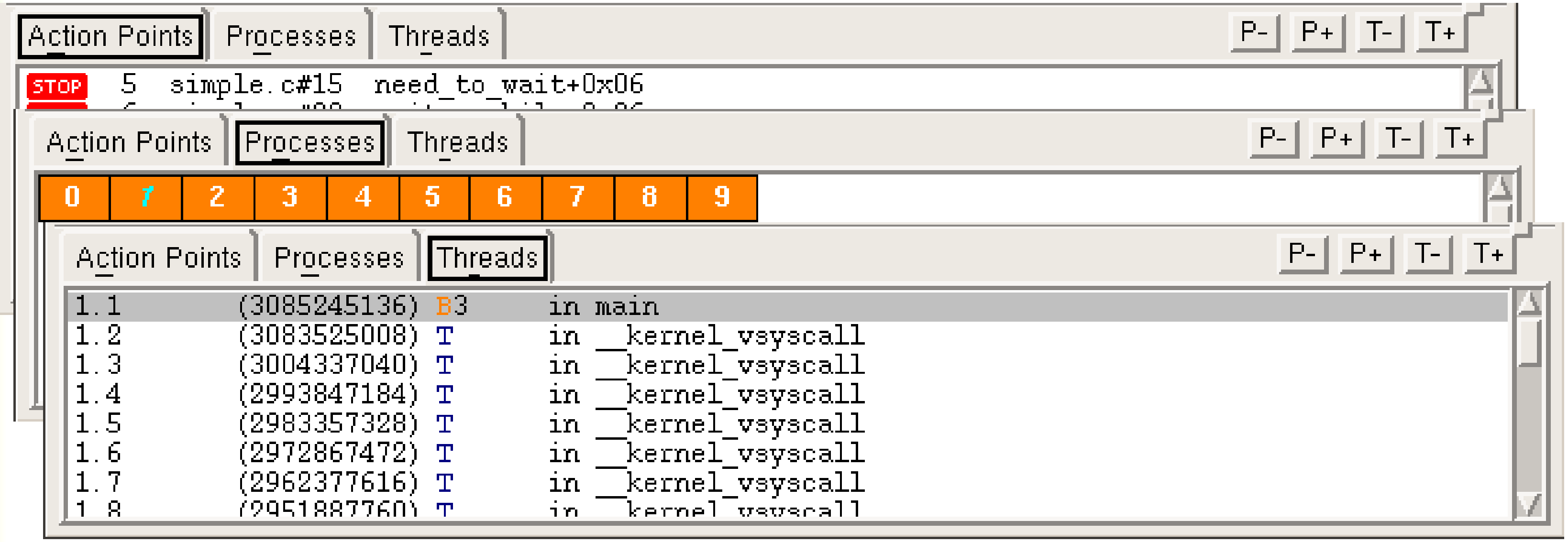


Function wait_a_while in simple.c

```
#include <mpi.h>
#endif //ADD_MPI

void wait_a_while(size_

void need_to_wait()
{
  wait_a_while();
}

void wait_a_while(size_
{
  usleep(microseconds)
}

void random_vector(std
{
  size_t count = (size_
  if(count < 100) count

  for(size_t i=0; i<cou
  {
    vec.push_back(rand
```

```
    0x08048bb6:  popl      %ebp
    0x08048bb7:  leal      -4(%ecx),%esp
    0x08048bb8:
    0x08048bb9:
    0x08048bba:  ret
    0x08048bbb:  nop
18  wait_a_while(unsigned int):  pushl    %ebp
    0x08048bbd:  movl      %esp,%ebp
    0x08048bbe:
    0x08048bbf:  subl      $8,%esp
    0x08048bc0:
    0x08048bc1:
    0x08048bc2:  movl      8(%ebp),%eax
    0x08048bc3:
    0x08048bc4:
    0x08048bc5:  movl      %eax,(%esp)
    0x08048bc6:
    0x08048bc7:
    0x08048bc8:  call      0x804880c
    0x08048bc9:
    0x08048bca:
    0x08048bcb:
    0x08048bcc:
```

Tabbed Pane



```
STOP  5  simple.c#15  need_to_wait+0x06
```

```
0  1  2  3  4  5  6  7  8  9
```

```
1.1        (3085245136) B3      in main
1.2        (3083525008) T       in __kernel_vsyscall
1.3        (3004337040) T       in __kernel_vsyscall
1.4        (2993847184) T       in __kernel_vsyscall
1.5        (2983357328) T       in __kernel_vsyscall
1.6        (2972867472) T       in __kernel_vsyscall
1.7        (2962377616) T       in __kernel_vsyscall
1.8        (2951887760) T       in __kernel_vsyscall
```

Action Points Tab

all currently defined action points

Processes Tab

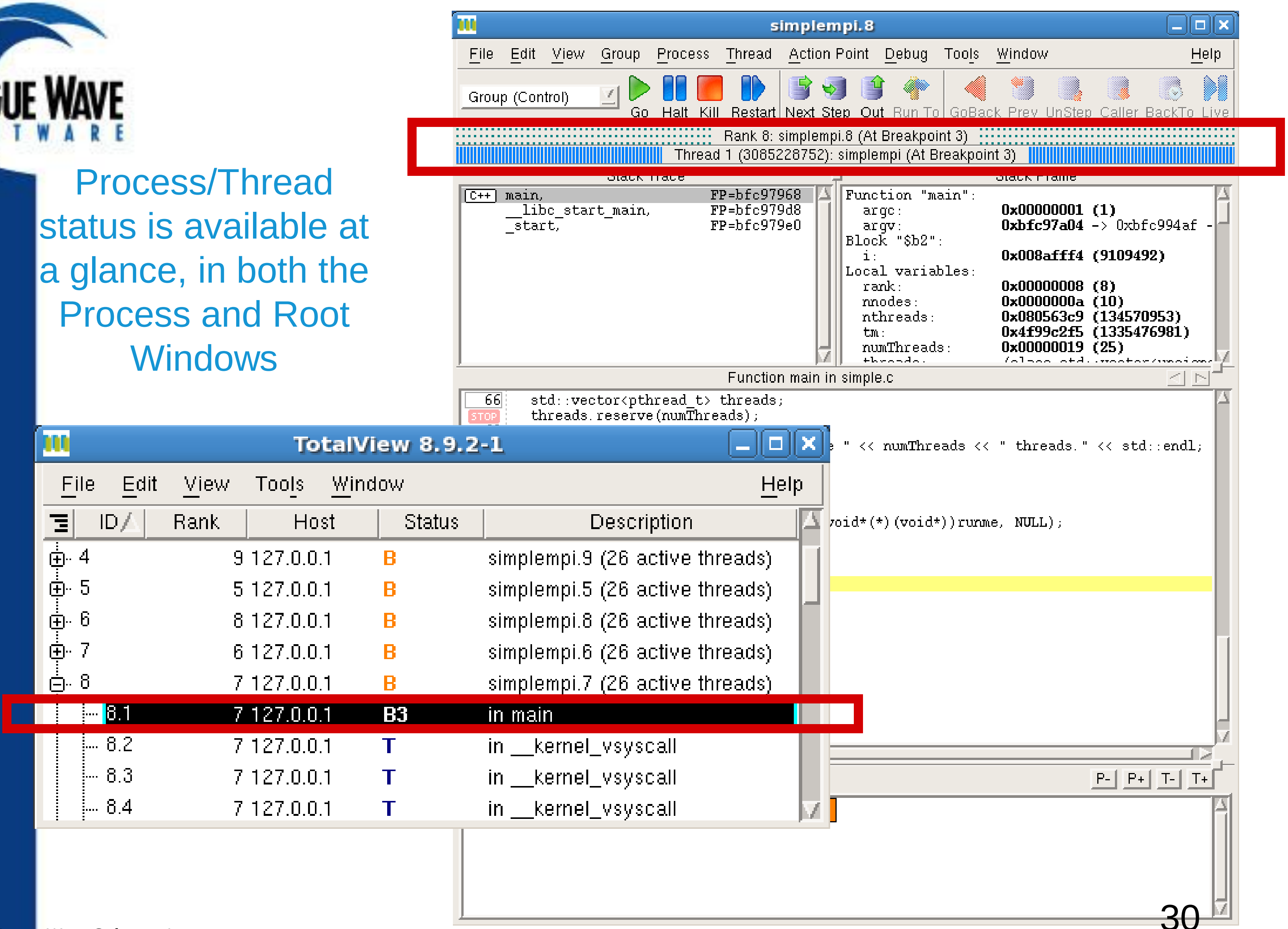
all current processes

Threads Tab:

all current threads, ID's, Status

Process Status

Process/Thread status is available at a glance, in both the Process and Root Windows


simplempi.8
File Edit View Group Process Thread Action Point Debug Tools Window Help
Group (Control)
Go Halt Kill Restart Next Step Out Run To GoBack Prev UnStep Caller BackTo Live
Stack Trace
Stack Frame
P- P+ T- T+
TotalView 8.9.2-1
File Edit View Tools Window Help

Rank 8: simplempi.8 (At Breakpoint 3)

Thread 1 (3085228752): simplempi (At Breakpoint 3)

```
C++  main,                FP=bfc97968
     __libc_start_main,   FP=bfc979d8
     _start,              FP=bfc979e0
```

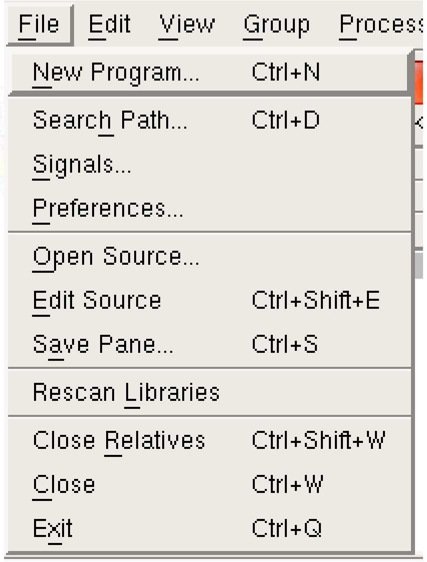
```
Function "main":
  argc:                0x00000001 (1)
  argv:                0xbfc97a04 -> 0xbfc994af -
Block "$b2":
  i:                   0x008afff4 (9109492)
Local variables:
  rank:                0x00000008 (8)
  nnodes:              0x0000000a (10)
  nthreads:            0x080563c9 (134570953)
  tm:                  0x4f99c2f5 (1335476981)
  numThreads:          0x00000019 (25)
```

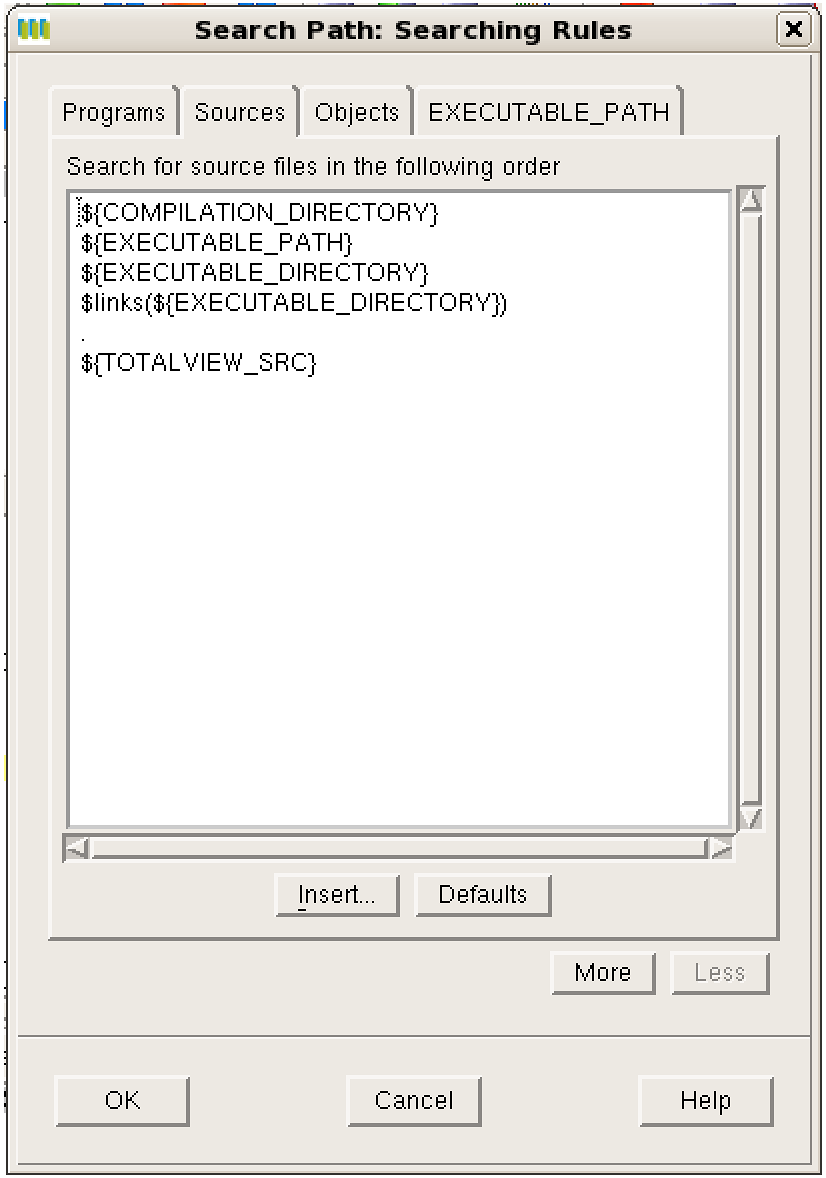
Function main in simple.c

```
 66      std::vector<pthread_t> threads;
STOP     threads.reserve(numThreads);
         e " << numThreads << " threads." << std::endl;
         void*(*)(void*))runme, NULL);
```

ID	Rank	Host	Status	Description
4	9	127.0.0.1	B	simplempi.9 (26 active threads)
5	5	127.0.0.1	B	simplempi.5 (26 active threads)
6	8	127.0.0.1	B	simplempi.8 (26 active threads)
7	6	127.0.0.1	B	simplempi.6 (26 active threads)
8	7	127.0.0.1	B	simplempi.7 (26 active threads)
8.1	7	127.0.0.1	B3	in main
8.2	7	127.0.0.1	T	in __kernel_vsyscall
8.3	7	127.0.0.1	T	in __kernel_vsyscall
8.4	7	127.0.0.1	T	in __kernel_vsyscall

Search Paths

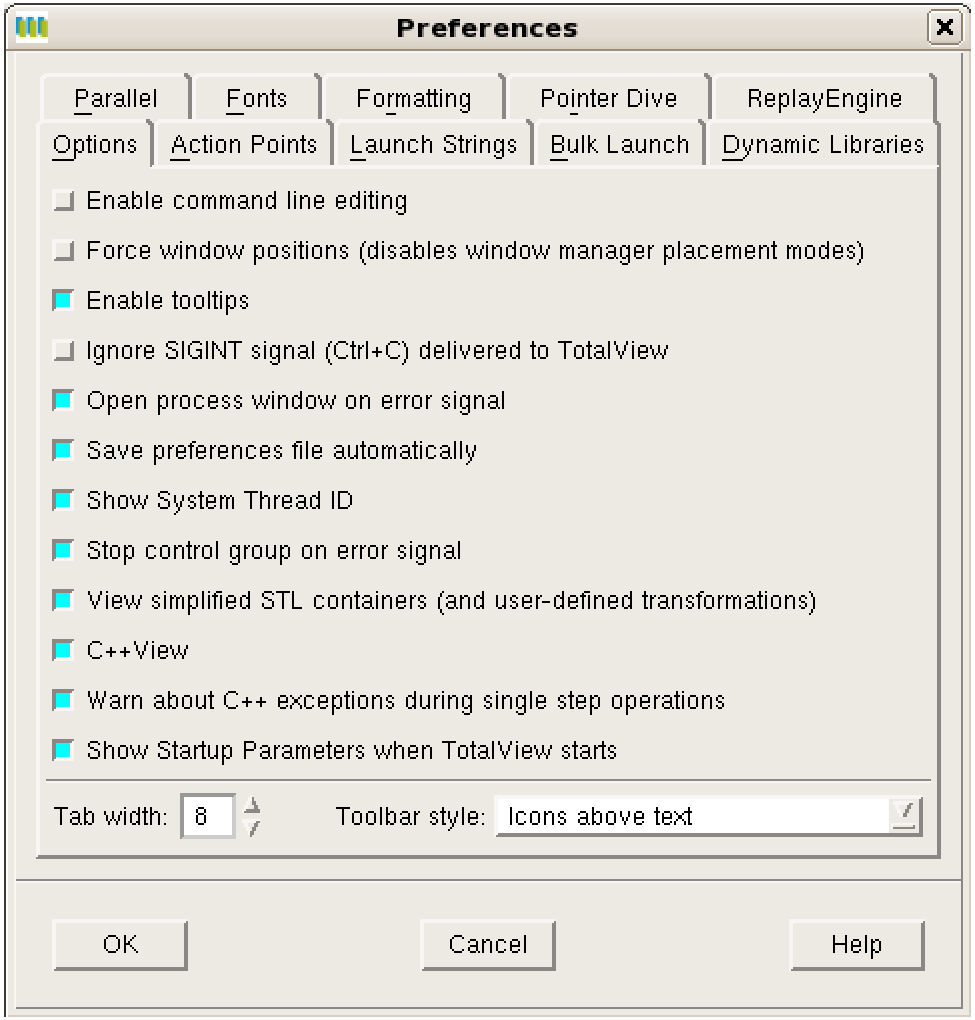




Search for source files in the following order

```
${COMPILATION_DIRECTORY}
${EXECUTABLE_PATH}
${EXECUTABLE_DIRECTORY}
$links(${EXECUTABLE_DIRECTORY})
.
${TOTALVIEW_SRC}
```

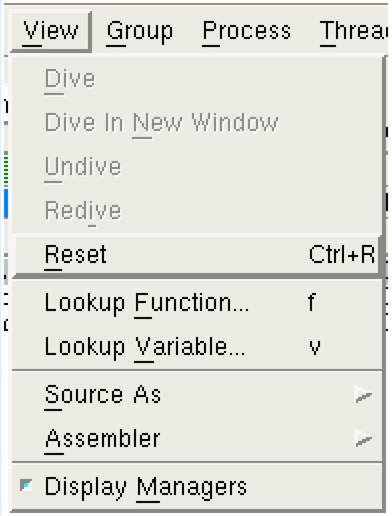
Preferences

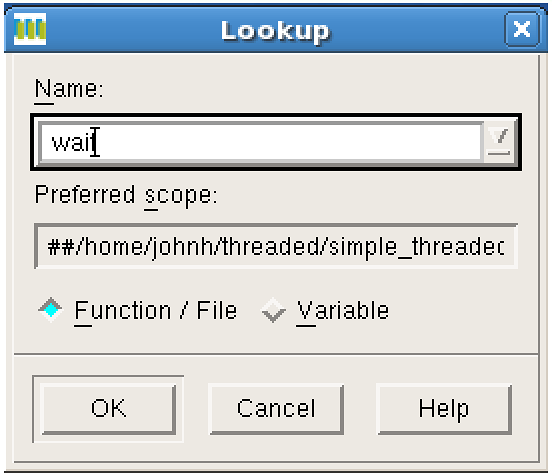


- □ Enable command line editing
- □ Force window positions (disables window manager placement modes)
- ☑ Enable tooltips
- □ Ignore SIGINT signal (Ctrl+C) delivered to TotalView
- ☑ Open process window on error signal
- ☑ Save preferences file automatically
- ☑ Show System Thread ID
- ☑ Stop control group on error signal
- ☑ View simplified STL containers (and user-defined transformations)
- ☑ C++View
- ☑ Warn about C++ exceptions during single step operations
- ☑ Show Startup Parameters when TotalView starts

Tab width: 8 Toolbar style: Icons above text

Finding Functions, Variables, and Source Files

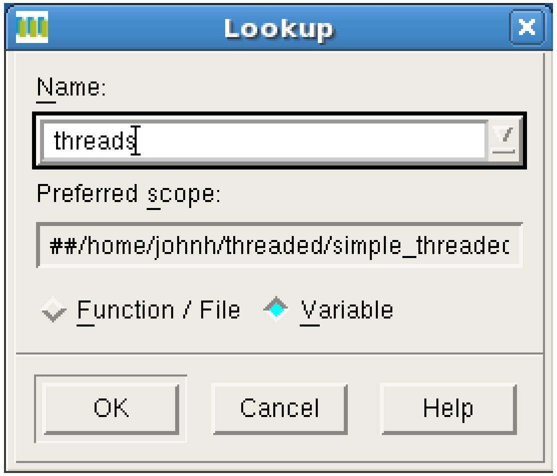




Name: wait

Preferred scope: ##/home/johnh/threaded/simple_threadec

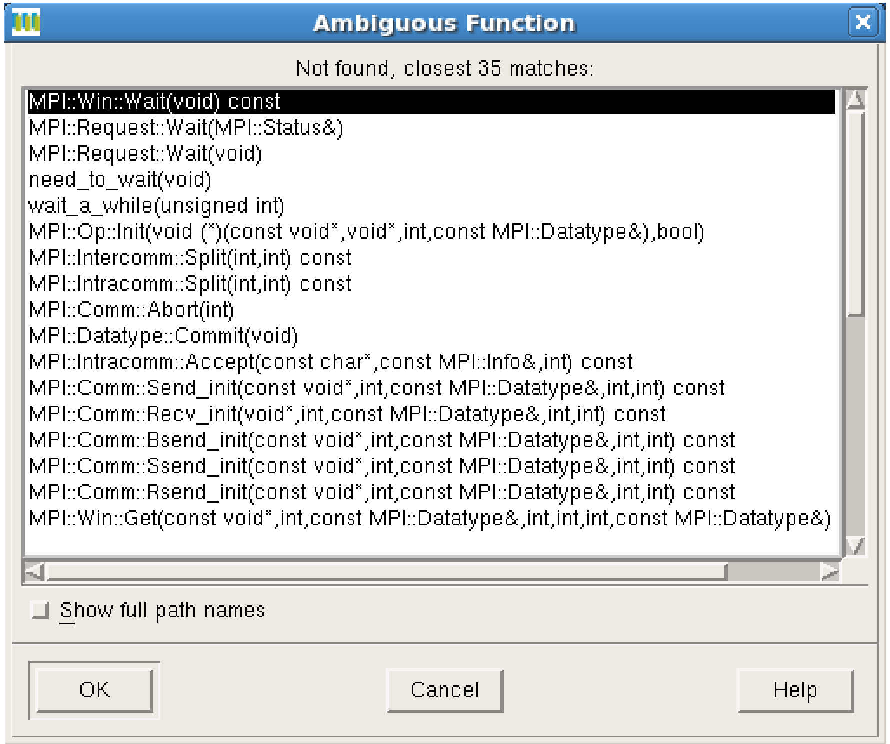
● Function / File ○ Variable



Name: threads

Preferred scope: ##/home/johnh/threaded/simple_threadec

○ Function / File ● Variable



Not found, closest 35 matches:

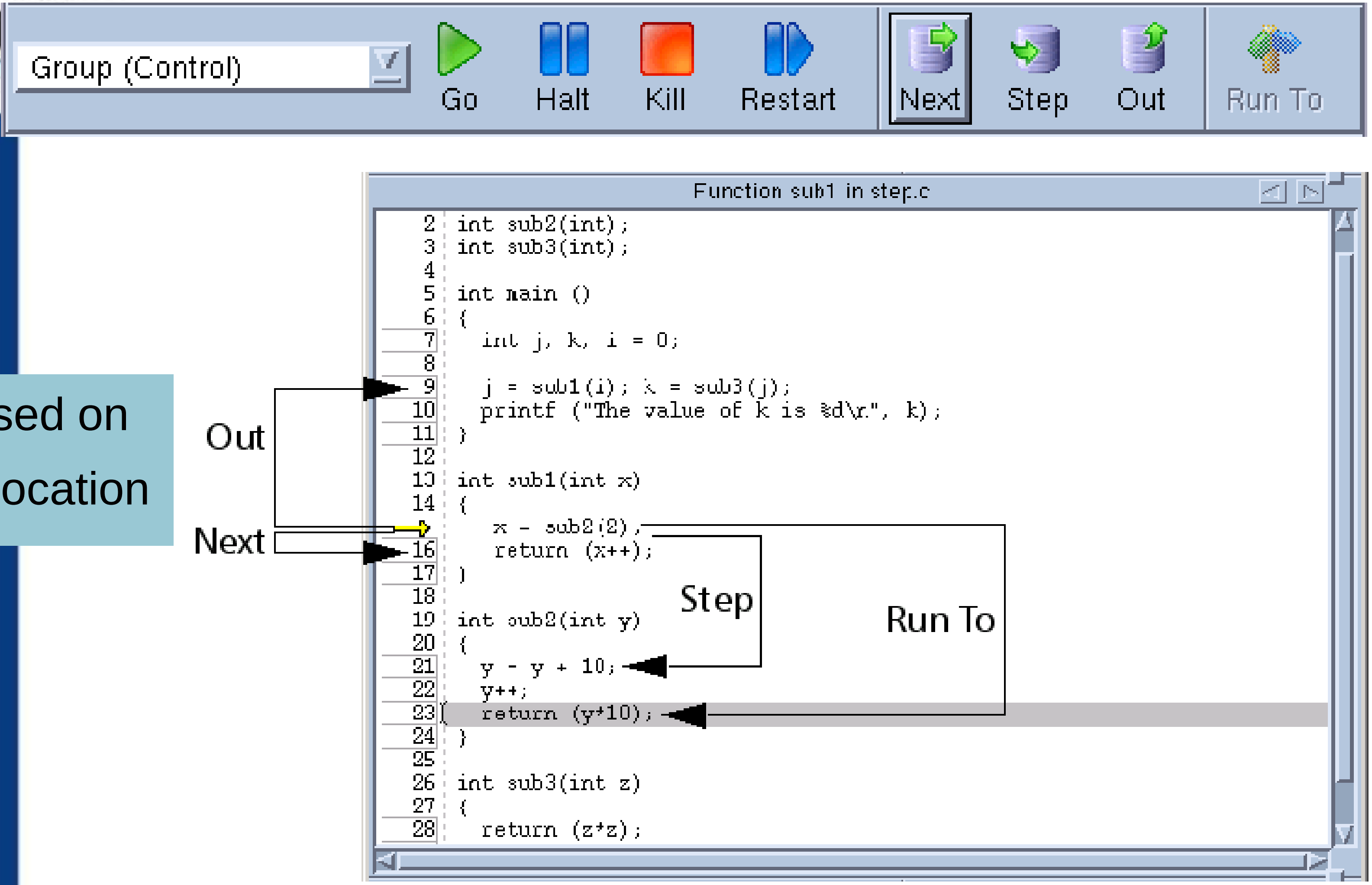
```
MPI::Win::Wait(void) const
MPI::Request::Wait(MPI::Status&)
MPI::Request::Wait(void)
need_to_wait(void)
wait_a_while(unsigned int)
MPI::Op::Init(void (*)(const void*,void*,int,const MPI::Datatype&),bool)
MPI::Intercomm::Split(int,int) const
MPI::Intracomm::Split(int,int) const
MPI::Comm::Abort(int)
MPI::Datatype::Commit(void)
MPI::Intracomm::Accept(const char*,const MPI::Info&,int) const
MPI::Comm::Send_init(const void*,int,const MPI::Datatype&,int,int) const
MPI::Comm::Recv_init(void*,int,const MPI::Datatype&,int,int) const
MPI::Comm::Bsend_init(const void*,int,const MPI::Datatype&,int,int) const
MPI::Comm::Ssend_init(const void*,int,const MPI::Datatype&,int,int) const
MPI::Comm::Rsend_init(const void*,int,const MPI::Datatype&,int,int) const
MPI::Win::Get(const void*,int,const MPI::Datatype&,int,int,int,const MPI::Datatype&)
```

□ Show full path names

Stepping Commands

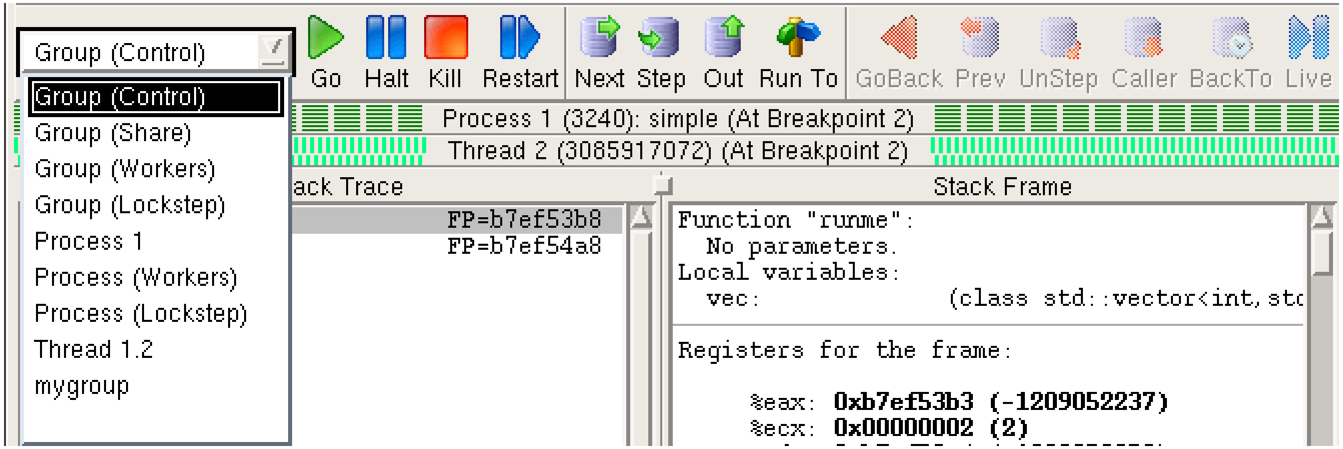
Based on

PC location



```
int sub2(int);
int sub3(int);

int main ()
{
  int j, k, i = 0;

  j = sub1(i); k = sub3(j);
  printf ("The value of k is %d\n", k);
}

int sub1(int x)
{
   x = sub2(2);
   return (x++);
}

int sub2(int y)
{
  y = y + 10;
  y++;
  return (y*10);
}

int sub3(int z)
{
  return (z*z);
```

Stepping Commands

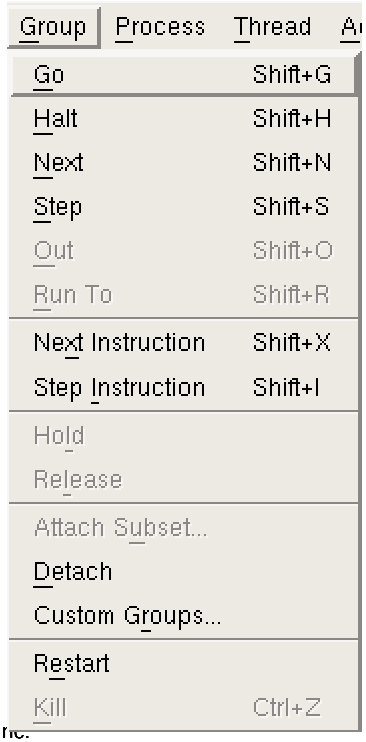


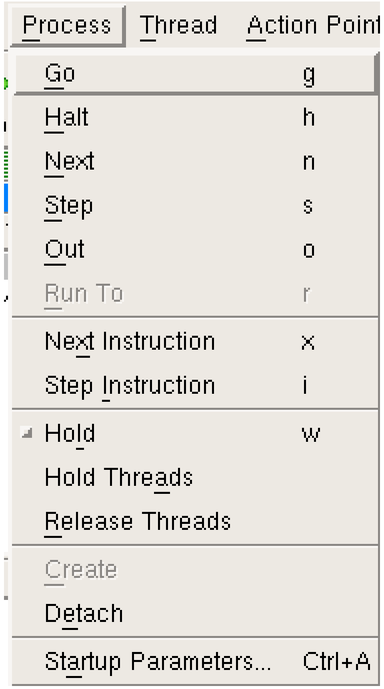
Process 1 (3240): simple (At Breakpoint 2)

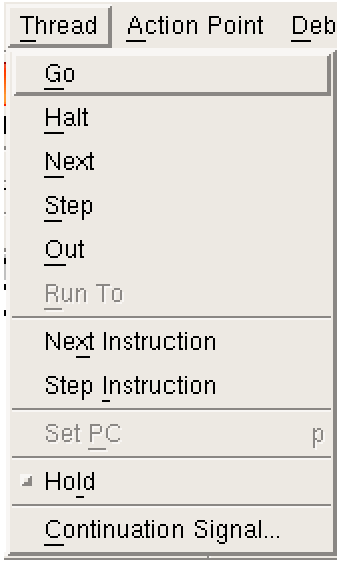
Thread 2 (3085917072) (At Breakpoint 2)

```
FP=b7ef53b8
FP=b7ef54a8
```

```
Function "runme":
  No parameters.
Local variables:
  vec:                 (class std::vector<int,std

Registers for the frame:

     %eax:  0xb7ef53b3 (-1209052237)
     %ecx:  0x00000002 (2)
```

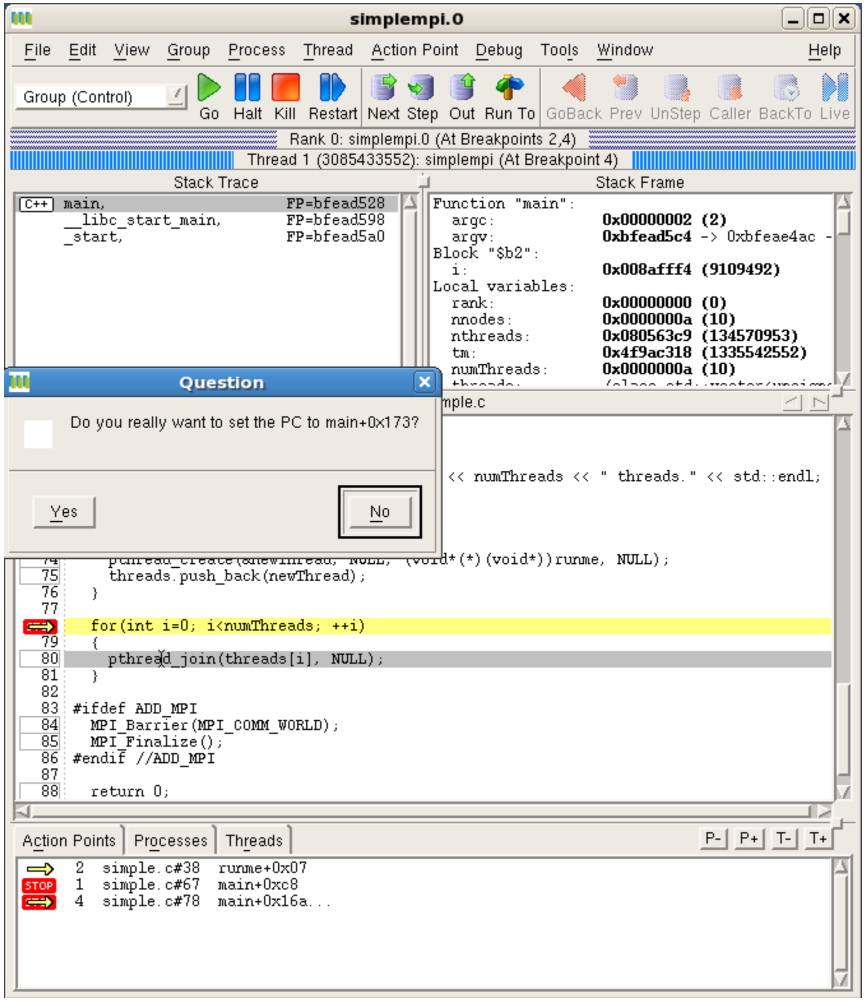






Using Set PC to resume execution at an arbitrary point

- **Select the line**
- **Thread->Set PC**
- **Click Yes to set the PC**



Rank 0: simplempi.0 (At Breakpoints 2,4)

Thread 1 (3085433552): simplempi (At Breakpoint 4)

```
C++  main,               FP=bfead528
     __libc_start_main,  FP=bfead598
     _start,             FP=bfead5a0
```

```
Function "main":
  argc:          0x00000002 (2)
  argv:          0xbfead5c4 -> 0xbfeae4ac -
Block "$b2":
  i:             0x008afff4 (9109492)
Local variables:
  rank:          0x00000000 (0)
  nnodes:        0x0000000a (10)
  nthreads:      0x080563c9 (134570953)
  tm:            0x4f9ac318 (1335542552)
  numThreads:    0x0000000a (10)
```

Question

Do you really want to set the PC to main+0x173?

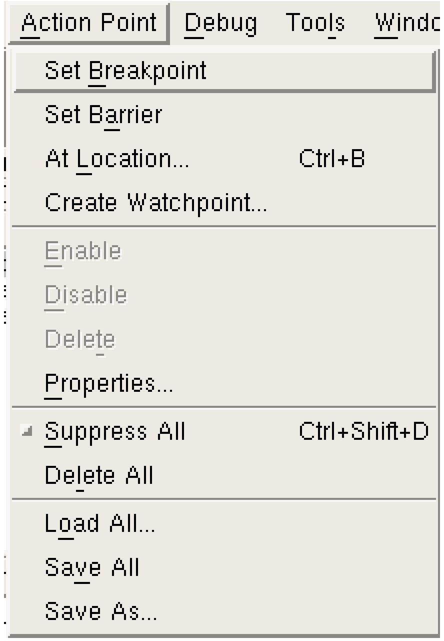
Yes   No

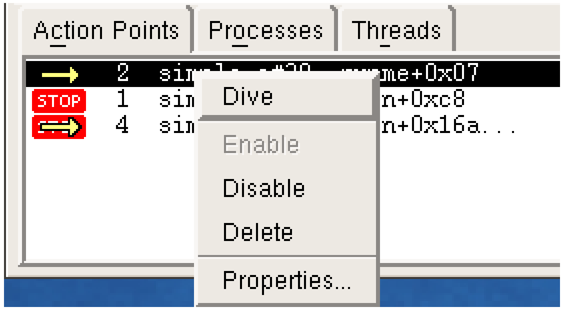
```
<< numThreads << " threads." << std::endl;

    pthread_create(&newThread, NULL, (void*(*)(void*))runme, NULL);
    threads.push_back(newThread);
  }

  for(int i=0; i<numThreads; ++i)
  {
    pthread_join(threads[i], NULL);
  }

#ifdef ADD_MPI
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();
#endif //ADD_MPI

  return 0;
```

```
      2  simple.c#38  runme+0x07
STOP  1  simple.c#67  main+0xc8
      4  simple.c#78  main+0x16a...
```

ACTION POINTS

Action Points





```
2 si        me+0x07
1 sim      n+0xc8
4 sim      n+0x16a. . .
```

Breakpoints

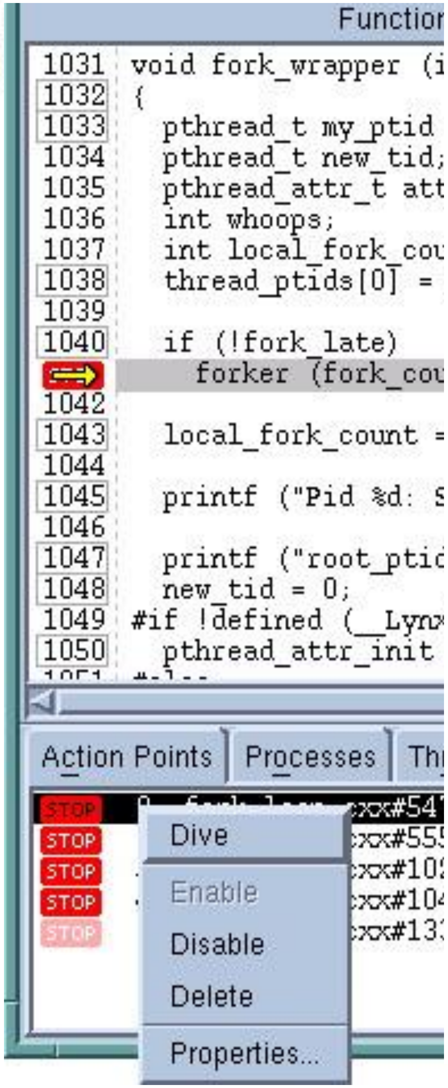
Barrier Points

Conditional Breakpoints

Evaluation Points

Watchpoints

Setting Breakpoints

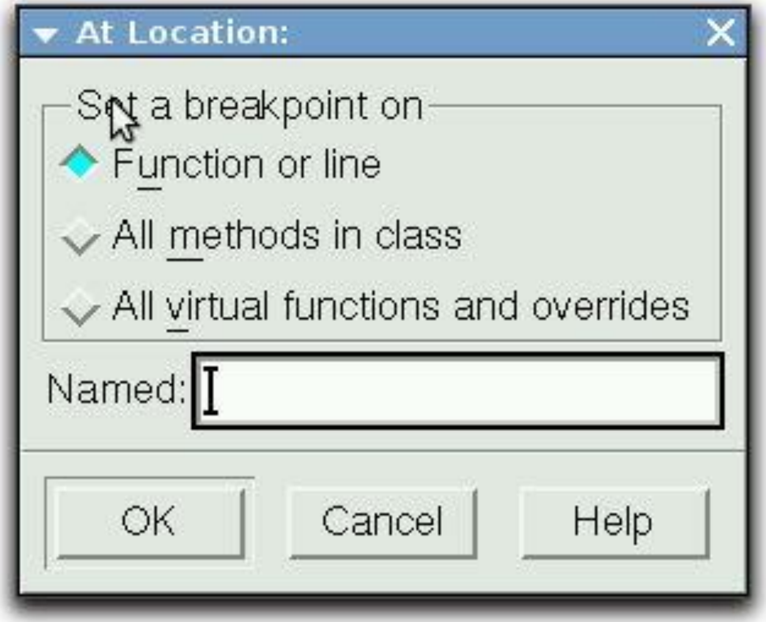


```
void fork_wrapper (i
{
  pthread_t my_ptid
  pthread_t new_tid;
  pthread_attr_t att
  int whoops;
  int local_fork_cou
  thread_ptids[0] =

  if (!fork_late)
    forker (fork_cou

  local_fork_count =

  printf ("Pid %d: S

  printf ("root_ptic
  new_tid = 0;
#if !defined (__Lynx
  pthread_attr_init
```

- 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

Setting Breakpoints

- **Breakpoint->At Location…**
  - Specify function name or line #
  - Specify class name and break on all methods in class, optionally with virtuals and overrides



At Location:

Set a breakpoint on

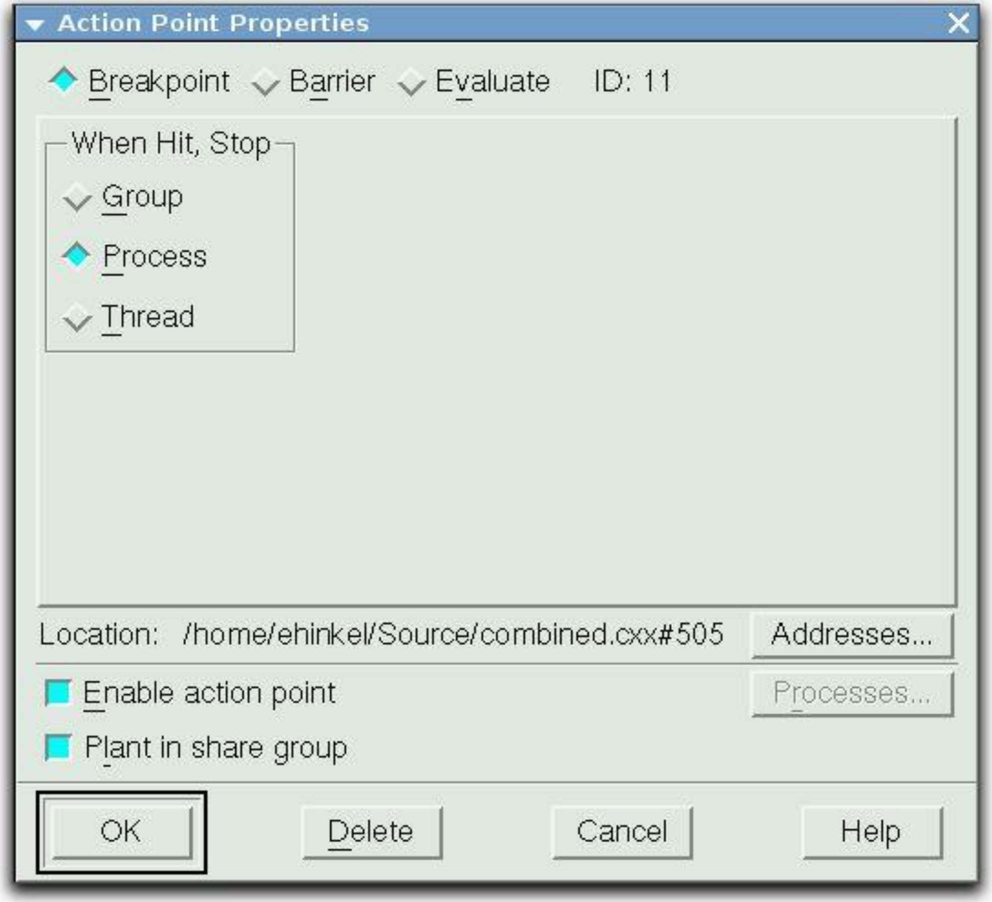
◆ Function or line

◇ All methods in class

◇ All virtual functions and overrides

Named:

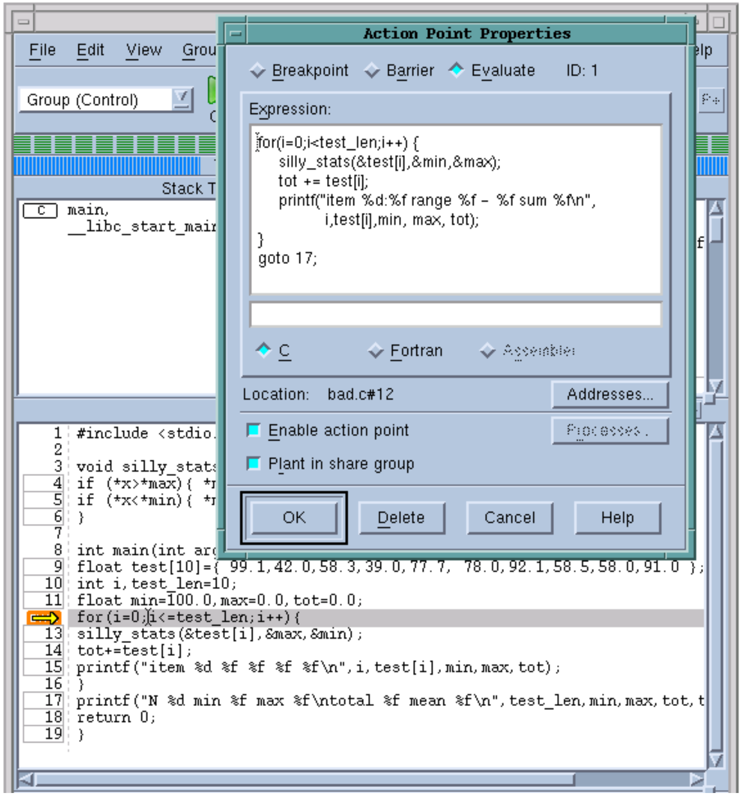
Setting Breakpoints



- Breakpoint type
- What to stop
- Set conditions
- Enable/disable
- In 1 process or share group

Evaluation Breakpoint…
Test Fixes on the Fly!

- Test small source code patches
- Call functions
- Set variables
- Test conditions
- C/C++ or Fortran
- Can’t use C++ constructors
- Use program variables



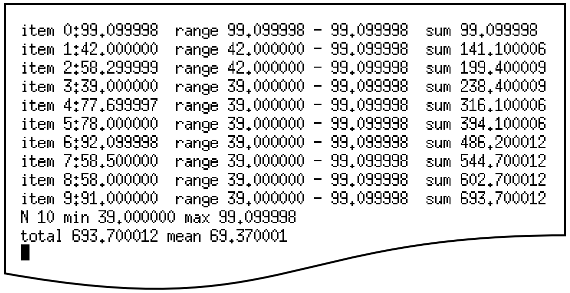
```
for(i=0;i<test_len;i++) {
     silly_stats(&test[i],&min,&max);
     tot += test[i];
     printf("item %d:%f range %f - %f sum %f\n",
            i,test[i],min, max, tot);
}
goto 17;
```

Location: bad.c#12

☑ Enable action point

☑ Plant in share group

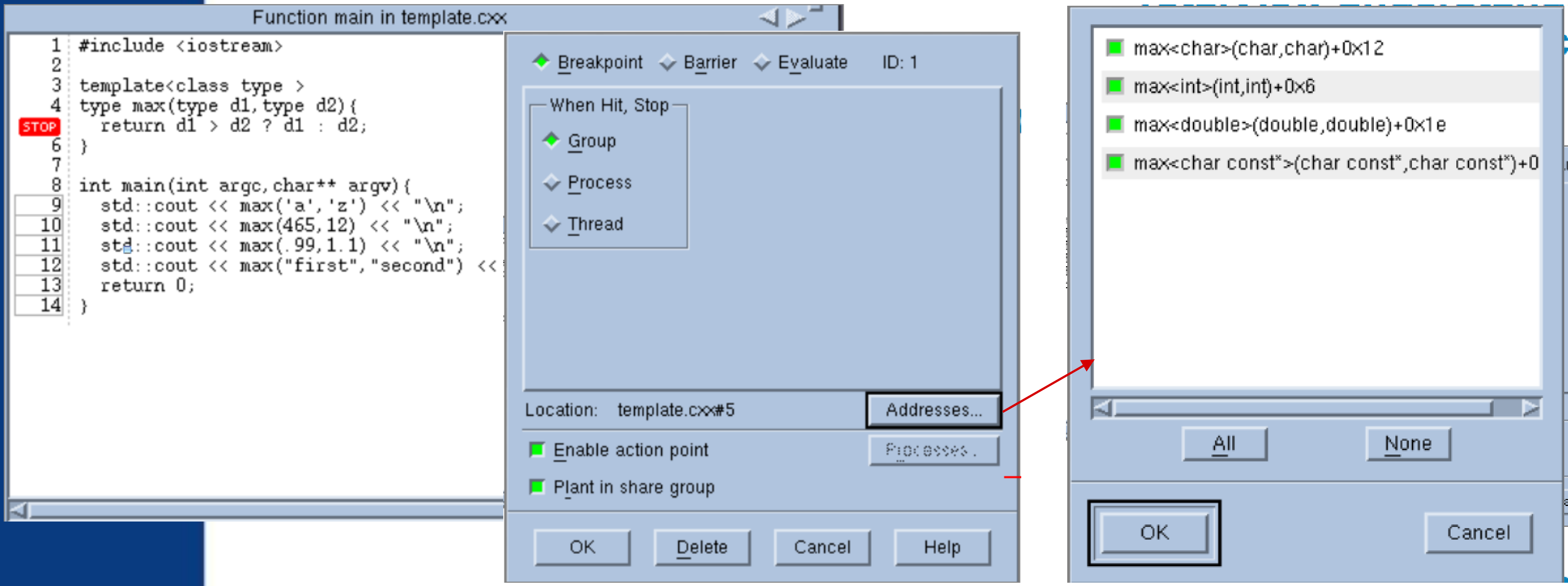
```
 1  #include <stdio.
 2
 3  void silly_stats
 4  if (*x>*max){ *r
 5  if (*x<*min){ *r
 6  }
 7
 8  int main(int arg
 9  float test[10]={ 99.1,42.0,58.3,39.0,77.7, 78.0,92.1,58.5,58.0,91.0 };
10  int i,test_len=10;
11  float min=100.0,max=0.0,tot=0.0;
⇨   for(i=0;i<=test_len;i++){
13  silly_stats(&test[i],&max,&min);
14  tot+=test[i];
15  printf("item %d %f %f %f %f\n",i,test[i],min,max,tot);
16  }
17  printf("N %d min %f max %f\ntotal %f mean %f\n",test_len,min,max,tot,t
18  return 0;
19  }
```



```
item 0:99.099998  range 99.099998 - 99.099998  sum 99.099998
item 1:42.000000  range 42.000000 - 99.099998  sum 141.100006
item 2:58.299999  range 42.000000 - 99.099998  sum 199.400009
item 3:39.000000  range 39.000000 - 99.099998  sum 238.400009
item 4:77.699997  range 39.000000 - 99.099998  sum 316.100006
item 5:78.000000  range 39.000000 - 99.099998  sum 394.100006
item 6:92.099998  range 39.000000 - 99.099998  sum 486.200012
item 7:58.500000  range 39.000000 - 99.099998  sum 544.700012
item 8:58.000000  range 39.000000 - 99.099998  sum 602.700012
item 9:91.000000  range 39.000000 - 99.099998  sum 693.700012
N 10 min 39.000000 max 99.099998
total 693.700012 mean 69.370001
█
```

Setting Breakpoints
With C++ Templates

TotalView understands C++ templates and gives you a choice ...



Function main in template.cxx

```
 1  #include <iostream>
 2
 3  template<class type >
 4  type max(type d1, type d2){
STOP  return d1 > d2 ? d1 : d2;
 6  }
 7
 8  int main(int argc,char** argv){
 9    std::cout << max('a','z') << "\n";
10    std::cout << max(465,12) << "\n";
11    std::cout << max(.99,1.1) << "\n";
12    std::cout << max("first","second") <<
13    return 0;
14  }
```

● Breakpoint ○ Barrier ○ Evaluate ID: 1

When Hit, Stop

● Group

○ Process

○ Thread

Location: template.cxx#5

☑ Enable action point

☑ Plant in share group

☑ max<char>(char,char)+0x12

☑ max<int>(int,int)+0x6

☑ max<double>(double,double)+0x1e

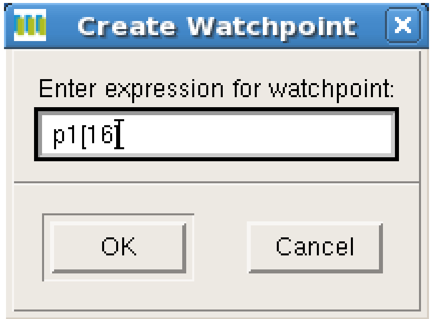
☑ max<char const*>(char const*,char const*)+0

Boxes with solid lines around line numbers indicate code that exists at more than one location.

Watchpoints

- **Watchpoints are set on a specific memory region**
- **Execution is stopped when that memory changes**

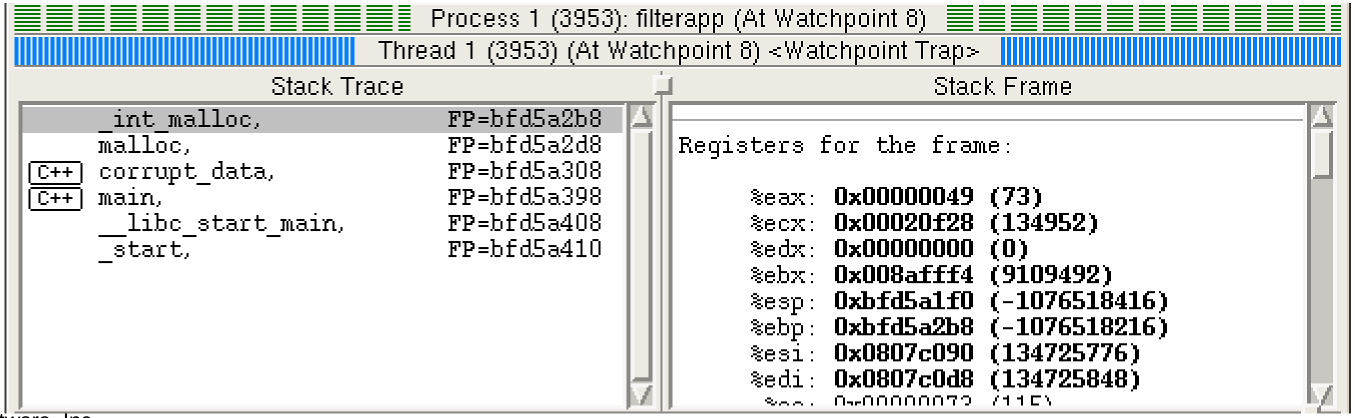
Action Point -> Create Watchpoint…



Create Watchpoint

Enter expression for watchpoint:

p1[16]



Process 1 (3953): filterapp (At Watchpoint 8)

Thread 1 (3953) (At Watchpoint 8) <Watchpoint Trap>

Stack Trace

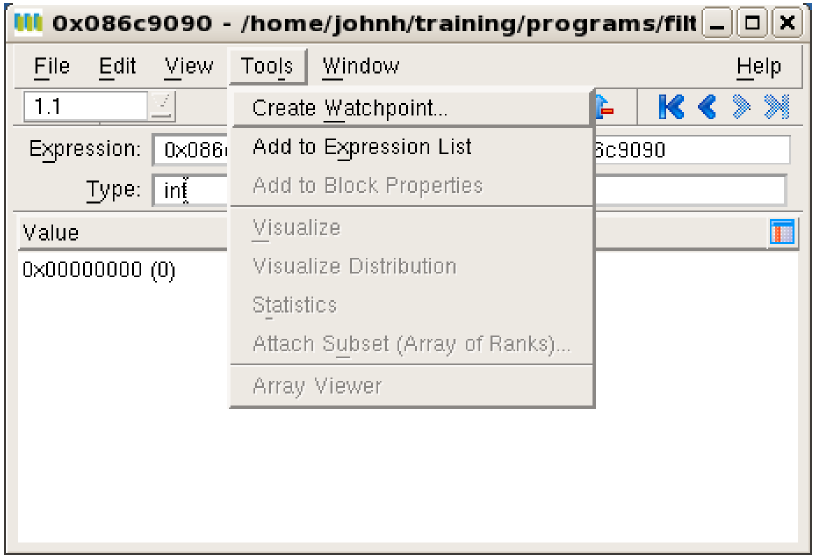
```
      _int_malloc,          FP=bfd5a2b8
      malloc,               FP=bfd5a2d8
C++   corrupt_data,         FP=bfd5a308
C++   main,                 FP=bfd5a398
      __libc_start_main,    FP=bfd5a408
      _start,               FP=bfd5a410
```

Stack Frame

```
Registers for the frame:

     %eax: 0x00000049 (73)
     %ecx: 0x00020f28 (134952)
     %edx: 0x00000000 (0)
     %ebx: 0x008afff4 (9109492)
     %esp: 0xbfd5a1f0 (-1076518416)
     %ebp: 0xbfd5a2b8 (-1076518216)
     %esi: 0x0807c090 (134725776)
     %edi: 0x0807c0d8 (134725848)
```

Watchpoints

- Can create from a variable window using Tools -> Watchpoint



0x086c9090 - /home/johnh/training/programs/filt

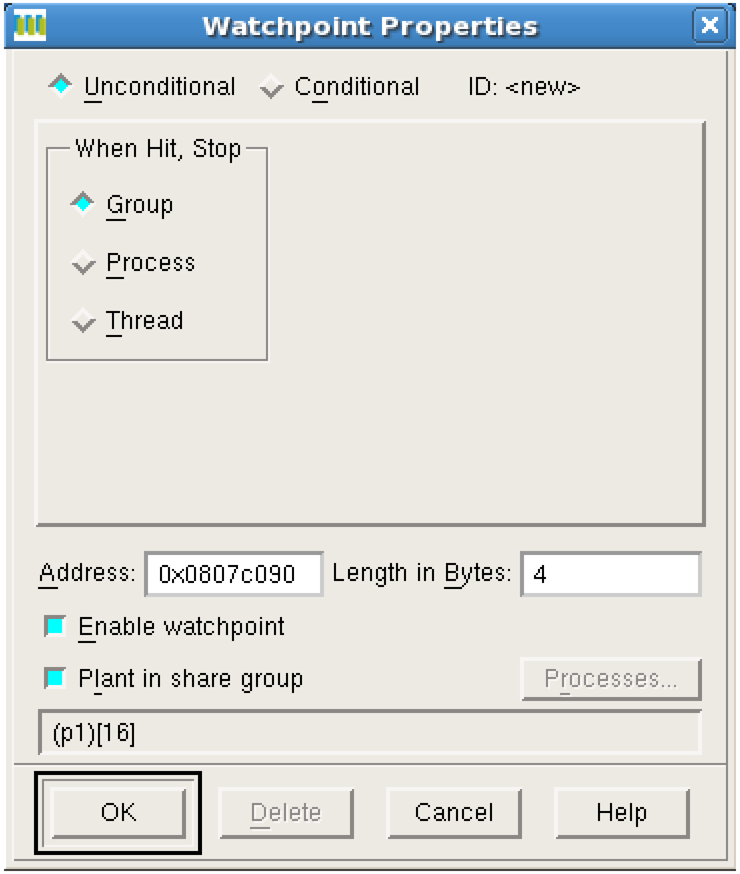
1.1

Expression: 0x086c9090

Type: int

Value

0x00000000 (0)



Watchpoint Properties

● Unconditional ○ Conditional ID: <new>

When Hit, Stop

● Group
○ Process
○ Thread

Address: 0x0807c090 Length in Bytes: 4

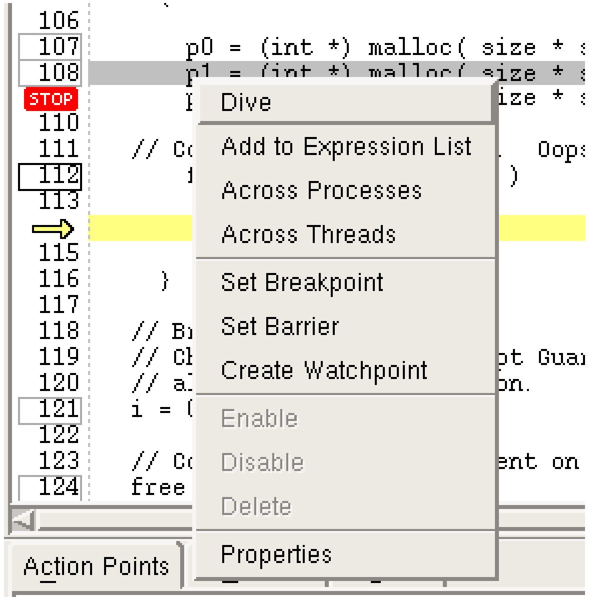
☑ Enable watchpoint

☑ Plant in share group

(p1)[16]

Watchpoints

- **Can create from right-click on variable in Source pane**



```
        p0 = (int *) malloc( size * s
        p1 = (int *) malloc( size * s
        I                        ize * s
    // Co                       .  Oops
        f                          )

    }

    // Br
    // Ch                       ot Gua
    // al                       on.
    i = (

    // Co                       ent on
    free
```

Watchpoints

- **Watchpoints are set on a memory region, not a variable**
- **Watch the variable scope and disable watchpoints when a variable is out of scope**
- **Can be conditional, just like other action points**
  - Use $newval and $oldval in your evaluation to find unexpected changes in value (such as a loop value changing by more than 1)

LAB 1: THE BASICS

DATA MONITORING AND VISUALIZATION

Diving on Variables

You can use Diving to:

… get more information

… open a variable in a Variable Window.

… chase pointers in complex data structures

… refocus the Process window Source Pane

You can Dive on:

… variable names to open a variable window

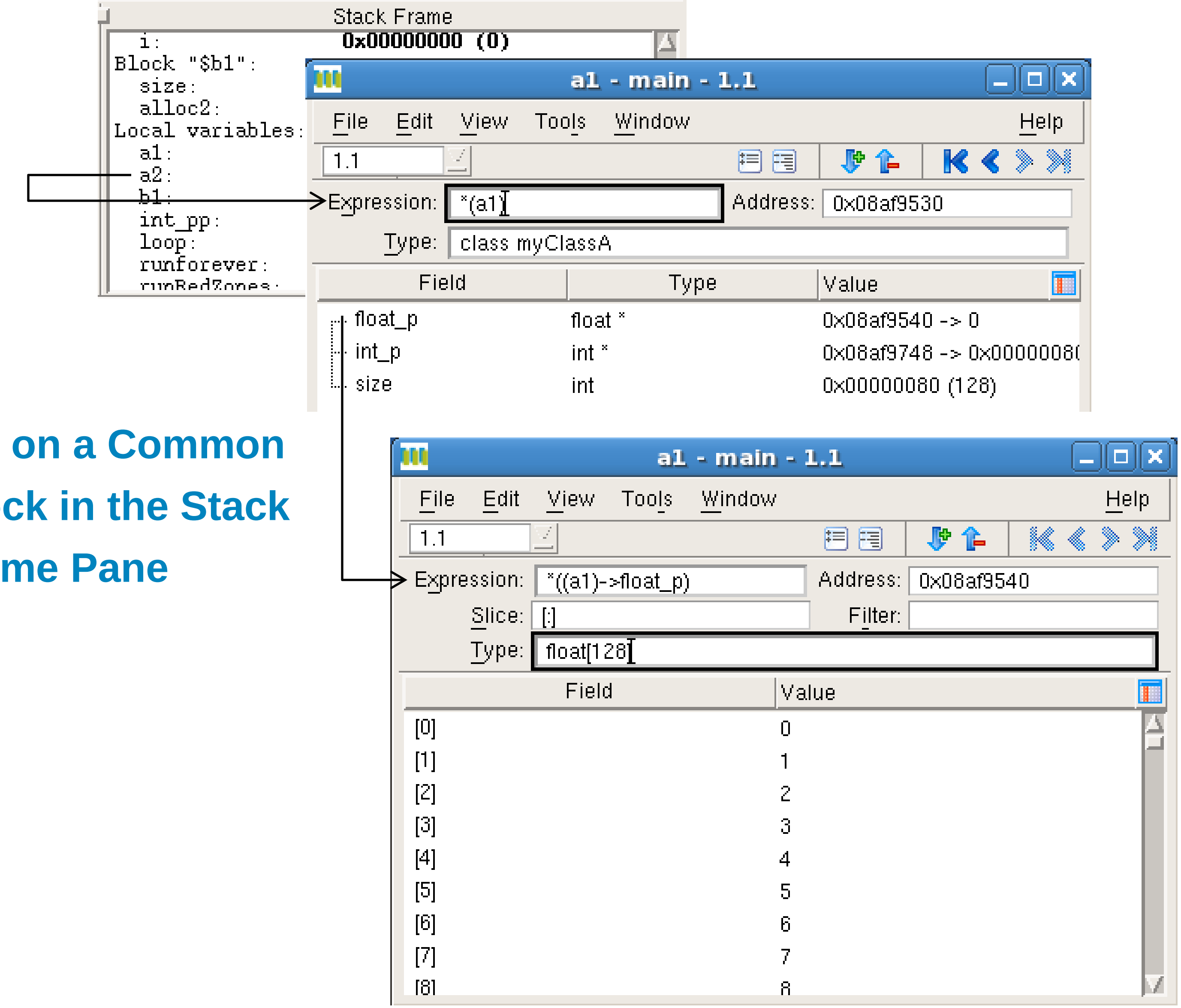
… function names to open the source in the Process Window.

… processes and threads in the Root Window.

How do I dive?

- Double-click the left mouse button on selection
- Single-click the middle mouse button on selection.
- Select Dive from context menu opened with the right mouse button

Diving



```
   i:            0x00000000 (0)
Block "$b1":
   size:
   alloc2:
Local variables:
   a1:
   a2:
   b1:
   int_pp:
   loop:
   runforever:
   runRedZones:
```

Expression: *(a1) Address: 0x08af9530

Type: class myClassA

Field	Type	Value
float_p	float *	0x08af9540 -> 0
int_p	int *	0x08af9748 -> 0x0000008(
size	int	0x00000080 (128)

Expression: *((a1)->float_p) Address: 0x08af9540

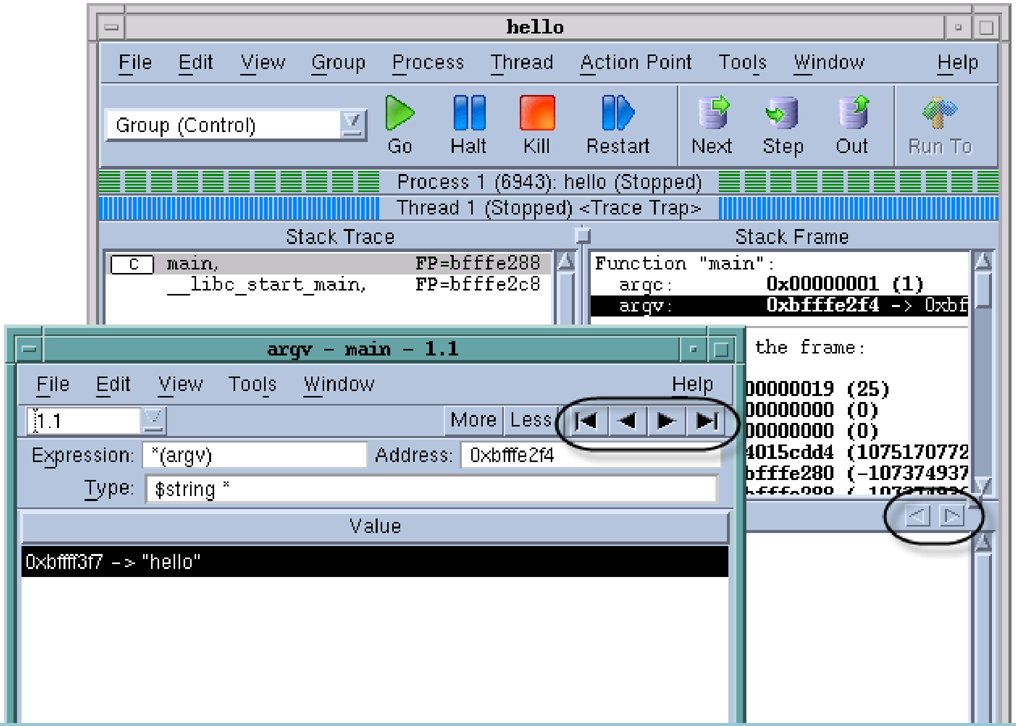
Slice: [:] Filter:

Type: float[128]

Field	Value
[0]	0
[1]	1
[2]	2
[3]	3
[4]	4
[5]	5
[6]	6
[7]	7
[8]	8

Diving on a Common Block in the Stack Frame Pane

Undiving


hello
File Edit View Group Process Thread Action Point Tools Window Help
Group (Control)
Go
Halt
Kill
Restart
Next
Step
Out
Run To
Process 1 (6943): hello (Stopped)
Thread 1 (Stopped) <Trace Trap>
Stack Trace
Stack Frame
argv - main - 1.1
File Edit View Tools Window Help
1.1
More
Less
Expression:
Address:
Type:
Value

```
C  main,               FP=bfffe288
   __libc_start_main,  FP=bfffe2c8
```

```
Function "main":
  argc:          0x00000001 (1)
  argv:          0xbfffe2f4 -> 0xbf
the frame:
00000019 (25)
00000000 (0)
00000000 (0)
4015cdd4 (1075170772
bfffe280 (-107374937
```

Expression: *(argv) Address: 0xbfffe2f4

Type: $string *

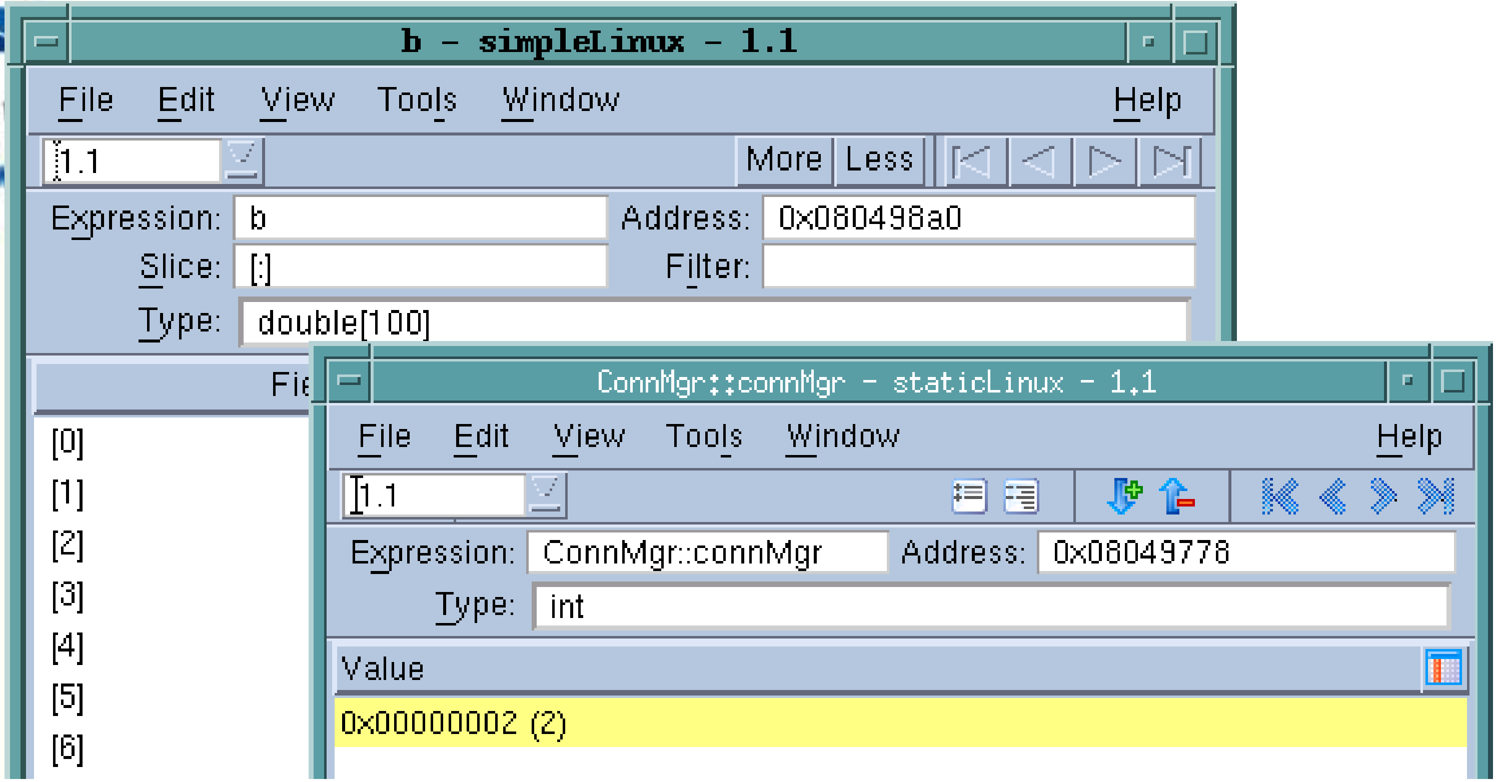
Value: 0xbffff3f7 -> "hello"

In a Process Window: retrace the path that has been explored with multiple dives.

In a Variable Window: replace contents with the previous contents.

You can also remove changes in the variable window with Edit > Reset Default.

The Variable Window



b - simpleLinux - 1.1

1.1

Expression: b

Address: 0x080498a0

Slice: [:]

Filter:

Type: double[100]

[0]
[1]
[2]
[3]
[4]
[5]
[6]

ConnMgr::connMgr - staticLinux - 1.1

1.1

Expression: ConnMgr::connMgr

Address: 0x08049778

Type: int

Value

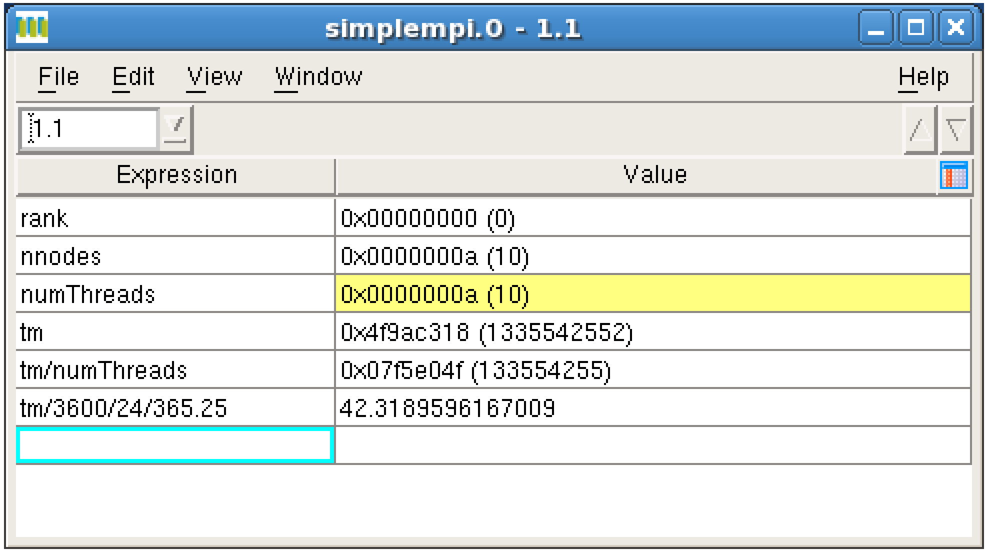
0x00000002 (2)

Editing Variables

- Window contents are updated automatically
- Changed values are highlighted
- “Last Value” column is available

- Click once on the value
- Cursor switches into edit more
- Esc key cancels editing
- Enter key commits a change
- Editing values changes the memory of the program

Expression List Window



Expression	Value
rank	0x00000000 (0)
nnodes	0x0000000a (10)
numThreads	0x0000000a (10)
tm	0x4f9ac318 (1335542552)
tm/numThreads	0x07f5e04f (133554255)
tm/3600/24/365.25	42.3189596167009

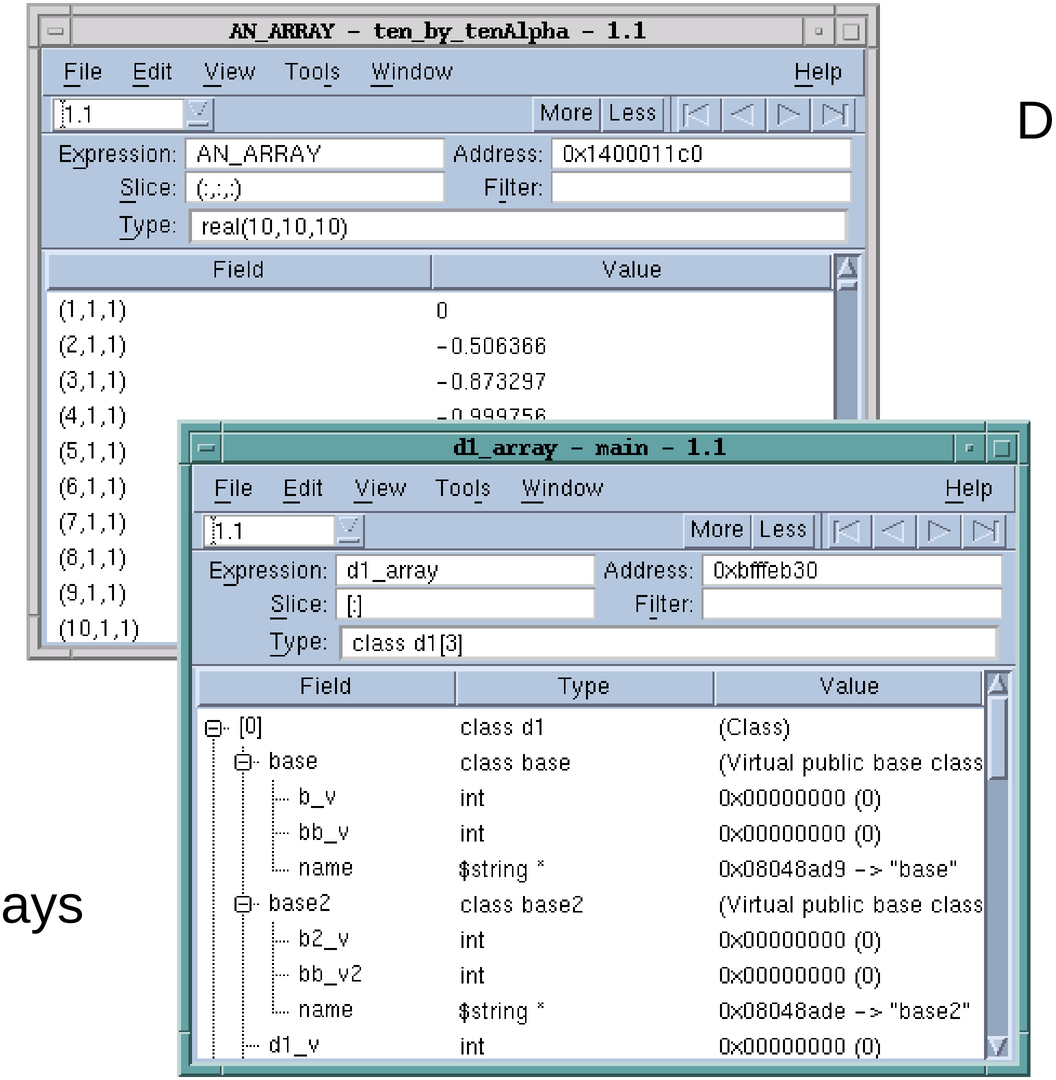
Add to the expression list using contextual menu with right-click on a variable, or by typing an expression directly in the window

- Reorder, delete, add
- Sort the expressions
- Edit expressions in place
- Dive to get more info

- Updated automatically
- Expression-based
- Simple values/expressions
- View just the values you want to monitor

Viewing Arrays

Data Arrays



Expression: AN_ARRAY
Address: 0x1400011c0
Slice: (:,:,:)
Filter:
Type: real(10,10,10)

Field	Value
(1,1,1)	0
(2,1,1)	-0.506366
(3,1,1)	-0.873297
(4,1,1)	-0.999756
(5,1,1)	
(6,1,1)	
(7,1,1)	
(8,1,1)	
(9,1,1)	
(10,1,1)	

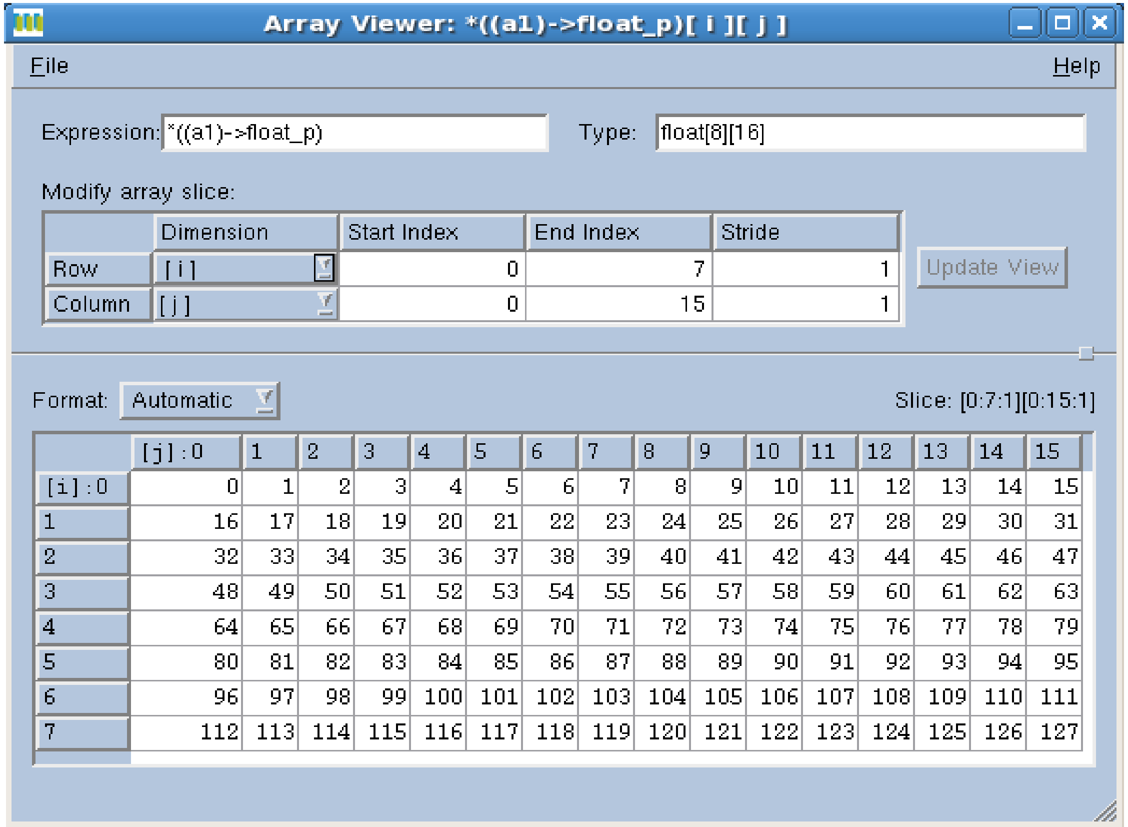
Expression: d1_array
Address: 0xbfffeb30
Slice: [:]
Filter:
Type: class d1[3]

Field	Type	Value
⊟ [0]	class d1	(Class)
⊟ base	class base	(Virtual public base class
b_v	int	0x00000000 (0)
bb_v	int	0x00000000 (0)
name	$string *	0x08048ad9 -> "base"
⊟ base2	class base2	(Virtual public base class
b2_v	int	0x00000000 (0)
bb_v2	int	0x00000000 (0)
name	$string *	0x08048ade -> "base2"
d1_v	int	0x00000000 (0)

Structure Arrays

Array Viewer

- **Variable Window select Tools -> Array Viewer**
- **View 2 dimensions of data**
  - Can be an arbitrary slice through a higher dimensional data cube
  - Can be strided



Expression: *((a1)->float_p)

Type: float[8][16]

Modify array slice:

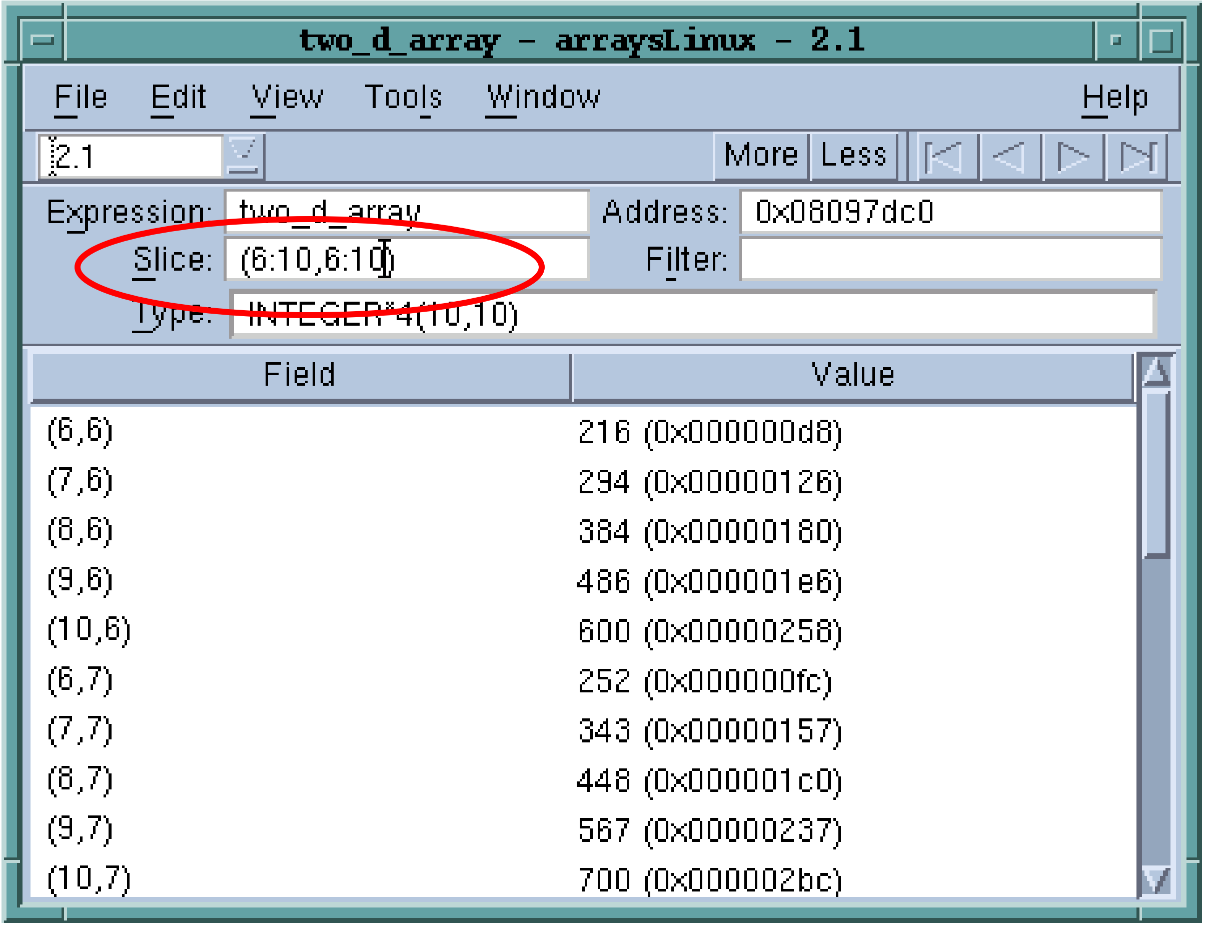
	Dimension	Start Index	End Index	Stride
Row	[i]	0	7	1
Column	[j]	0	15	1

Format: Automatic

Slice: [0:7:1][0:15:1]

	[j] : 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
[i] : 0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Slicing Arrays



Expression: two_d_array

Address: 0x08097dc0

Slice: (6:10,6:10)

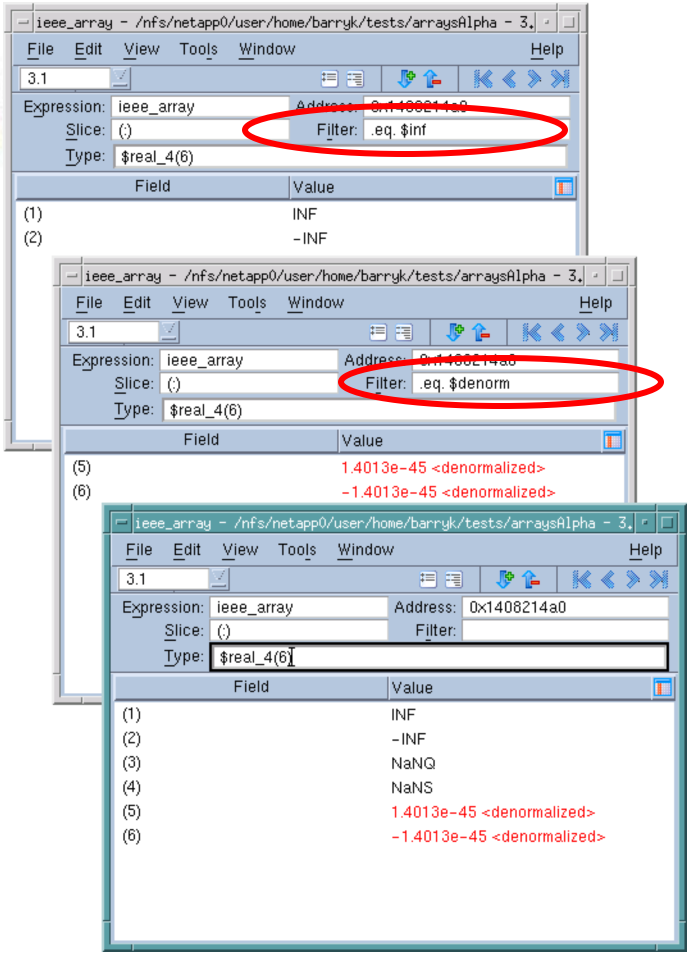
Filter:

Type: INTEGER*4(10,10)

Field	Value
(6,6)	216 (0x000000d8)
(7,6)	294 (0x00000126)
(8,6)	384 (0x00000180)
(9,6)	486 (0x000001e6)
(10,6)	600 (0x00000258)
(6,7)	252 (0x000000fc)
(7,7)	343 (0x00000157)
(8,7)	448 (0x000001c0)
(9,7)	567 (0x00000237)
(10,7)	700 (0x000002bc)

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

Filtering Arrays



ieee_array - /nfs/netapp0/user/home/barryk/tests/arraysAlpha - 3.

Expression: ieee_array
Address: 0x1408214a0
Slice: (:)
Filter: .eq. $inf
Type: $real_4(6)

```
(1)    INF
(2)    -INF
```

ieee_array - /nfs/netapp0/user/home/barryk/tests/arraysAlpha - 3.

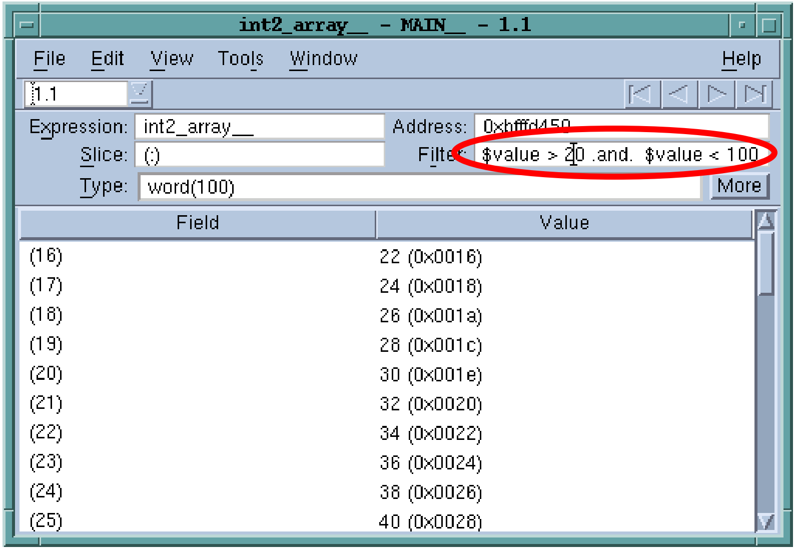
Expression: ieee_array
Address: 0x1408214a0
Slice: (:)
Filter: .eq. $denorm
Type: $real_4(6)

```
(5)    1.4013e-45 <denormalized>
(6)    -1.4013e-45 <denormalized>
```

ieee_array - /nfs/netapp0/user/home/barryk/tests/arraysAlpha - 3.

Expression: ieee_array
Address: 0x1408214a0
Slice: (:)
Filter:
Type: $real_4(6)

```
(1)    INF
(2)    -INF
(3)    NaNQ
(4)    NaNS
(5)    1.4013e-45 <denormalized>
(6)    -1.4013e-45 <denormalized>
```

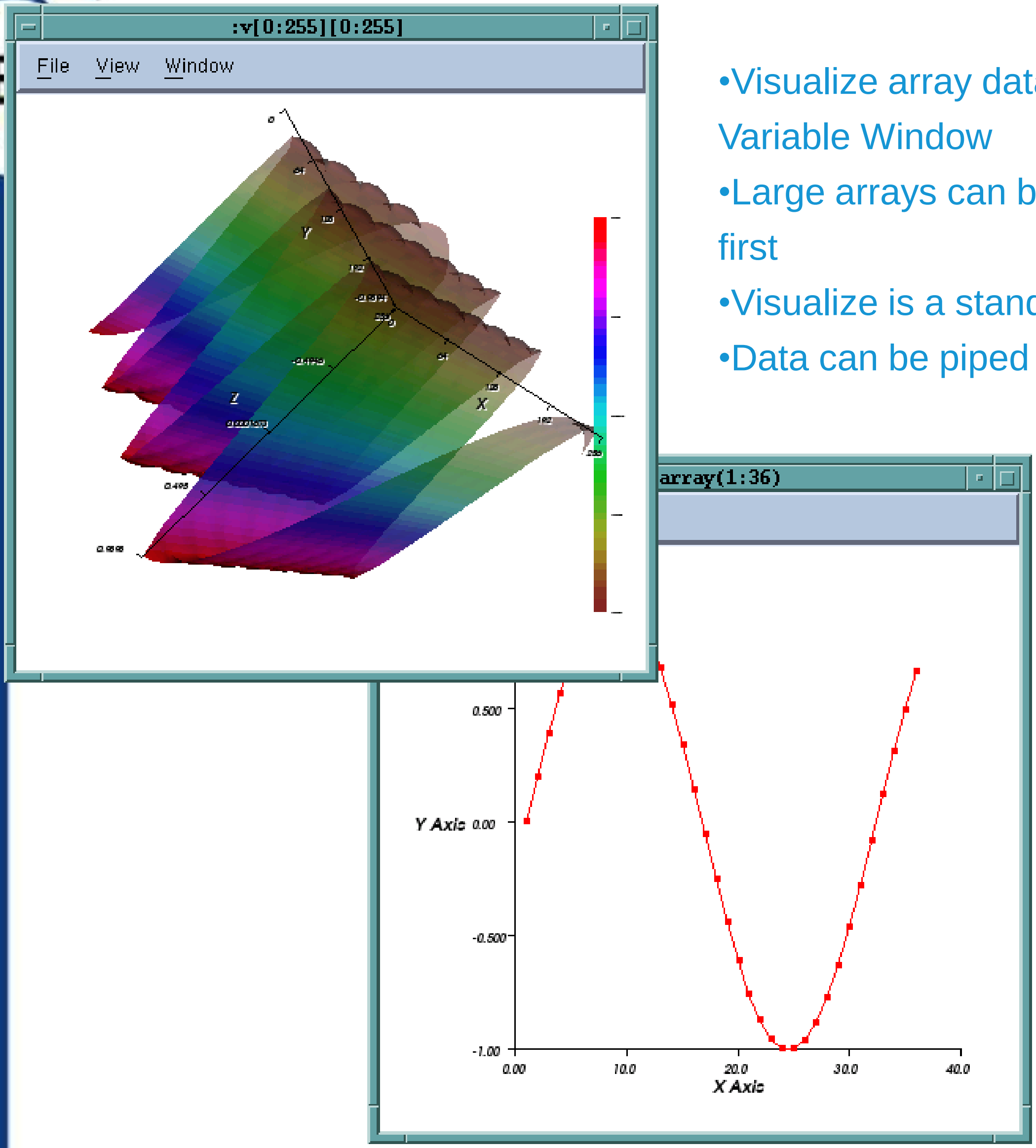


int2_array__ - MAIN__ - 1.1

Expression: int2_array__
Address: 0xbfffd450
Slice: (:)
Filter: $value > 20 .and. $value < 100
Type: word(100)

```
(16)    22 (0x0016)
(17)    24 (0x0018)
(18)    26 (0x001a)
(19)    28 (0x001c)
(20)    30 (0x001e)
(21)    32 (0x0020)
(22)    34 (0x0022)
(23)    36 (0x0024)
(24)    38 (0x0026)
(25)    40 (0x0028)
```

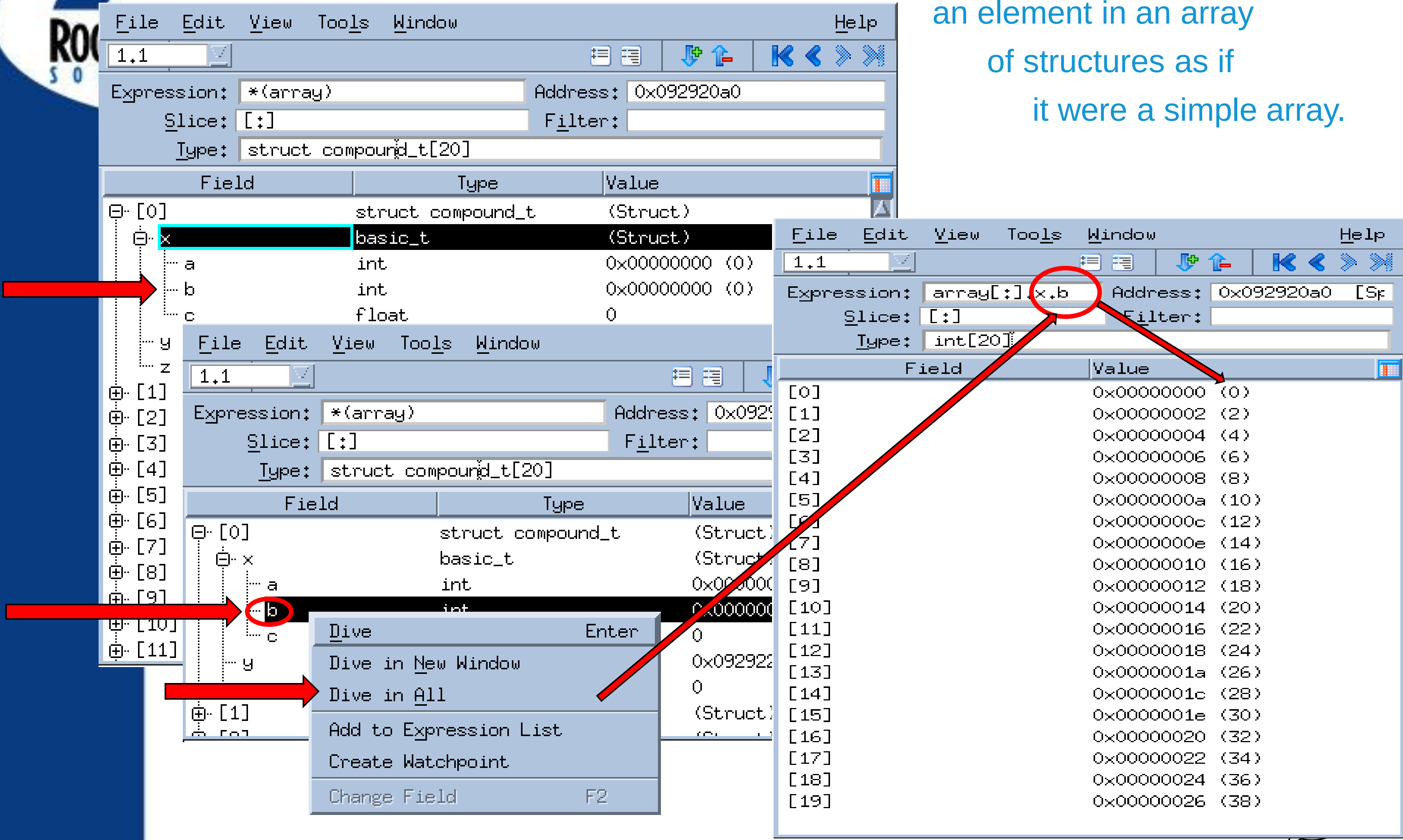
Visualizing Arrays



- •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.

Dive in All

Dive in All will display
an element in an array
of structures as if
it were a simple array.



Expression: *(array) Address: 0x092920a0
Slice: [:] Filter:
Type: struct compound_t[20]

Field	Type	Value
[0]	struct compound_t	(Struct)
x	basic_t	(Struct)
a	int	0x00000000 (0)
b	int	0x00000000 (0)
c	float	0
y		
z		
[1]		
[2]		
[3]		
[4]		
[5]		
[6]		
[7]		
[8]		
[9]		
[10]		
[11]		

Expression: *(array) Address: 0x0929
Slice: [:] Filter:
Type: struct compound_t[20]

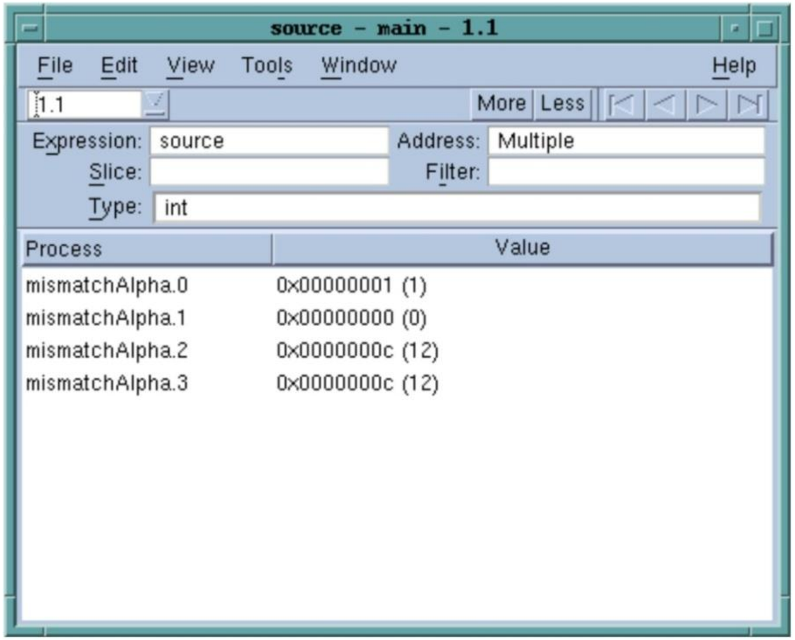
Field	Type	Value
[0]	struct compound_t	(Struct)
x	basic_t	(Struct
a	int	0x000000
b	int	0x000000
c		0
y		0x092922
		0
[1]		(Struct)

Expression: array[:].x.b Address: 0x092920a0 [Sp
Slice: [:] Filter:
Type: int[20]

Field	Value
[0]	0x00000000 (0)
[1]	0x00000002 (2)
[2]	0x00000004 (4)
[3]	0x00000006 (6)
[4]	0x00000008 (8)
[5]	0x0000000a (10)
[6]	0x0000000c (12)
[7]	0x0000000e (14)
[8]	0x00000010 (16)
[9]	0x00000012 (18)
[10]	0x00000014 (20)
[11]	0x00000016 (22)
[12]	0x00000018 (24)
[13]	0x0000001a (26)
[14]	0x0000001c (28)
[15]	0x0000001e (30)
[16]	0x00000020 (32)
[17]	0x00000022 (34)
[18]	0x00000024 (36)
[19]	0x00000026 (38)

Looking at Variables across Processes

- **TotalView allows you to look at the value of a variable in all MPI processes**
  - Right Click on the variable
  - Select the View > View Across
- **TotalView creates an array indexed by process**
- **You can filter and visualize**
- **Use for viewing distributed arrays as well.**



Expression: source
Address: Multiple
Slice:
Filter:
Type: int

Process	Value
mismatchAlpha.0	0x00000001 (1)
mismatchAlpha.1	0x00000000 (0)
mismatchAlpha.2	0x0000000c (12)
mismatchAlpha.3	0x0000000c (12)

Typecasting Variables

- Edit the type of a variable
- View data as type…
- Often used with pointers

Type Casts Read from Right to Left

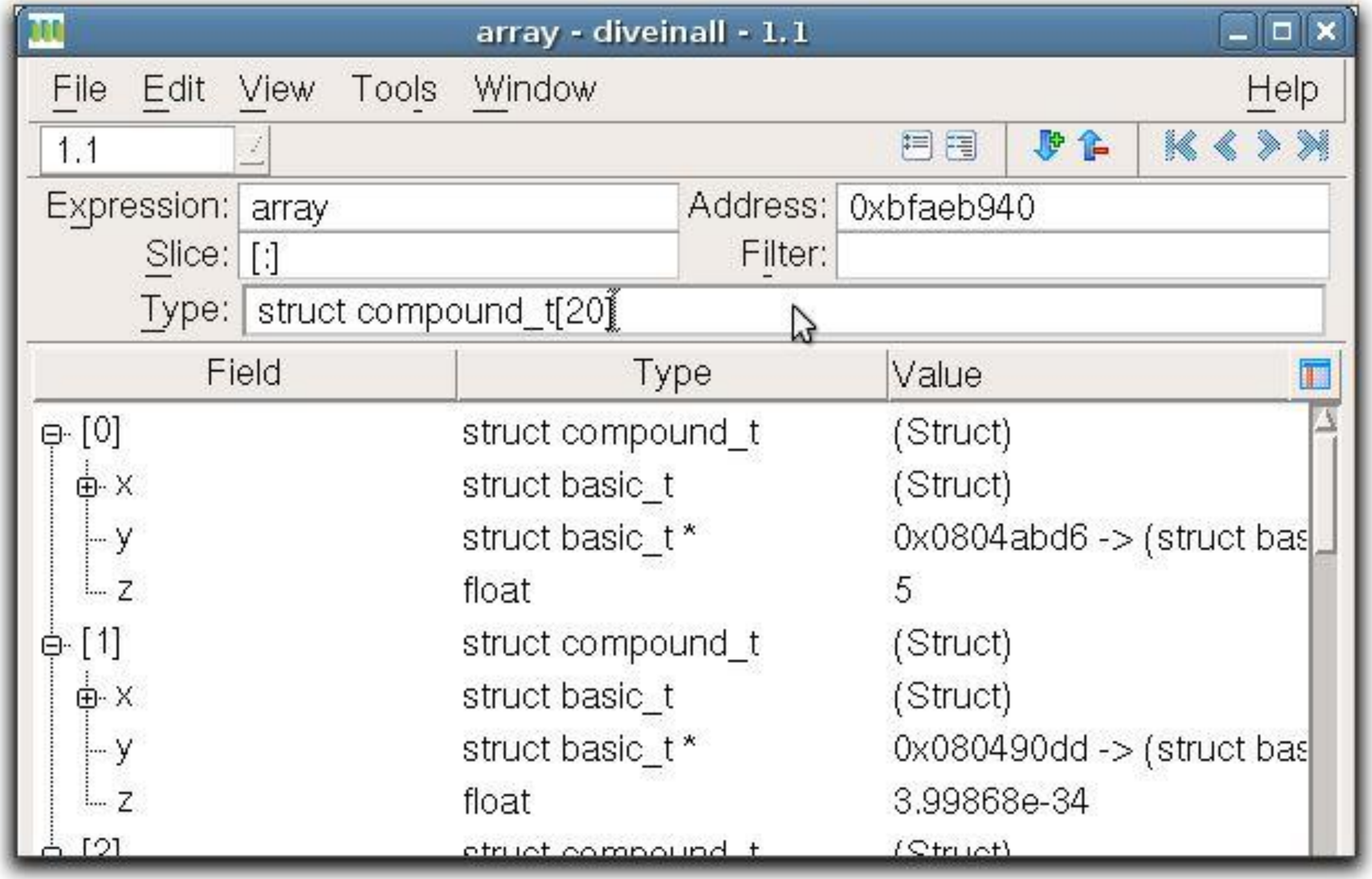
int[10]*	Pointer to an array of 10 int
int*[10]	Array of 10 pointers to int

- **Cast** float * **to** float [100]* **to see a dynamic array's values**
- **Cast to built-in types like $**string **to view a variable as a null-terminated string**
- **Cast to $**void **for no type interpretation or for displaying regions of memory**

The Bottom Line

Give TotalView a starting memory address and you can tell TotalView how to interpret your memory from that starting location.

Typecasting a Dynamic Array



Expression: array

Address: 0xbfaeb940

Slice: [:]

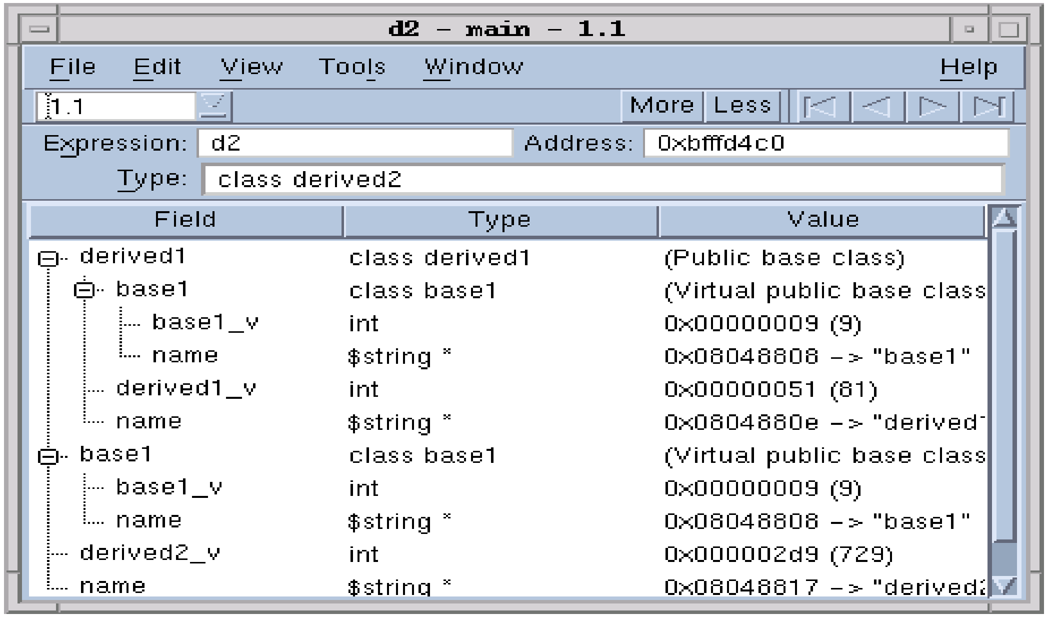
Filter:

Type: struct compound_t[20]

Field	Type	Value
[0]	struct compound_t	(Struct)
x	struct basic_t	(Struct)
y	struct basic_t *	0x0804abd6 -> (struct bas
z	float	5
[1]	struct compound_t	(Struct)
x	struct basic_t	(Struct)
y	struct basic_t *	0x080490dd -> (struct bas
z	float	3.99868e-34
[2]	struct compound_t	(Struct)

C++ Class Hierarchies

Variable Window shows class hierarchy using indentation



Expression: d2

Address: 0xbfffd4c0

Type: class derived2

Field	Type	Value
derived1	class derived1	(Public base class)
base1	class base1	(Virtual public base class
base1_v	int	0x00000009 (9)
name	$string *	0x08048808 -> "base1"
derived1_v	int	0x00000051 (81)
name	$string *	0x0804880e -> "derived
base1	class base1	(Virtual public base class
base1_v	int	0x00000009 (9)
name	$string *	0x08048808 -> "base1"
derived2_v	int	0x000002d9 (729)
name	$string *	0x08048817 -> "derived

Example:

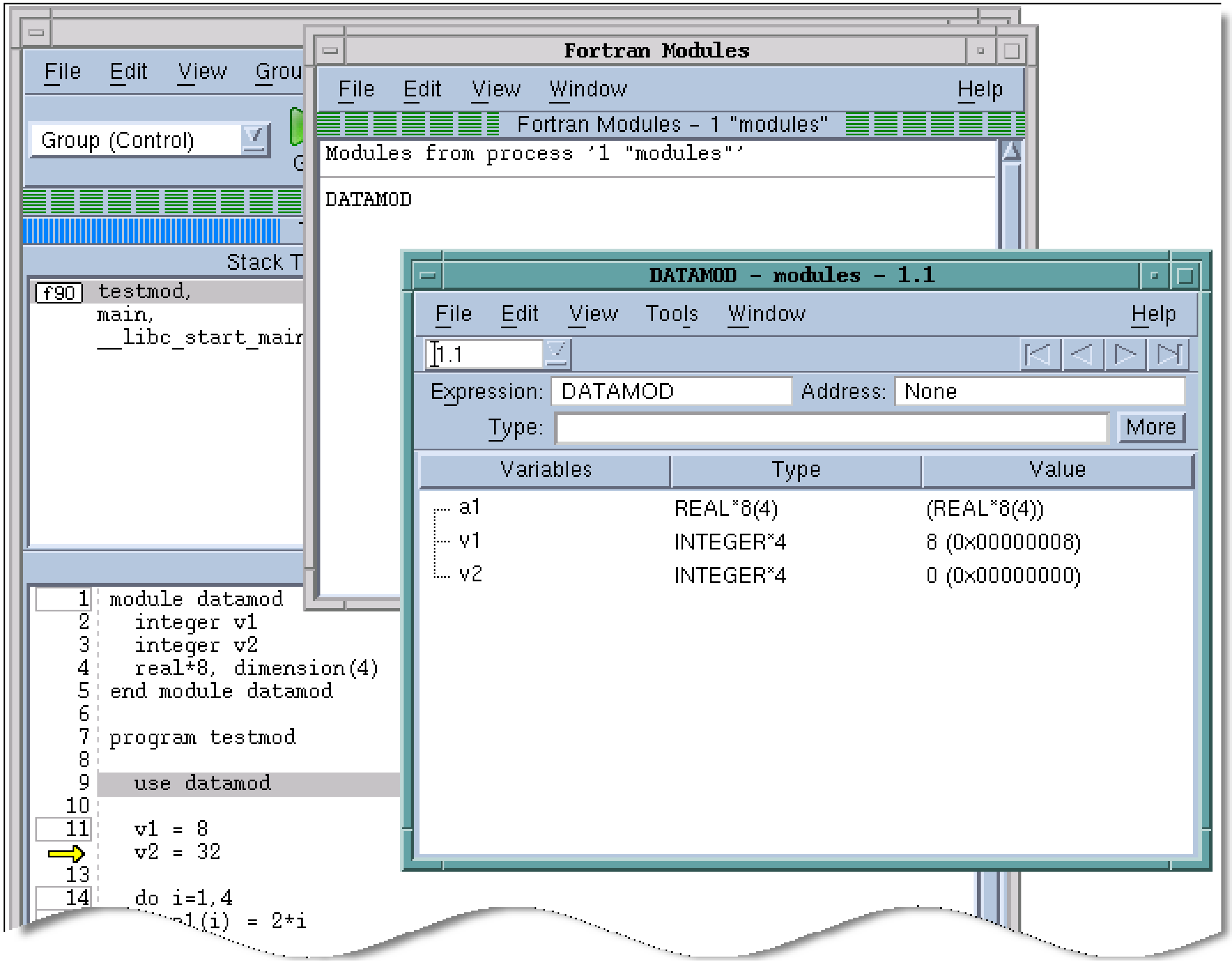
- derived2 inherits from base1 and derived1
- derived1 inherits from base1

Note:

- Virtual public base classes appear each time they are referenced
- The vtable entry here is part of the C++ implementation but can provide useful information

Fortran 90 Modules

Tools > Fortran Modules



Fortran Modules - 1 "modules"

Modules from process '1 "modules"'

DATAMOD

```
f90 testmod,
    main,
    __libc_start_mair
```

Expression: DATAMOD Address: None

Type:

Variables	Type	Value
a1	REAL*8(4)	(REAL*8(4))
v1	INTEGER*4	8 (0x00000008)
v2	INTEGER*4	0 (0x00000000)

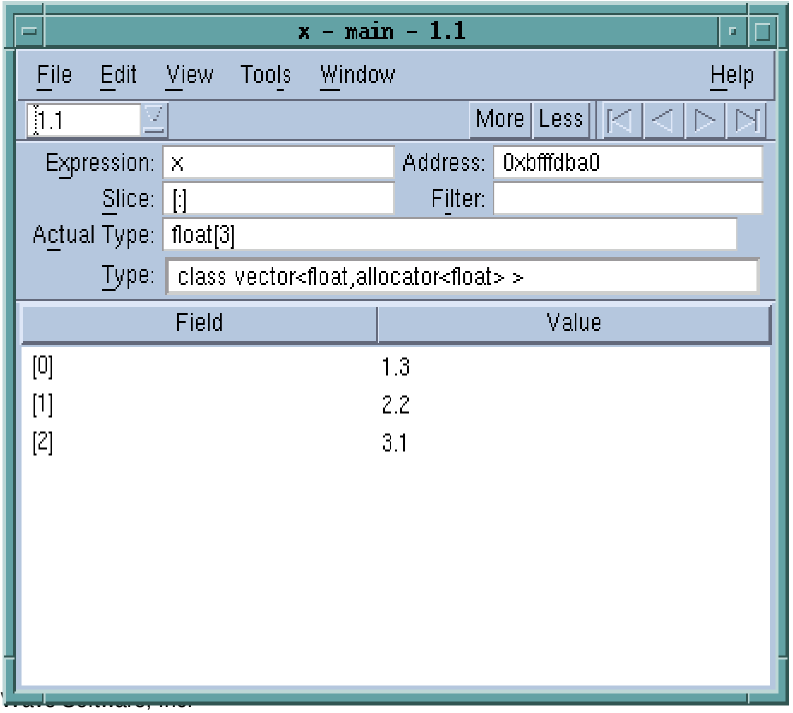
```
module datamod
  integer v1
  integer v2
  real*8, dimension(4)
end module datamod

program testmod

  use datamod

  v1 = 8
  v2 = 32

  do i=1,4
    a1(i) = 2*i
```

STLView

STLView transforms templates into readable and understandable information

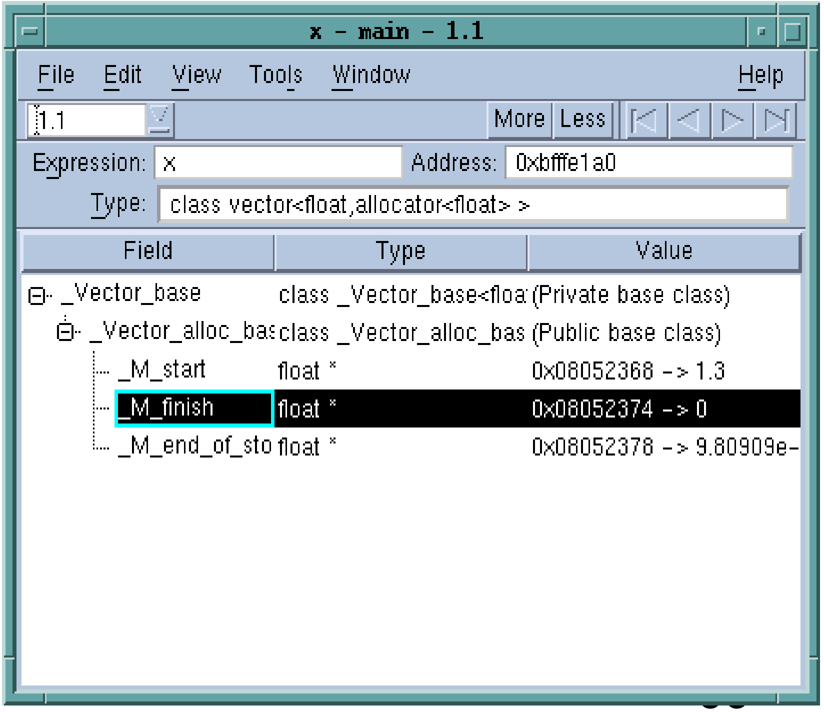
–STLView supports std::vector, std::list, std::map, std::string

–See doc for which STL implementations are supported



Expression: x
Address: 0xbfffdba0
Slice: [:]
Filter:
Actual Type: float[3]
Type: class vector<float,allocator<float> >

Field	Value
[0]	1.3
[1]	2.2
[2]	3.1

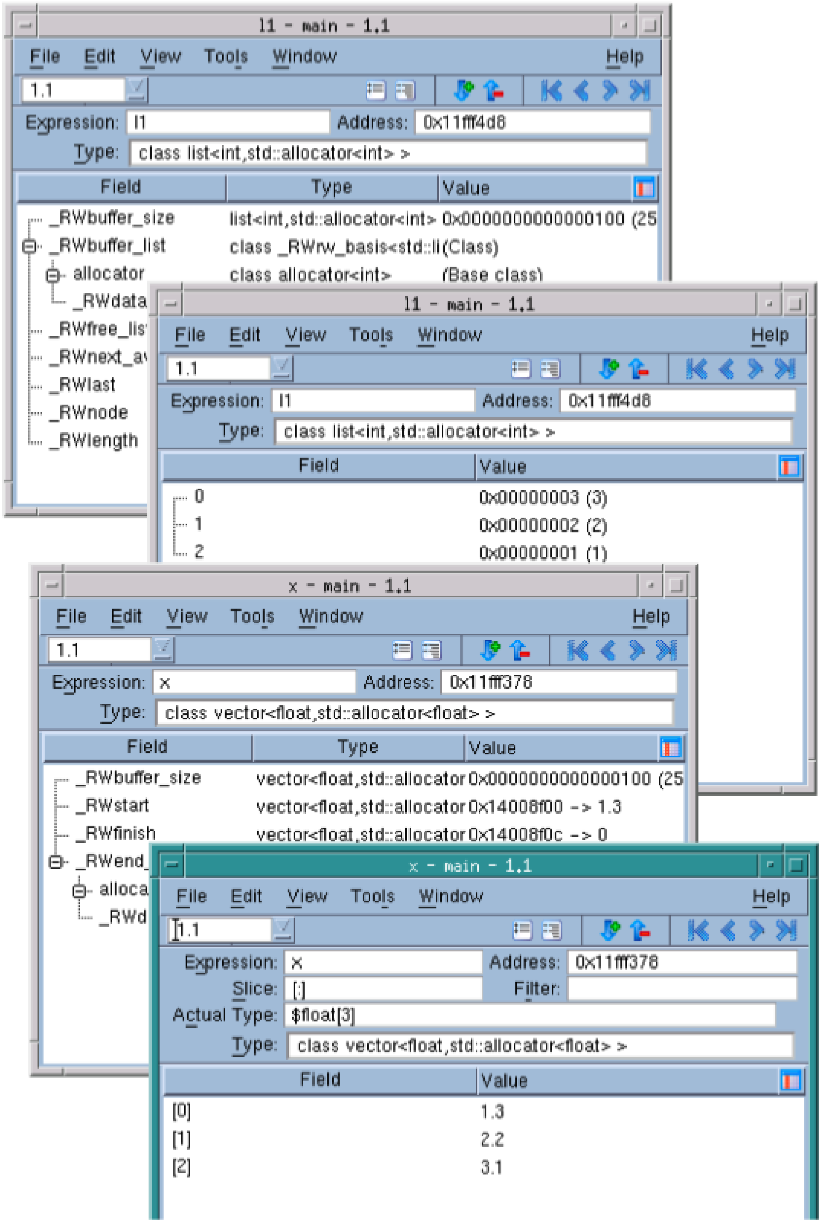


Expression: x
Address: 0xbfffe1a0
Type: class vector<float,allocator<float> >

Field	Type	Value
_Vector_base	class _Vector_base<floa	(Private base class)
_Vector_alloc_bas	class _Vector_alloc_bas	(Public base class)
_M_start	float *	0x08052368 -> 1.3
_M_finish	float *	0x08052374 -> 0
_M_end_of_sto	float *	0x08052378 -> 9.80909e-

STLView

STLView transforms templates into readable and understandable information



Untransformed list

```
l1 - main - 1.1
File  Edit  View  Tools  Window                Help
1.1
Expression: l1                  Address: 0x11fff4d8
      Type: class list<int,std::allocator<int> >
Field            Type                          Value
_RWbuffer_size   list<int,std::allocator<int>  0x0000000000000100 (25
_RWbuffer_list   class _RWrw_basis<std::li     (Class)
  allocator      class allocator<int>          (Base class)
    _RWdata
_RWfree_lis
_RWnext_av
_RWlast
_RWnode
_RWlength
```

Transformed list

```
l1 - main - 1.1
File  Edit  View  Tools  Window                Help
1.1
Expression: l1                  Address: 0x11fff4d8
      Type: class list<int,std::allocator<int> >
Field                 Value
0                     0x00000003 (3)
1                     0x00000002 (2)
2                     0x00000001 (1)
```

Untransformed vector

```
x - main - 1.1
File  Edit  View  Tools  Window                Help
1.1
Expression: x                   Address: 0x11fff378
      Type: class vector<float,std::allocator<float> >
Field            Type                         Value
_RWbuffer_size   vector<float,std::allocator  0x0000000000000100 (25
_RWstart         vector<float,std::allocator  0x14008f00 -> 1.3
_RWfinish        vector<float,std::allocator  0x14008f0c -> 0
_RWend_
  alloca
    _RWd
```

Transformed vector

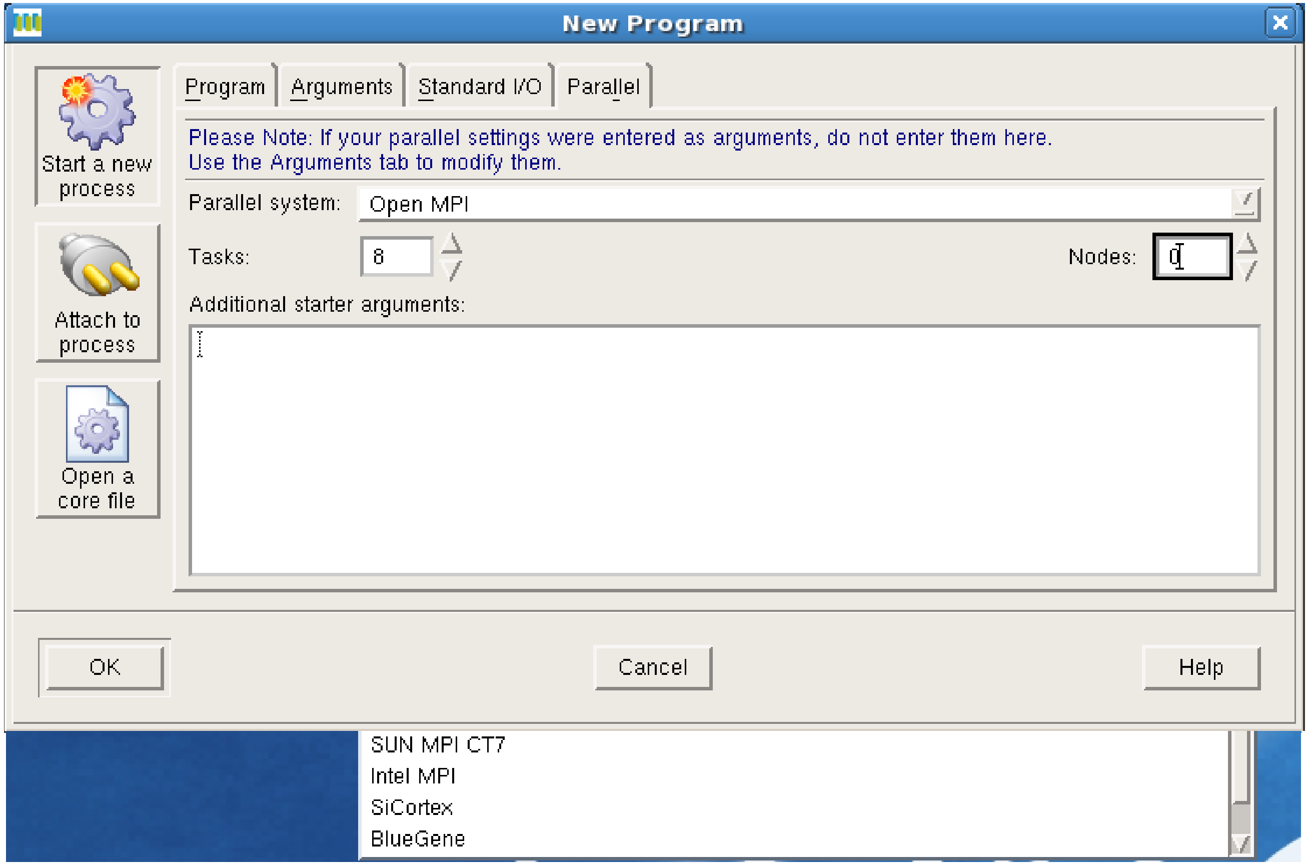
```
x - main - 1.1
File  Edit  View  Tools  Window                Help
1.1
 Expression: x                  Address: 0x11fff378
      Slice: [:]                 Filter:
Actual Type: $float[3]
       Type: class vector<float,std::allocator<float> >
Field                 Value
[0]                   1.3
[1]                   2.2
[2]                   3.1
```

LAB 2: VIEWING, EXAMING,WATCHING AND EDITING DATA

DEBUGGING FOR PARALLEL APPLICATIONS

TotalView Startup with MPI

In the Parallel tab, select:



Please Note: If your parallel settings were entered as arguments, do not enter them here.
Use the Arguments tab to modify them.

Parallel system: Open MPI

Tasks: 8

Nodes: 0

Additional starter arguments:

SUN MPI CT7
Intel MPI
SiCortex
BlueGene

your MPI preference, number of tasks, and number of nodes.

… then add any additional starter arguments

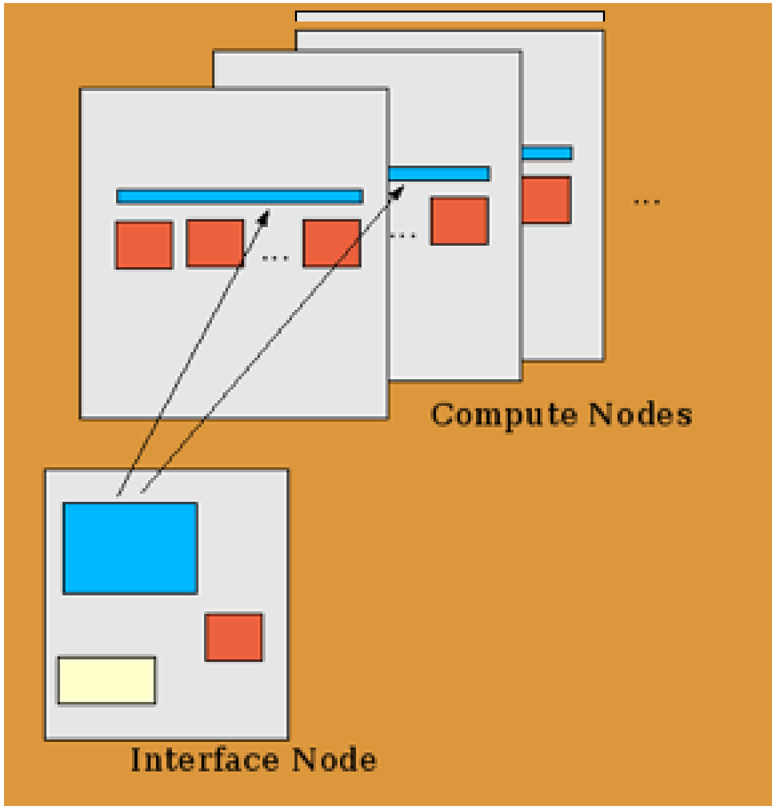
TotalView Startup with MPI: old school

IBM	`totalview poe -a myprog -procs 4 -rmpool 0`
QUADRICS **Intel Linux under SLURM**	`totalview srun -a -n 16 -p pdebug myprog`
MVAPICH **Opteron Linux under SLURM**	`totalview srun -a -n 16 -p pdebug myprog`
SGI	`totalview mpirun -a myprog -np 16`
Sun	`totalview mprun -a myprog -np 16`
MPICH	`mpirun -np 16 -tv myprog`
MPICH2 **Intel MPI**	`Totalview python –a ‘which mpiexec’ –tvsu -np 16 myprog`

The order of arguments and executables is important, and differs between platforms.

Architecture for Cluster Debugging

- **Single Front End (TotalView)**
  - GUI
  - debug engine
- **Debugger Agents (tvdsvr)**
  - Low overhead, 1 per node
  - Traces multiple rank processes
- **TotalView communicates directly with tvdsvrs**
  - Not using MPI
  - Protocol optimization

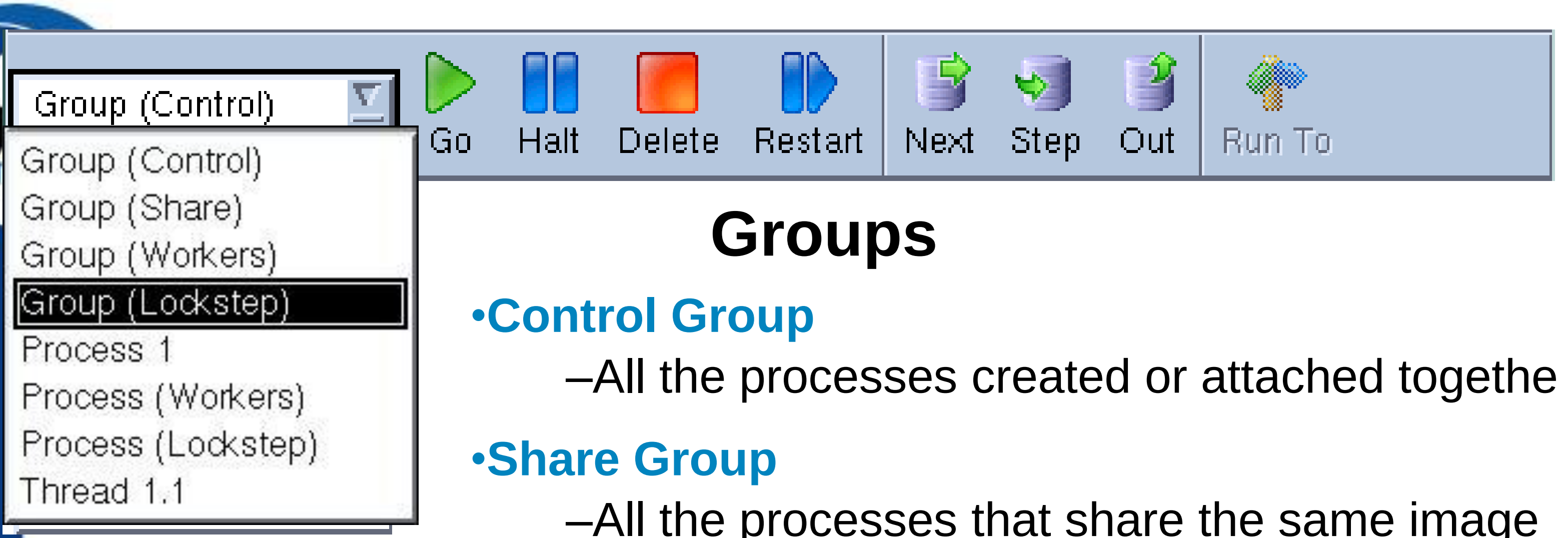


Provides Robust, Scalable and efficient operation with Minimal Program Impact

Process Control Concepts

- Each process window is always focused on a specific process.
- Process focus can be easily switched
  - P+/P-, Dive in Root window and Process tab
- Processes can be ‘held’ - they will not run till unheld.
  - Process > Hold
- Breakpoints can be set to stop the process or the group
- Breakpoint and command scope can be simply controlled

Basic Process Control

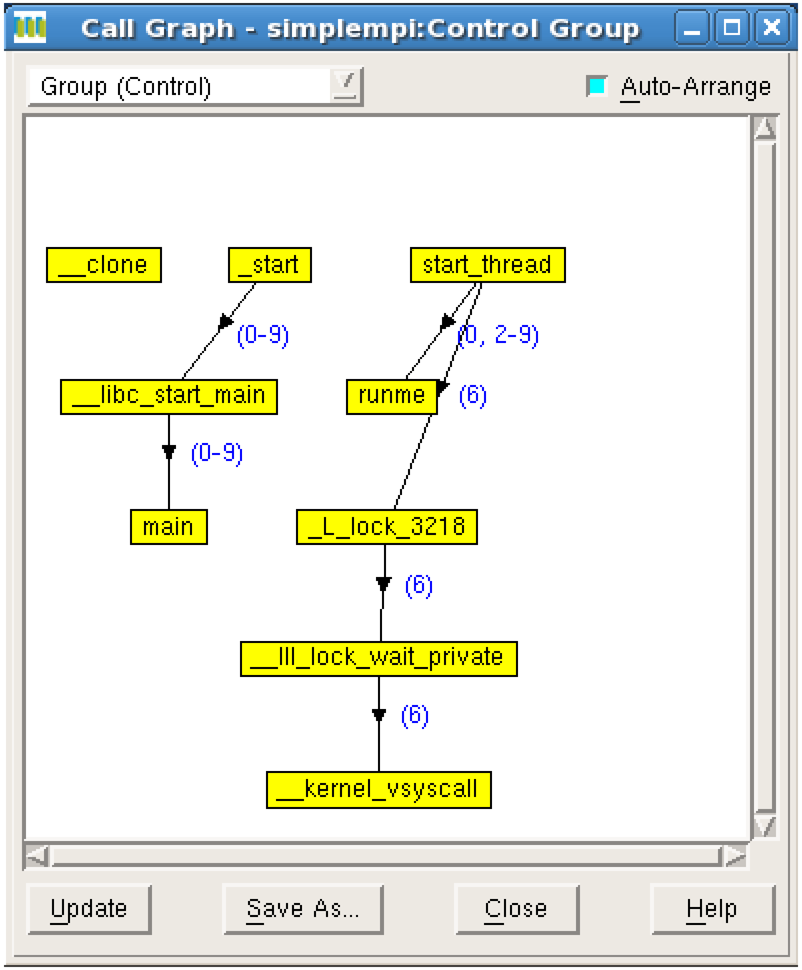


Groups

- **Control Group**
  - All the processes created or attached together
- **Share Group**
  - All the processes that share the same image
- **Workers Group**
  - All the threads that are not recognized as manager or service threads
- **Lockstep Group**
  - All threads at the same PC
- **Process, Process (Workers), Process (Lockstep)**
  - All process members as above
- **User Defined Group**
  - Process group defined in Custom Groups dialog

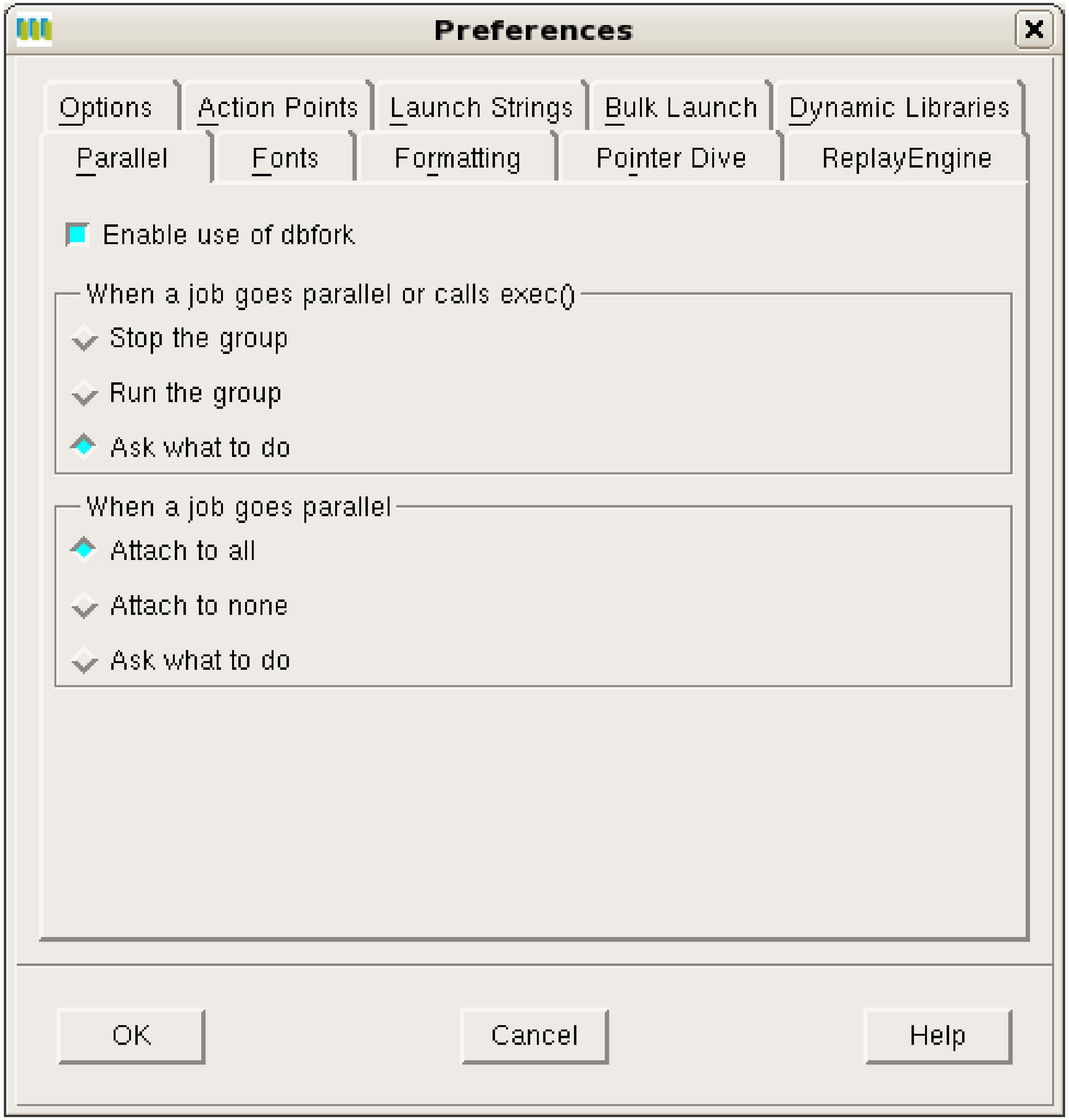
Call Graph

- **Quick view of program state**
  - Each call stack is a path
  - Functions are nodes
  - Calls are edges
    - Labled with the MPI rank
  - Construct process groups
- **Look for outliers**



Dive on a node in the call graph to create a Call Graph group.

Parallel Preferences



☑ Enable use of dbfork

When a job goes parallel or calls exec()

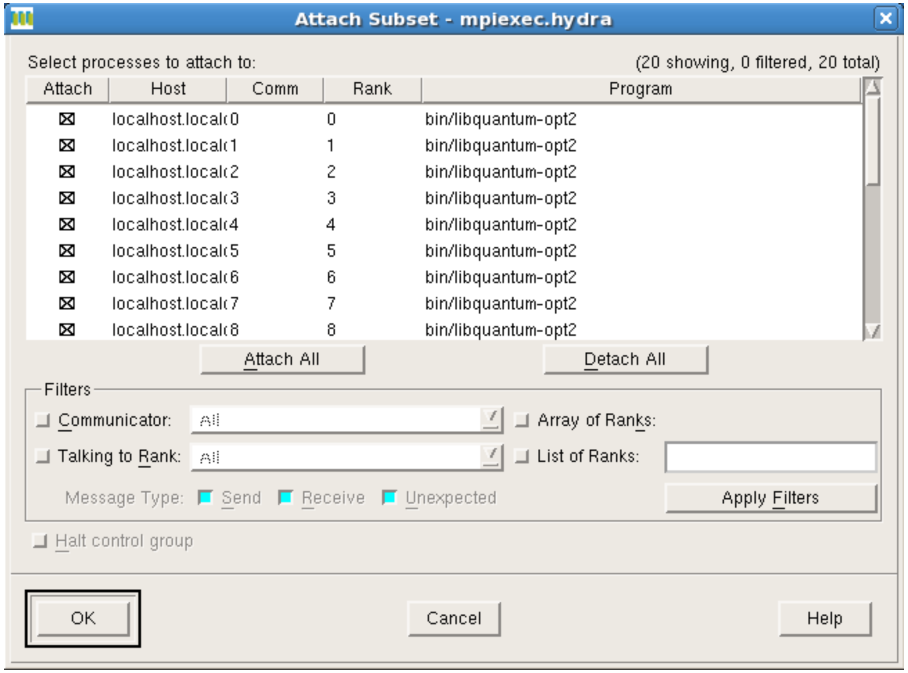
○ Stop the group
○ Run the group
● Ask what to do

When a job goes parallel

● Attach to all
○ Attach to none
○ Ask what to do

Subset Attach

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



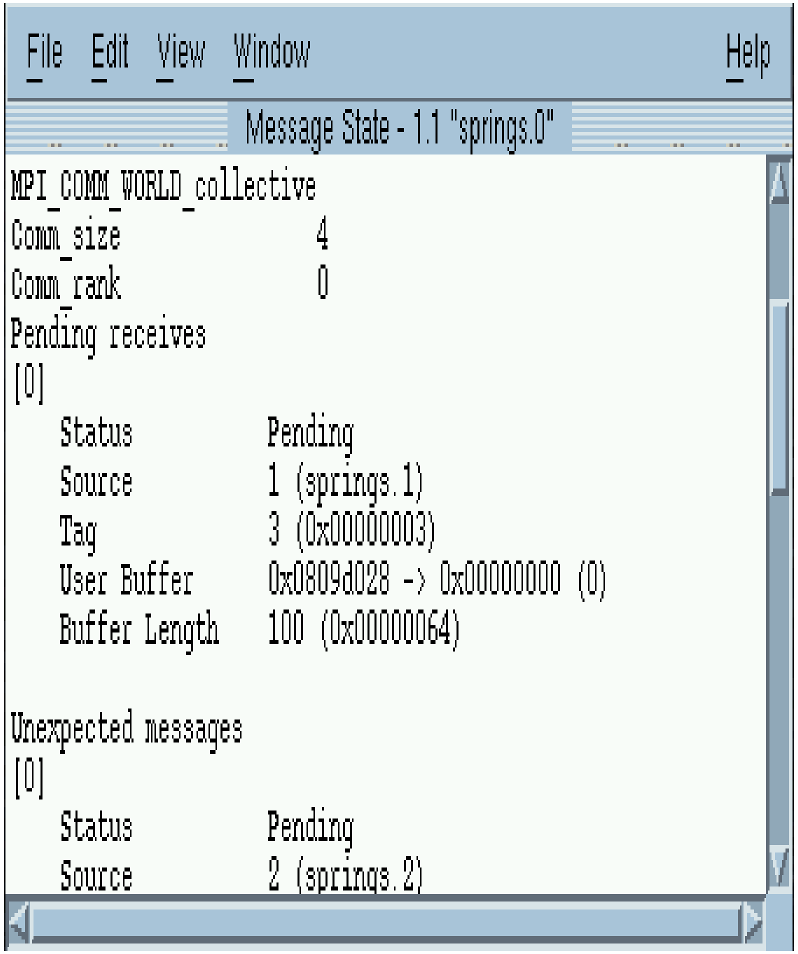
Select processes to attach to: (20 showing, 0 filtered, 20 total)

Attach	Host	Comm	Rank	Program
☒	localhost.local(0		0	bin/libquantum-opt2
☒	localhost.local(1		1	bin/libquantum-opt2
☒	localhost.local(2		2	bin/libquantum-opt2
☒	localhost.local(3		3	bin/libquantum-opt2
☒	localhost.local(4		4	bin/libquantum-opt2
☒	localhost.local(5		5	bin/libquantum-opt2
☒	localhost.local(6		6	bin/libquantum-opt2
☒	localhost.local(7		7	bin/libquantum-opt2
☒	localhost.local(8		8	bin/libquantum-opt2

□ Communicator: All
□ Talking to Rank: All
□ Array of Ranks:
□ List of Ranks:
Message Type: ☑ Send ☑ Receive ☑ Unexpected
□ Halt control group

View MPI Message Queues

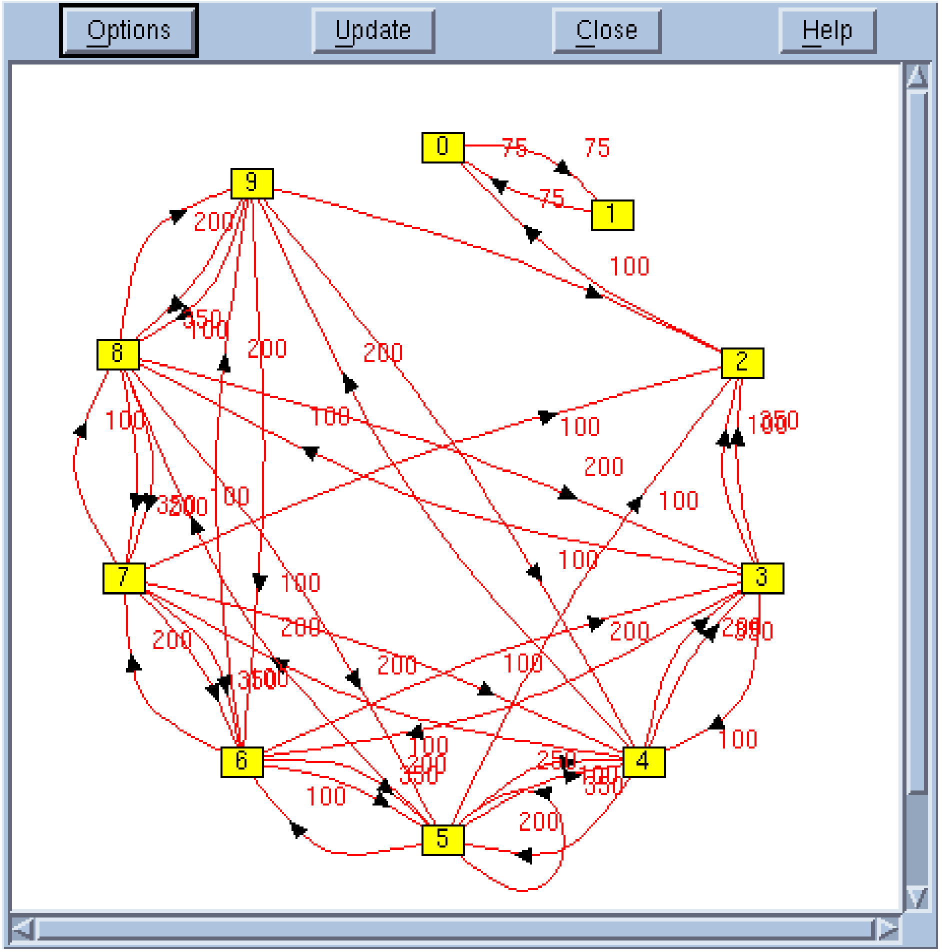
- **Information visible whenever MPI rank processes are halted**
- **Provides information from the MPI layer**
  - Unexpected messages
  - Pending Sends
  - Pending Receives
- **Use this info to debug**
  - Deadlock situations
  - Load balancing
- **May need to be enabled in the MPI library**
  - --enable-debug



```
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

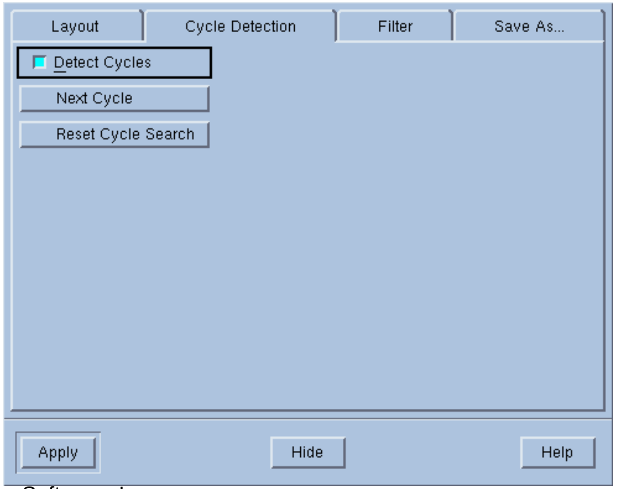
- **Hangs & Deadlocks**
- **Pending Messages**
  - Receives
  - Sends
  - Unexpected
- **Inspect**
  - Individual entries
- **Patterns**

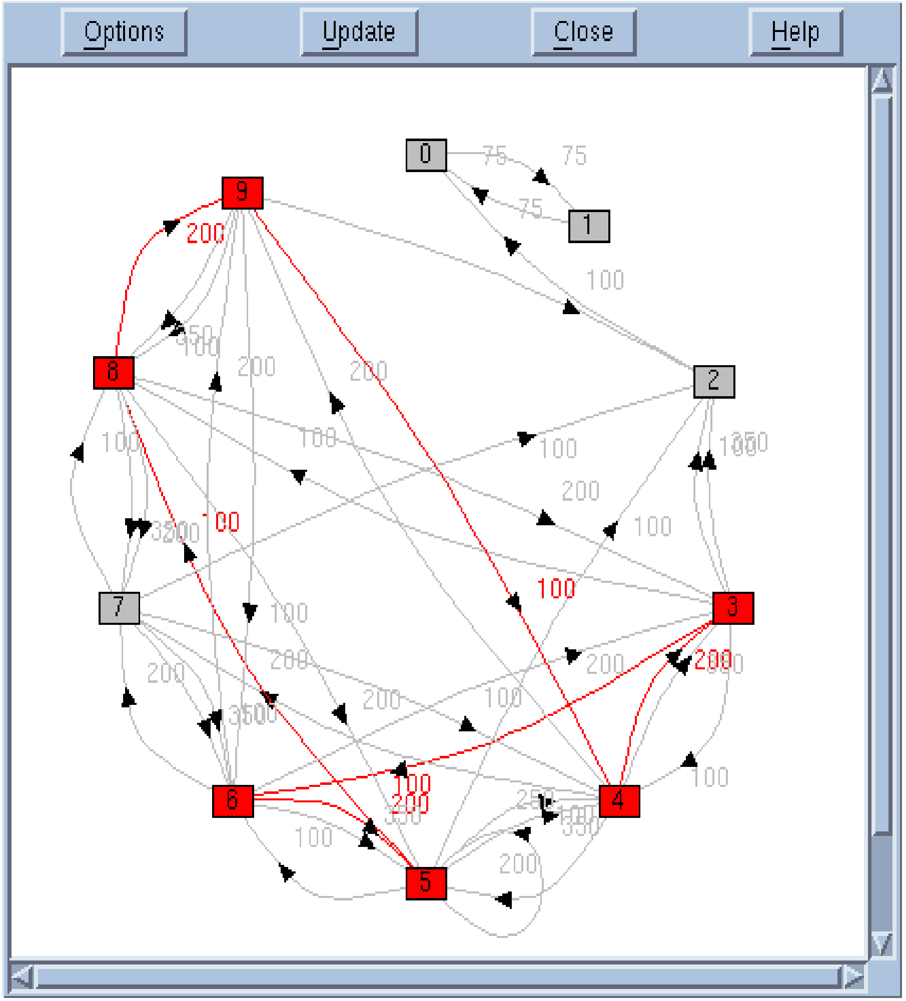


Message Queue Graph

Message Queue Debugging

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





LAB 3: EXAMINING AND CONTROLLING A PARALLEL APPLICATION

MEMORY DEBUGGING WITH MEMORYSCAPE

What is a Memory Bug?

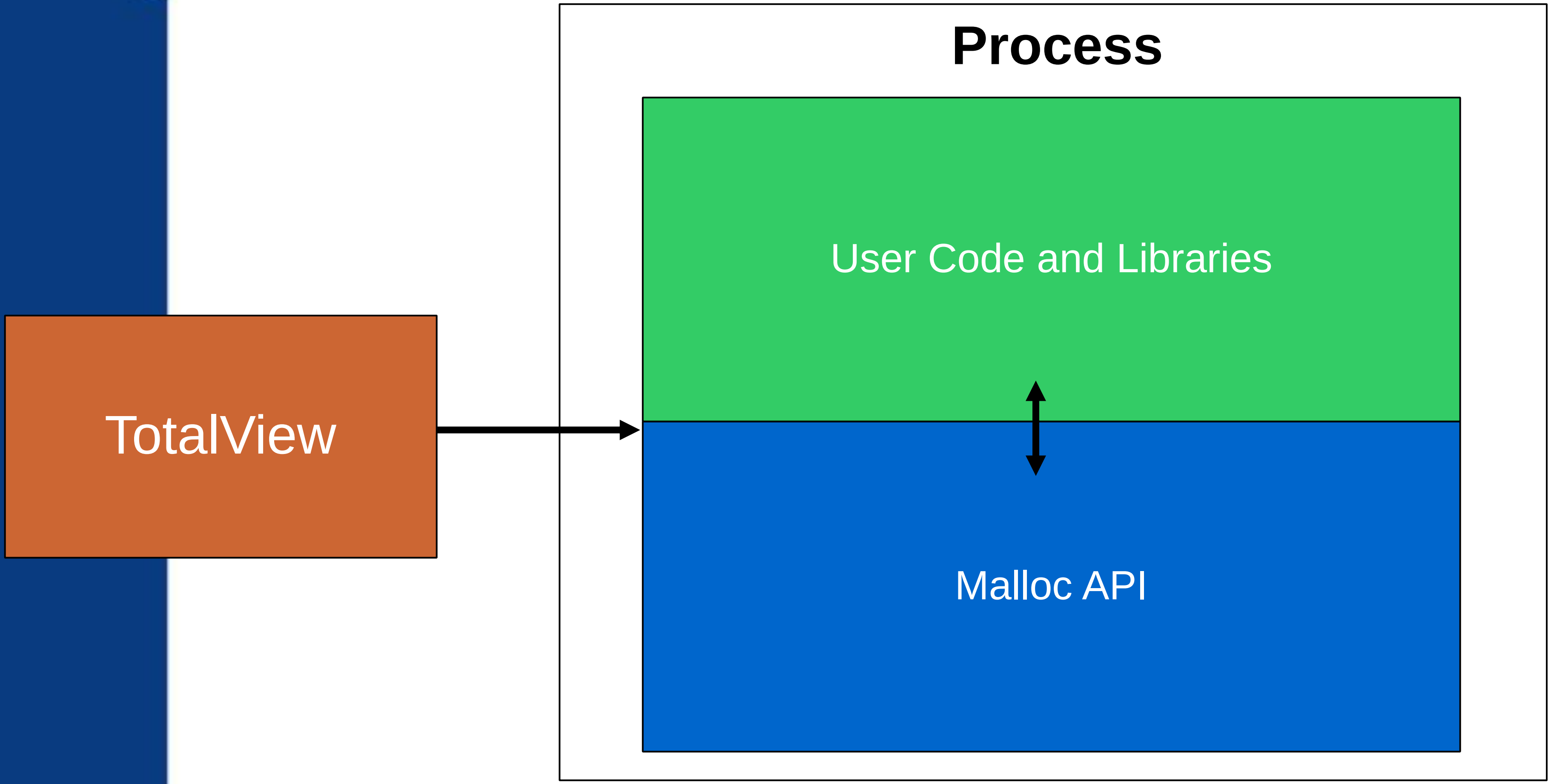
- **A Memory Bug is a mistake in the management of heap memory**
  - Failure to check for error conditions
  - Leaking: Failure to free memory
  - Dangling references: Failure to clear pointers
  - Memory Corruption
    - Writing to memory not allocated
    - Overrunning array bounds

Why Are Memory Bugs Hard to Find?

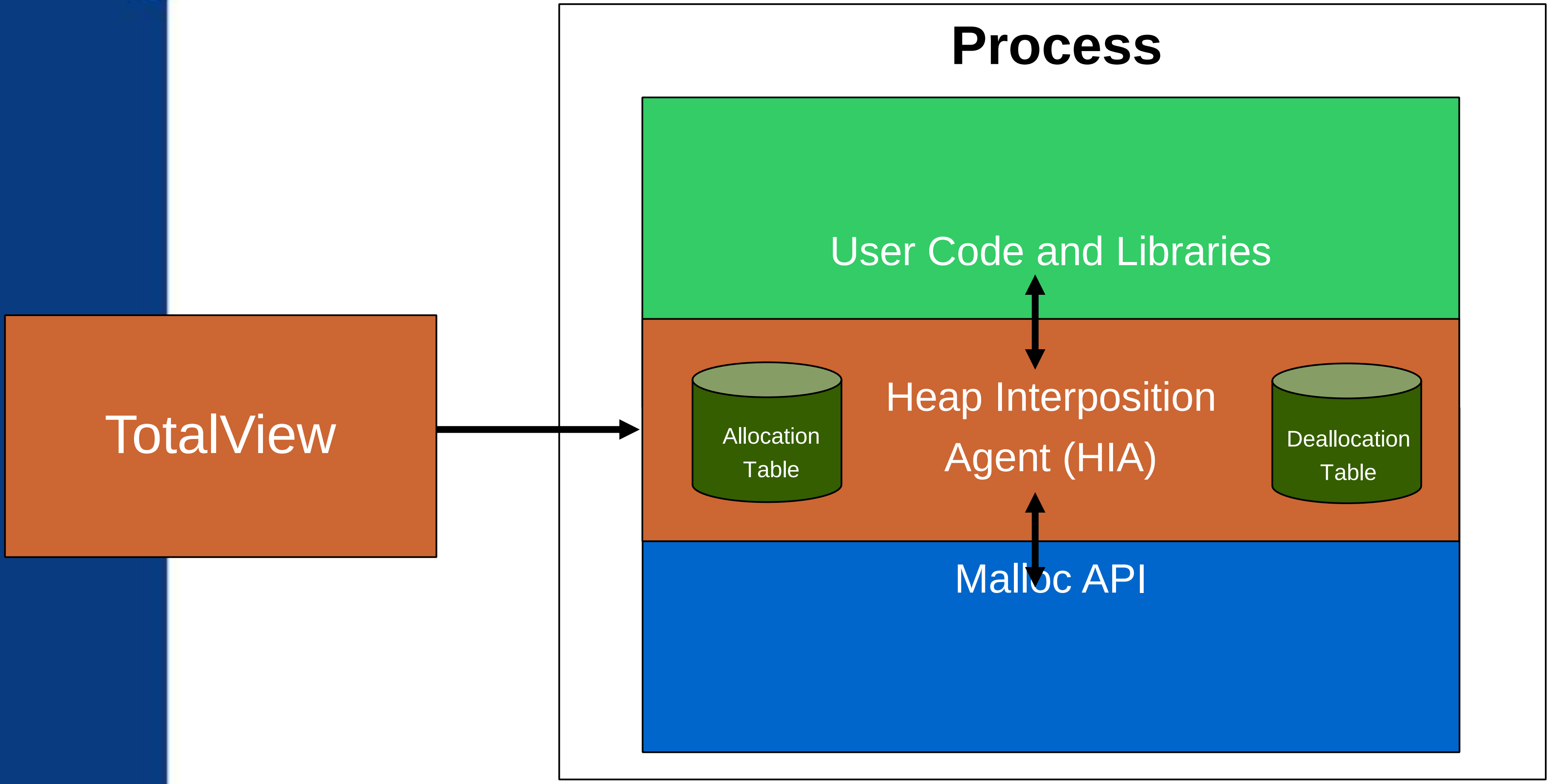
What is a Memory Bug?

- Memory problems can lurk
  - For a given scale or platform or problem, they may not be fatal
  - Libraries could be source of problem
  - The fallout can occur at any subsequent memory access through a pointer
  - The mistake is rarely fatal in and of itself
  - The mistake and fallout can be widely separated
- Potentially 'racy'
  - Memory allocation pattern non-local
  - Even the fallout is not always fatal. It can result in data corruption which may or may not result in a subsequent crash
- May be caused by or cause of a 'classic' bug

The Agent and Interposition



The Agent and Interposition



TotalView HIA Technology

- ## Advantages of TotalView HIA Technology

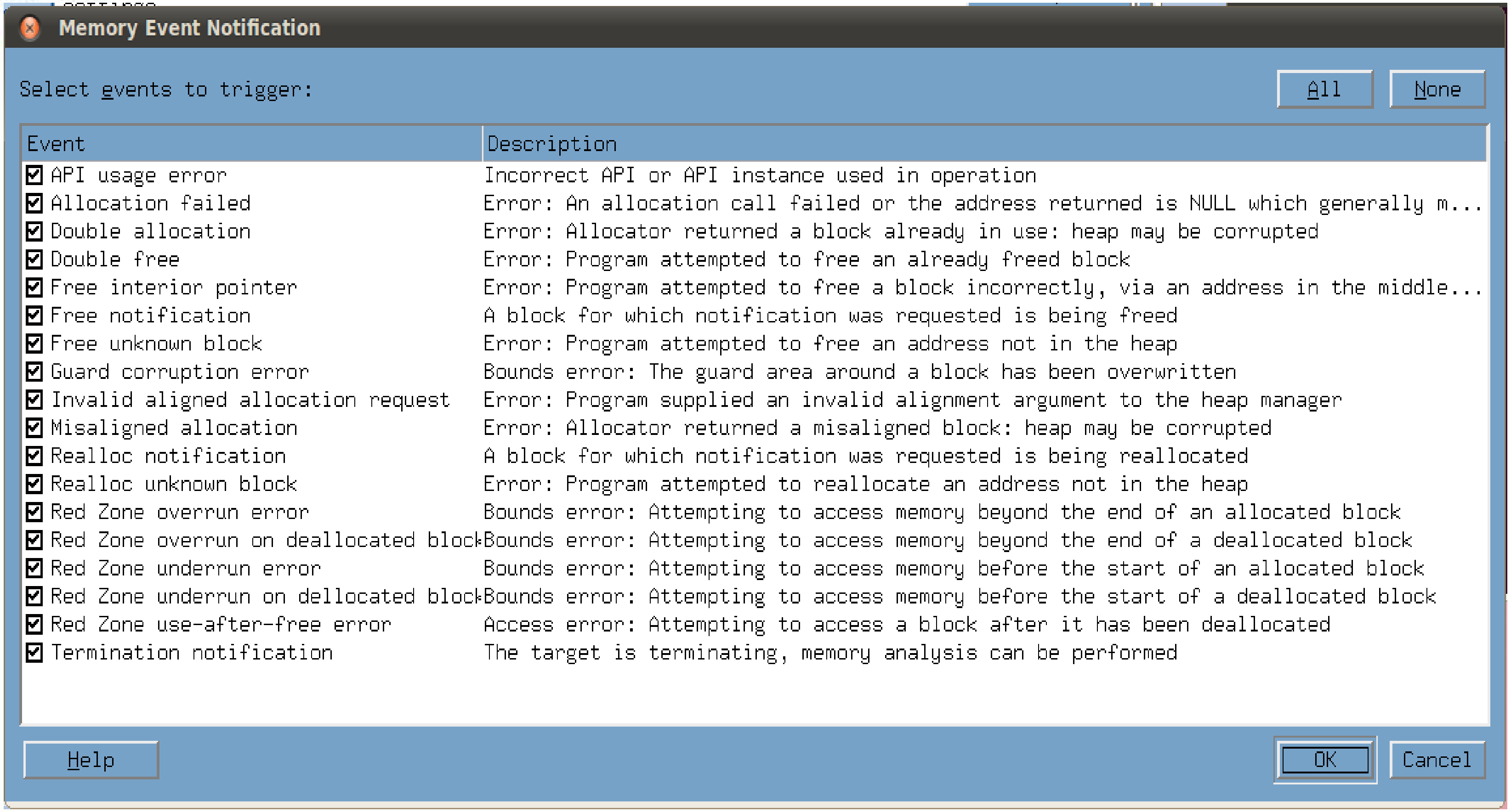
- Use it with your existing builds
  - No Source Code or Binary Instrumentation
- Programs run nearly full speed
  - Low performance overhead
- Low memory overhead
  - Efficient memory usage
- Support wide range of platforms and compilers

Memory Debugger Features

- Automatically detect allocation problems
- View the heap
- Leak detection
- Block painting
- Memory Hoarding
- Dangling pointer detection
- Deallocation/reallocation notification
- Memory Corruption Detection - Guard Blocks
- Memory Comparisons between processes
- Collaboration features

Enabling Memory Debugging

Memory Event Notification

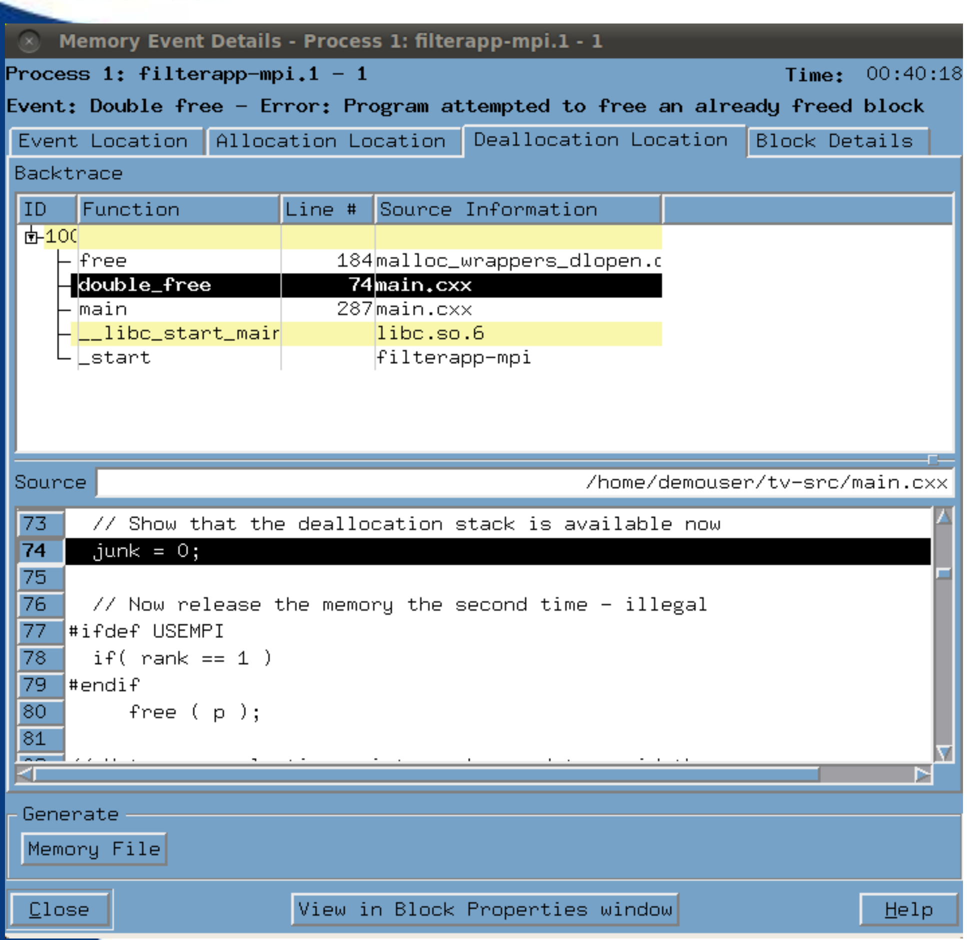


Memory Event Notification

Select events to trigger:

Event	Description
☑ API usage error	Incorrect API or API instance used in operation
☑ Allocation failed	Error: An allocation call failed or the address returned is NULL which generally m...
☑ Double allocation	Error: Allocator returned a block already in use: heap may be corrupted
☑ Double free	Error: Program attempted to free an already freed block
☑ Free interior pointer	Error: Program attempted to free a block incorrectly, via an address in the middle...
☑ Free notification	A block for which notification was requested is being freed
☑ Free unknown block	Error: Program attempted to free an address not in the heap
☑ Guard corruption error	Bounds error: The guard area around a block has been overwritten
☑ Invalid aligned allocation request	Error: Program supplied an invalid alignment argument to the heap manager
☑ Misaligned allocation	Error: Allocator returned a misaligned block: heap may be corrupted
☑ Realloc notification	A block for which notification was requested is being reallocated
☑ Realloc unknown block	Error: Program attempted to reallocate an address not in the heap
☑ Red Zone overrun error	Bounds error: Attempting to access memory beyond the end of an allocated block
☑ Red Zone overrun on deallocated block	Bounds error: Attempting to access memory beyond the end of a deallocated block
☑ Red Zone underrun error	Bounds error: Attempting to access memory before the start of an allocated block
☑ Red Zone underrun on dellocated block	Bounds error: Attempting to access memory before the start of a deallocated block
☑ Red Zone use-after-free error	Access error: Attempting to access a block after it has been deallocated
☑ Termination notification	The target is terminating, memory analysis can be performed

Memory Event Details Window



Memory Event Details - Process 1: filterapp-mpi.1 - 1

Process 1: filterapp-mpi.1 - 1 Time: 00:40:18

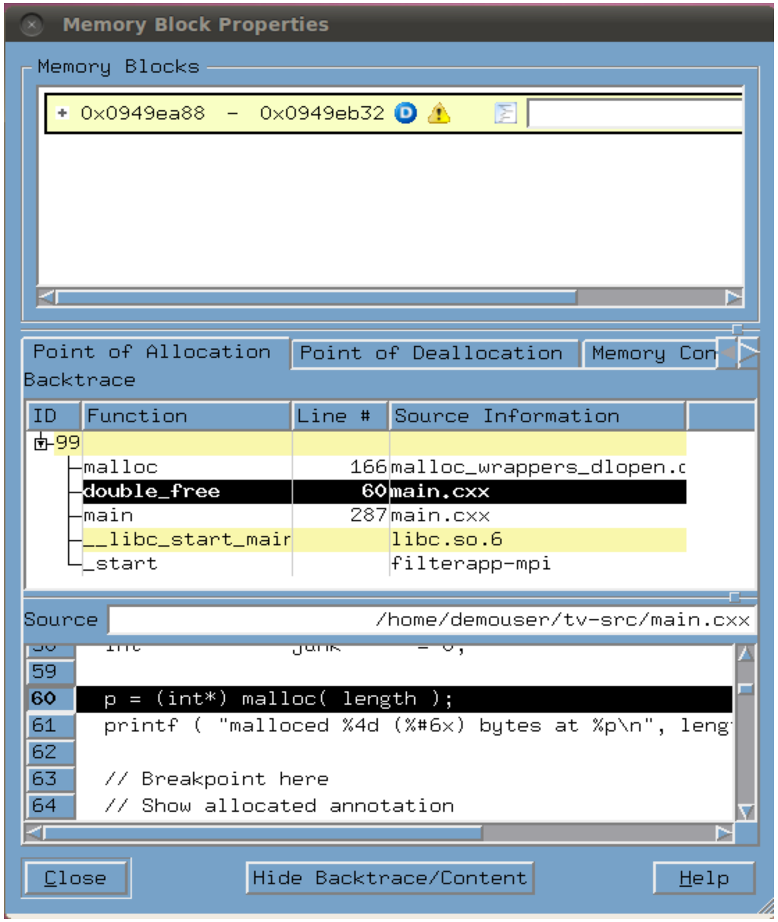
Event: Double free - Error: Program attempted to free an already freed block

Backtrace

ID	Function	Line #	Source Information
10(			
	free	184	malloc_wrappers_dlopen.c
	double_free	74	main.cxx
	main	287	main.cxx
	__libc_start_mair		libc.so.6
	_start		filterapp-mpi

Source /home/demouser/tv-src/main.cxx

```
  // Show that the deallocation stack is available now
  junk = 0;

  // Now release the memory the second time - illegal
#ifdef USEMPI
  if( rank == 1 )
#endif
     free ( p );
```



Memory Block Properties

Memory Blocks

0x0949ea88 – 0x0949eb32

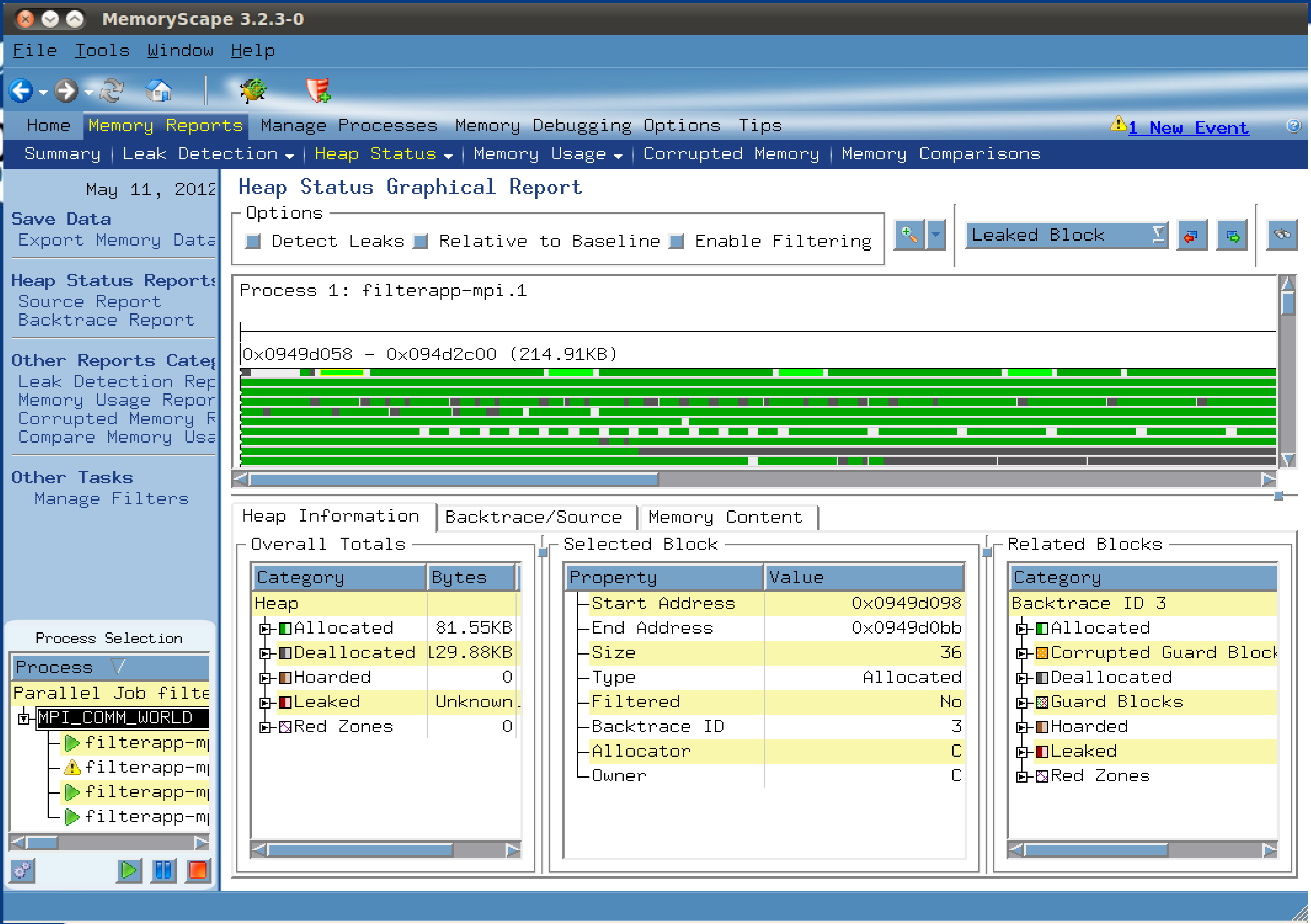
Backtrace

ID	Function	Line #	Source Information
99			
	malloc	166	malloc_wrappers_dlopen.c
	double_free	60	main.cxx
	main	287	main.cxx
	__libc_start_mair		libc.so.6
	_start		filterapp-mpi

Source /home/demouser/tv-src/main.cxx

```

  p = (int*) malloc( length );
  printf ( "malloced %4d (%#6x) bytes at %p\n", leng

  // Breakpoint here
  // Show allocated annotation
```

Heap Graphical View



Options: □ Detect Leaks □ Relative to Baseline □ Enable Filtering

Process 1: filterapp-mpi.1

0x0949d058 - 0x094d2c00 (214.91KB)

Overall Totals

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

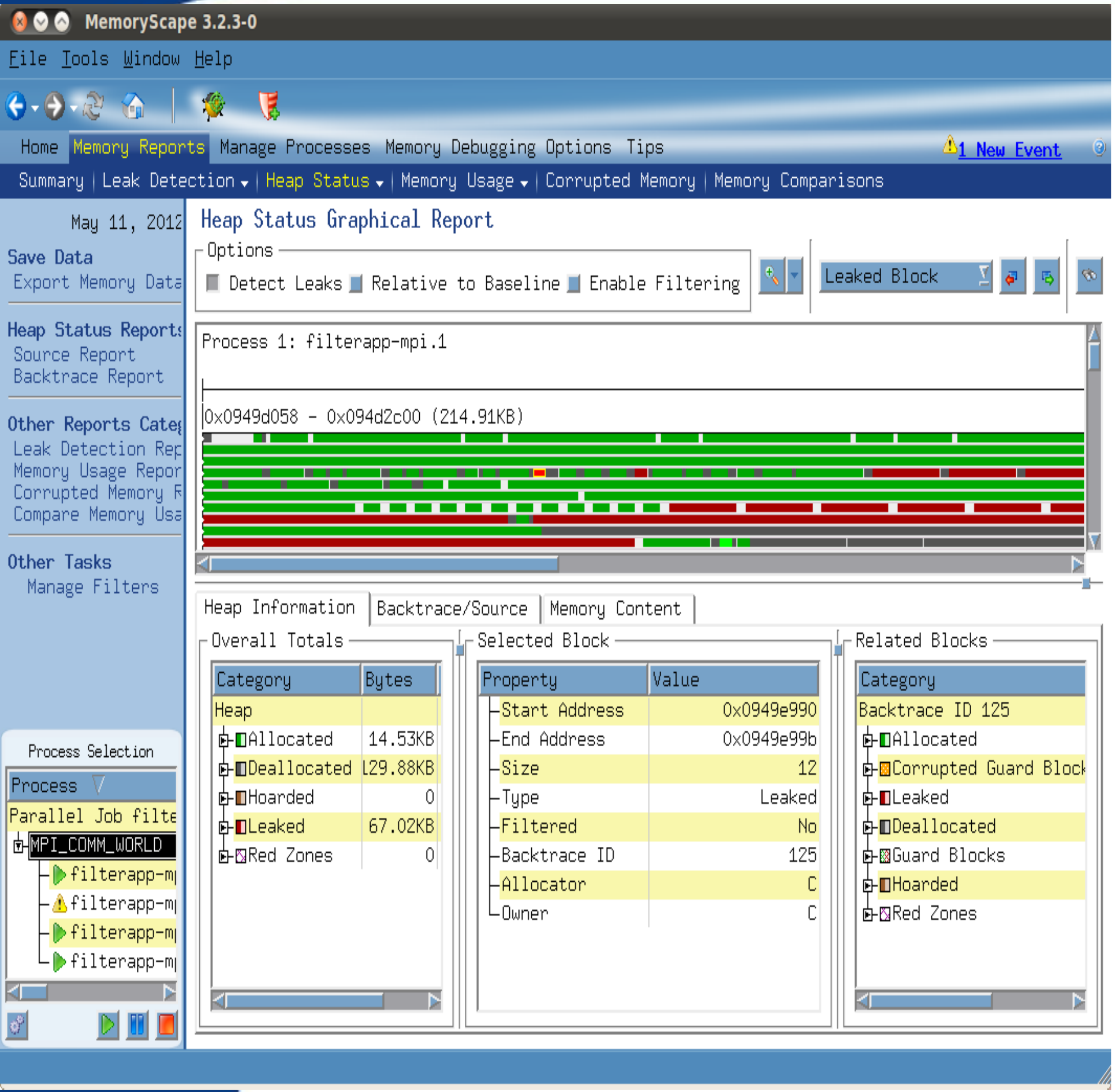
Selected Block

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

Leak Detection



Heap Status Graphical Report

Options

□ Detect Leaks □ Relative to Baseline □ Enable Filtering

Process 1: filterapp-mpi.1

0x0949d058 - 0x094d2c00 (214.91KB)

Overall Totals

Category	Bytes
Heap	
Allocated	14.53KB
Deallocated	129.88KB
Hoarded	0
Leaked	67.02KB
Red Zones	0

Selected Block

Property	Value
Start Address	0x0949e990
End Address	0x0949e99b
Size	12
Type	Leaked
Filtered	No
Backtrace ID	125
Allocator	C
Owner	C

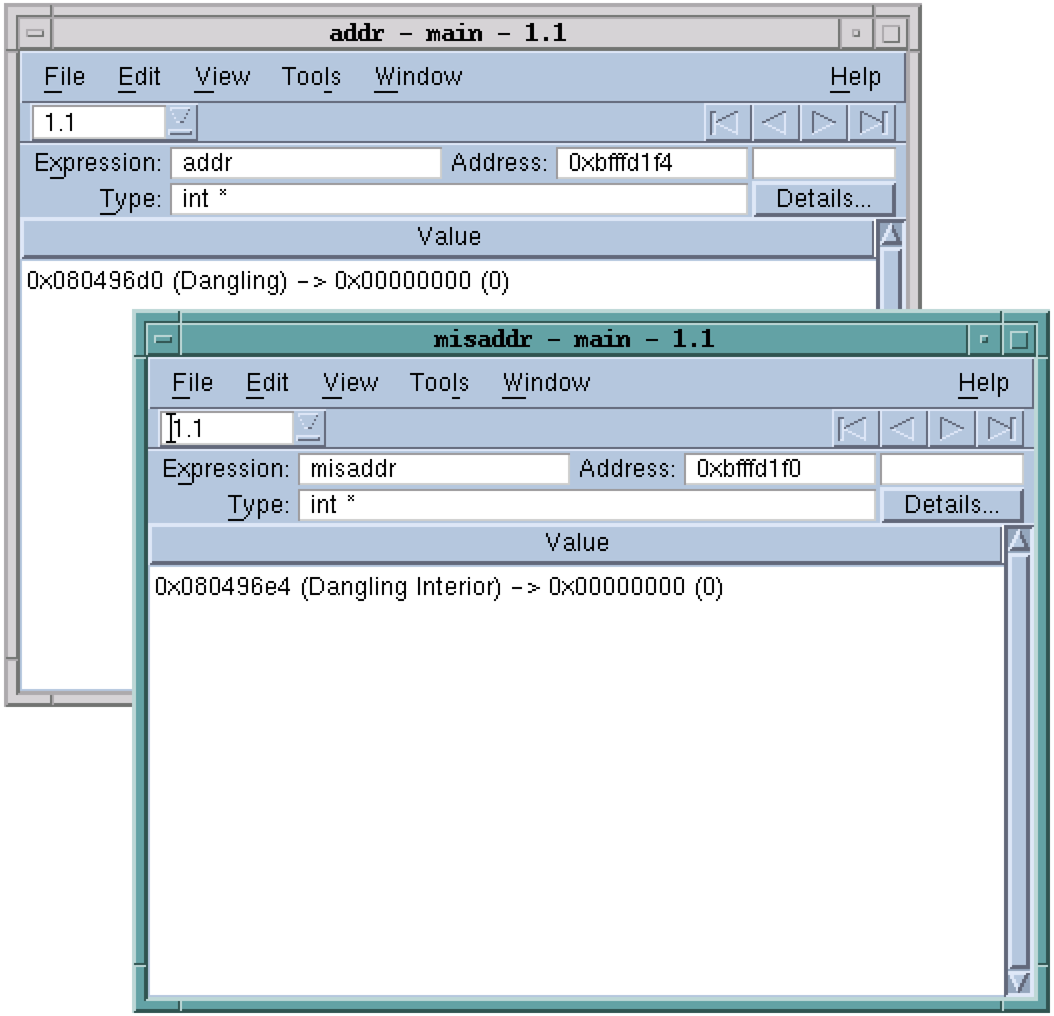
Related Blocks

Category
Backtrace ID 125
Allocated
Corrupted Guard Block
Leaked
Deallocated
Guard Blocks
Hoarded
Red Zones

Leak Detection

- **Based on Conservative Garbage Collection**
- **Can be performed at any point in runtime**
  - **Helps localize leaks in time**
- **Multiple Reports**
  - **Backtrace Report**
  - **Source Code Structure**
  - **Graphically Memory Location**

Dangling Pointer Detection



addr - main - 1.1

1.1

Expression: addr Address: 0xbfffd1f4

Type: int *

0x080496d0 (Dangling) -> 0x00000000 (0)

misaddr - main - 1.1

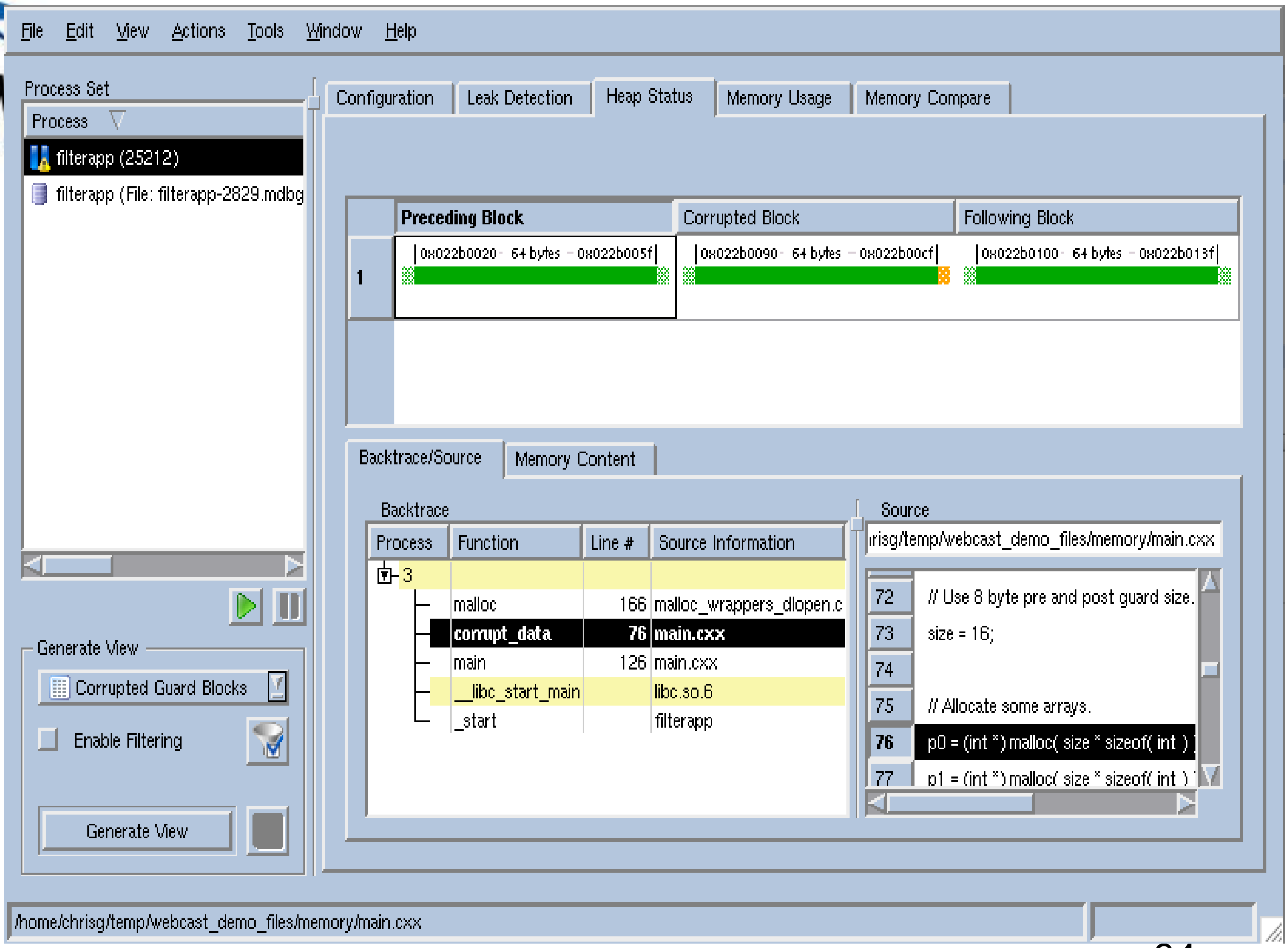
1.1

Expression: misaddr Address: 0xbfffd1f0

Type: int *

0x080496e4 (Dangling Interior) -> 0x00000000 (0)

Memory Corruption Report



Process Set:

- filterapp (25212)
- filterapp (File: filterapp-2829.mdbg

Generate View: Corrupted Guard Blocks

□ Enable Filtering

	Preceding Block	Corrupted Block	Following Block
1	0x022b0020 - 64 bytes - 0x022b005f	0x022b0090 - 64 bytes - 0x022b00cf	0x022b0100 - 64 bytes - 0x022b013f

Backtrace

Process	Function	Line #	Source Information
3			
	malloc	166	malloc_wrappers_dlopen.c
	corrupt_data	76	main.cxx
	main	126	main.cxx
	__libc_start_main		libc.so.6
	_start		filterapp

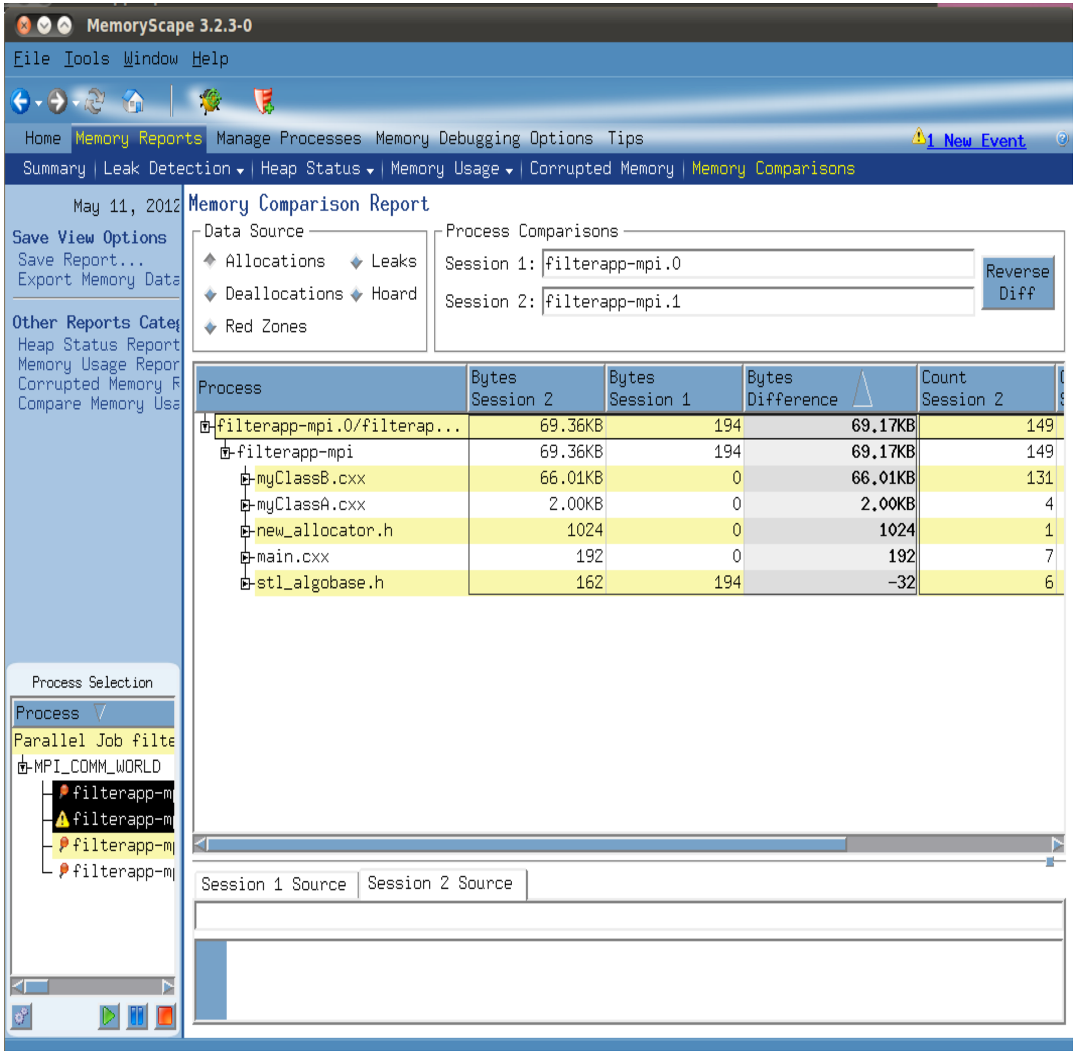
Source: rrisg/temp/webcast_demo_files/memory/main.cxx

```
72   // Use 8 byte pre and post guard size.
73   size = 16;
74
75   // Allocate some arrays.
76   p0 = (int *) malloc( size * sizeof( int ) )
77   p1 = (int *) malloc( size * sizeof( int ) )
```

/home/chrisg/temp/webcast_demo_files/memory/main.cxx

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



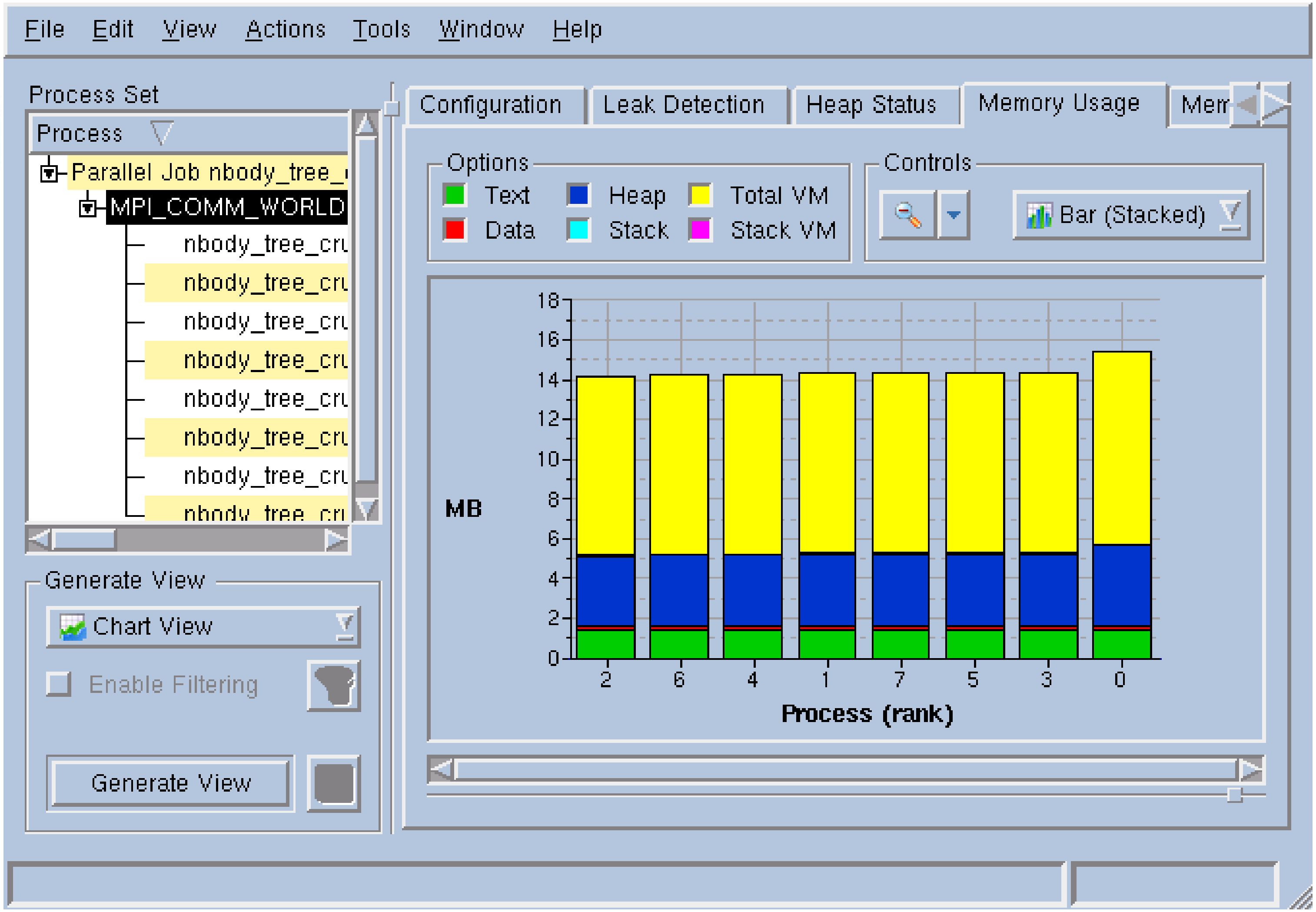
Session 1: filterapp-mpi.0

Session 2: filterapp-mpi.1

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

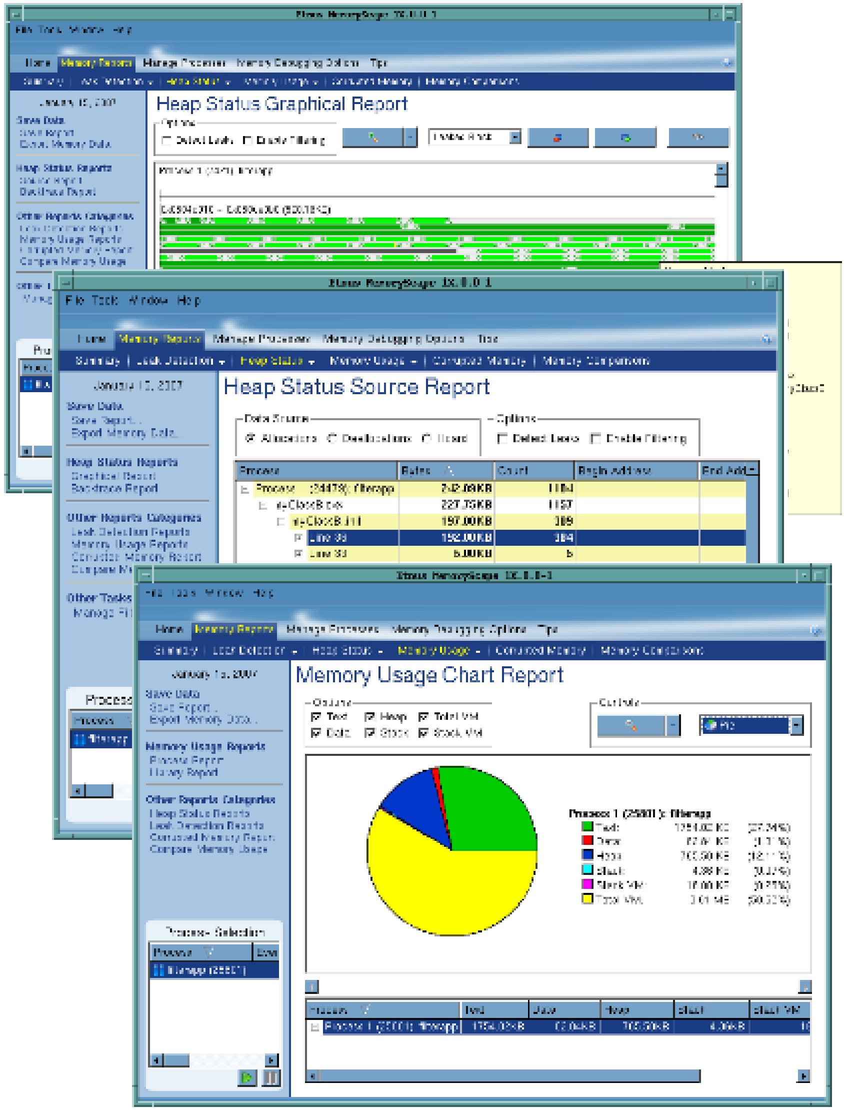
Process
Parallel Job filte
MPI_COMM_WORLD
filterapp-m
filterapp-m
filterapp-m
filterapp-m

Memory Usage Statistics



Memory Reports

- **Multiple Reports**
  - Memory Statistics
  - Interactive Graphical Display
  - Source Code Display
  - Backtrace Display
- **Allow the user to**
  - Monitor Program Memory Usage
  - Discover Allocation Layout
  - Look for Inefficient Allocation
  - Look for Memory Leaks



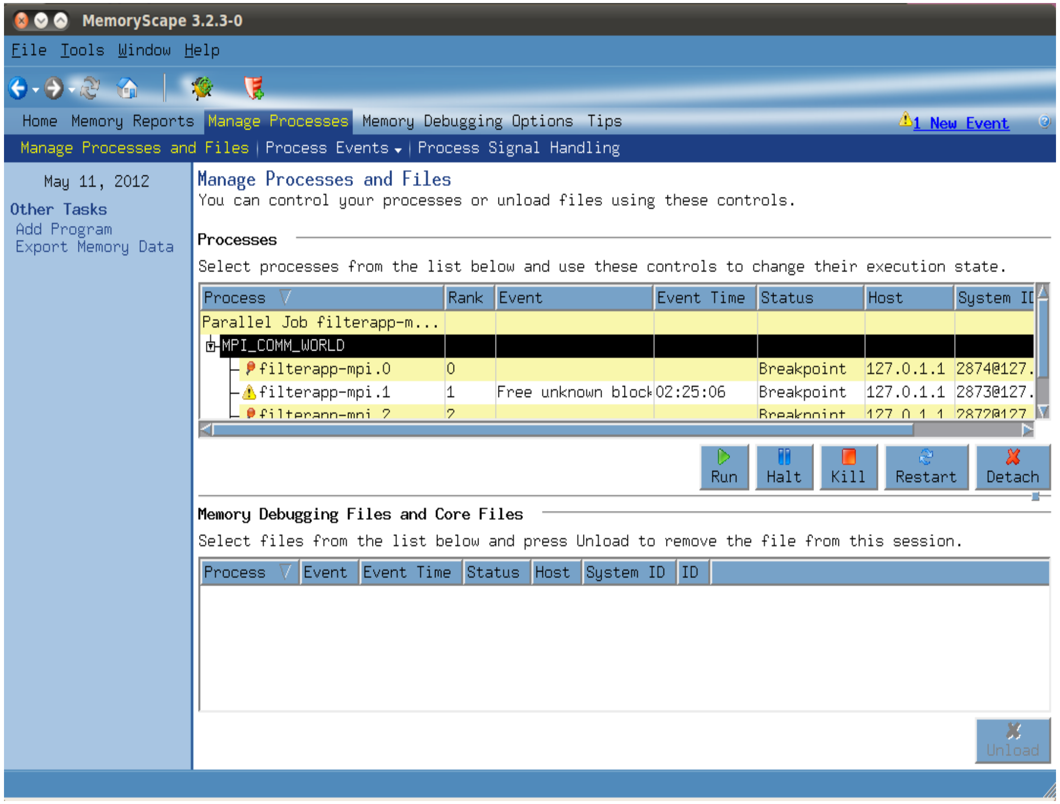
MEMORY DEBUGGING: MEMORYSCAPE

- **Preview: Debugging Memory with MemoryScape**
  - **Startup**
    - Integrated and Bundled with TotalView
    - Typically started from the TotalView gui
  - **Multi-threaded and multi-process programs**
    - Setup from TotalView or stand alone.
    - The multi-process and multi-threaded GUI interface is very similar to TotalView.
- **Automation Support**
- **Block painting**
- **Memory Corruption Detection - Guard Blocks**
- **Memory Hoarding**

Memory Debugging: MemoryScape

Multi-Process and Multi-Thread

- **Memory debug many processes at the same time**
  - MPI
  - Client-Server
  - Fork-Exec
  - Compare two runs
- **Remote applications**
- **Mutli-threaded applications**



MemoryScape 3.2.3-0
File Tools Window Help
Home Memory Reports Manage Processes Memory Debugging Options Tips
1 New Event
Manage Processes and Files | Process Events | Process Signal Handling
May 11, 2012
Other Tasks
Add Program
Export Memory Data

Manage Processes and Files

You can control your processes or unload files using these controls.

Processes

Select processes from the list below and use these controls to change their execution state.

Process	Rank	Event	Event Time	Status	Host	System ID
Parallel Job filterapp-m...						
MPI_COMM_WORLD						
filterapp-mpi.0	0			Breakpoint	127.0.1.1	2874@127.
filterapp-mpi.1	1	Free unknown block	02:25:06	Breakpoint	127.0.1.1	2873@127.
filterapp-mpi.2	2			Breakpoint	127.0.1.1	2872@127

Memory Debugging Files and Core Files

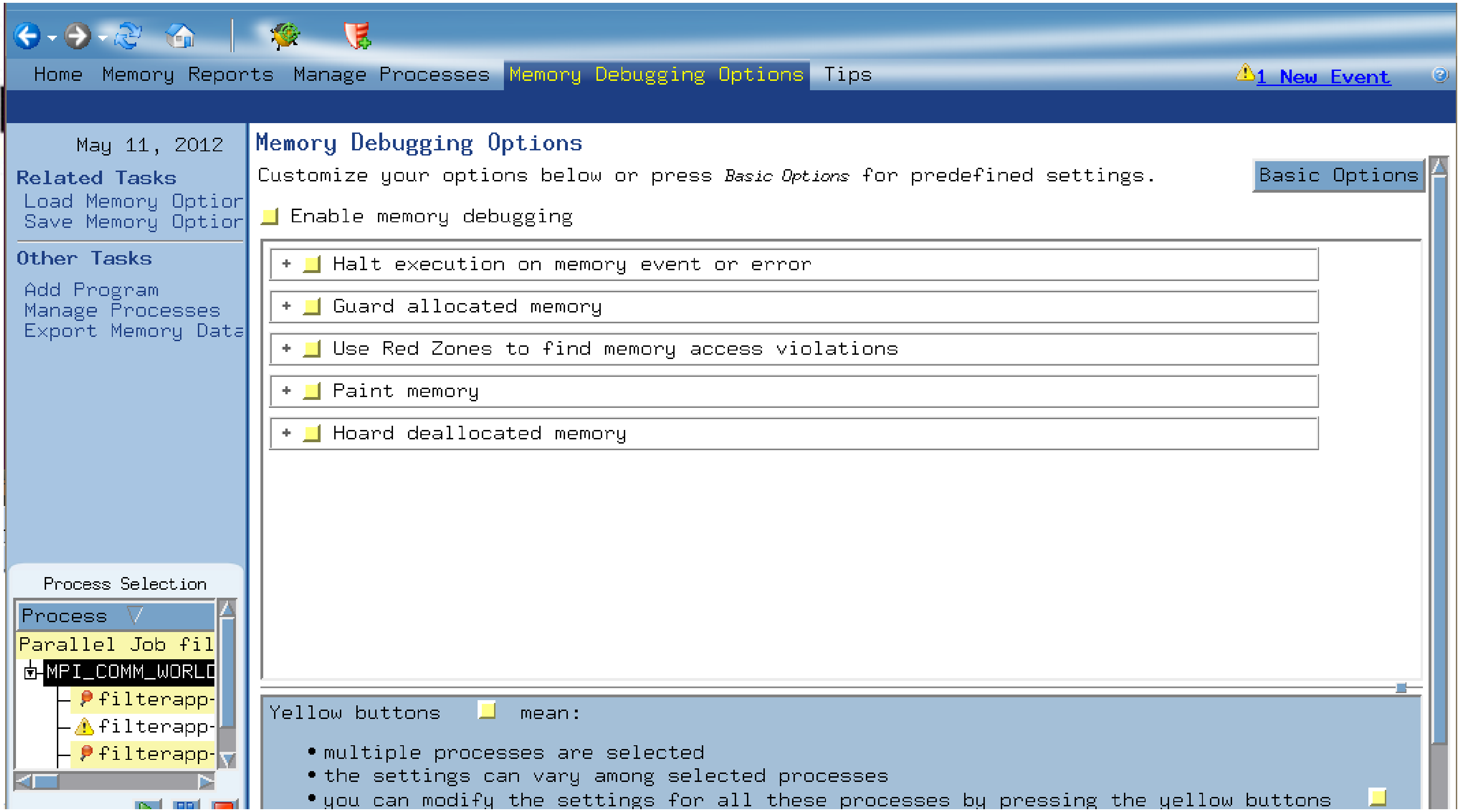
Select files from the list below and press Unload to remove the file from this session.

Process	Event	Event Time	Status	Host	System ID	ID

Script Mode - MemScript

- **Automation Support**
  - MemoryScape lets users run tests and check programs for memory leaks without having to be in front of the program
  - Simple command line program called MemScript
    - Doesn’t start up the GUI
    - Can be run from within a script or test harness
  - The user defines
    - What configuration options are active
    - What thing to look for
    - Actions MemoryScape should take for each type of event that may occur

Memory Debugging: MemoryScape



Memory Debugging Options

Customize your options below or press *Basic Options* for predefined settings.

□ Enable memory debugging

- + □ Halt execution on memory event or error
- + □ Guard allocated memory
- + □ Use Red Zones to find memory access violations
- + □ Paint memory
- + □ Hoard deallocated memory

Yellow buttons □ mean:

- multiple processes are selected
- the settings can vary among selected processes
- you can modify the settings for all these processes by pressing the yellow buttons □

Menu Selections:

Block painting, Guard block and Hoarding

Memory Debugging: MemoryScape

Red Zones instant array bounds detection for Linux

- **Red Zones provides:**
  - 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.

Memory Debugging: MemoryScape

Red Zones instant array bounds detection for Linux

- **Red Zones allocation size range controls**
  - The optional use of Red zones will increase the memory consumption of your program.
  - Controls are provided to allow the full management of Red Zone usage. These controls allow:
    - Restriction of red zones to allocations in several user defined size ranges
    - Easily turning red zones on and off at any time during your programs execution.

Memory Debugging: MemoryScape

Red Zones instant array bounds detection for Linux

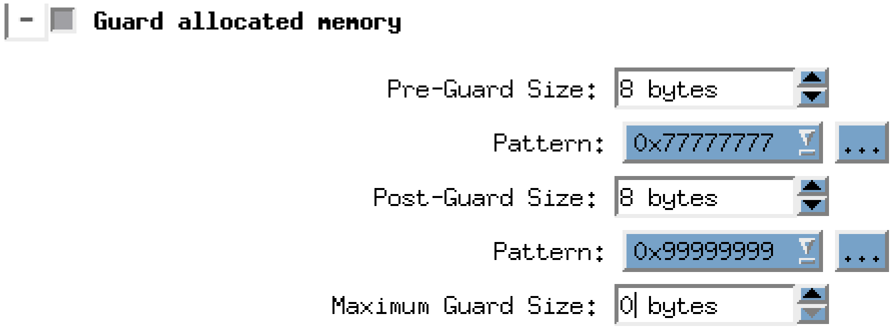
- **Red Zones support in the CLI**
  - The Command Line Interface also provides support for RedZones
- **Scripting support of new commands and command qualifiers**
  - TVScript
  - MemScript

Memory Debugging: MemoryScape

Configuring Guard Blocks

Guard allocated memory

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



- ■ Guard allocated memory

Pre-Guard Size: 8 bytes

Pattern: 0x77777777

Post-Guard Size: 8 bytes

Pattern: 0x99999999

Maximum Guard Size: 0 bytes

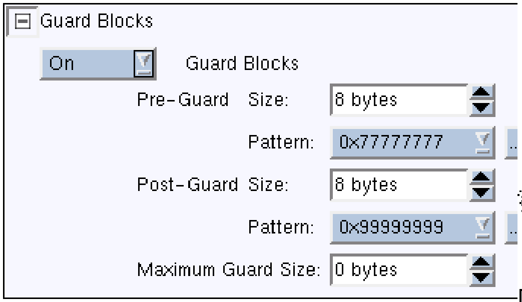
Pre-Guard and Post-Guard Size:

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

Memory Corruption Detection (Guard Blocks)



Guard Blocks

On Guard Blocks

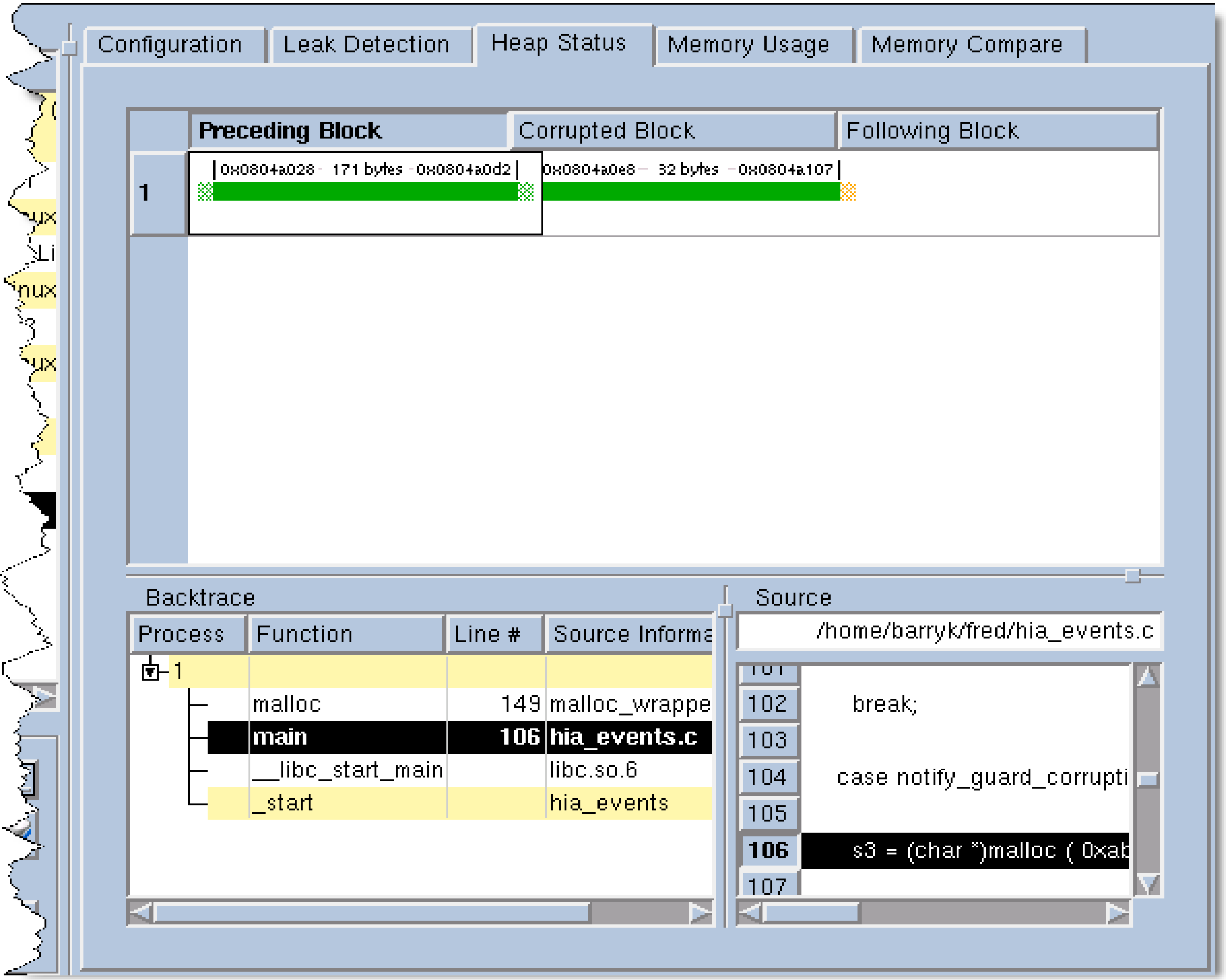
Pre-Guard Size: 8 bytes

Pattern: 0x77777777

Post-Guard Size: 8 bytes

Pattern: 0x99999999

Maximum Guard Size: 0 bytes



	Preceding Block	Corrupted Block	Following Block
1	0x0804a028 - 171 bytes - 0x0804a0d2	0x0804a0e8 - 32 bytes - 0x0804a107	

Backtrace

Process	Function	Line #	Source Informa
1			
	malloc	149	malloc_wrappe
	main	106	hia_events.c
	__libc_start_main		libc.so.6
	_start		hia_events

Source

/home/barryk/fred/hia_events.c

```
101
102        break;
103
104      case notify_guard_corrupti
105
106        s3 = (char *)malloc ( 0xab
107
```

Memory Debugging: MemoryScape

Improved Memory Hoarding support

The use of memory hoarding in MemoryScape increases the risk of running out of available memory. MemoryScape now has the capability to manage this condition and alert you when you are at risk.

- **Hoard Low Memory Controls**
  - Automatically release hoarded memory when available memory gets low, allowing your program to run longer
- **Hoard Low Memory events**
  - MemoryScape can stop execution as notification that the hoard droppped below a particular threshold. This provides an indication that the program is getting close to running out of memory.
- **Hoard Low Memory scripting and CLI support**
  - TVScript
  - MemScript

Memory Debugging: MemoryScape

Improved Memory Hoarding support

- **Hoard Low Memory support in the CLI**
- **Hoard Low Memory scripting support**
  - TVScript
  - MemScript

LAB 4, 5, 6: MEMORY LABS

Using scripts for unattended debugging

BATCH DEBUGGING

tvscript and memscript

- A straightforward language for unattended and/or batch debugging with TotalView and/or MemoryScape
- Usable whenever jobs need to be submitted or batched
- Can be used for automation
- A more powerful version of printf, no recompilation necessary between runs
- Schedule automated debug runs with *cron* jobs
- Expand its capabilities using TCL

Output

- **All of the following information is provided by default for each print**
  - Process id
  - Thread id
  - Rank
  - Timestamp
  - Event/Action description
- **A single output file is written containing all of the information regardless of the number of processes/threads being debugged**

Sample Output

- **Simple interface to create an action point**

  -create_actionpoint ”#85=>print foreign_addr”

- **Sample output with all information**

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Print
!
! Process:
!   ./TVscript_demo (Debugger Process ID:  5, System ID:  2457@127.0.1.1)
! Thread:
!   Debugger ID:  5.1, System ID:  3077191888
! Rank:
!   0
! Time Stamp:
!   05-14-2012 17:11:24
! Triggered from event:
!   actionpoint
! Results:
!    err_detail = {
!      intervals = 0x0000000a (10)
!      almost_pi = 3.1424259850011
!      delta = 0.00083324398852502 3
!    }
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Events

- General
  - any_event
- Source code debugging events
  - actionpoint
  - error
- Memory events (just a few, all are listed in Chapter 4 of TotalView Reference Guide)
  - any_memory_event
  - free_not_allocated
  - guard_corruption
  - rz_overrun, rz_underrun, rz_use_after_free

Actions

- **Source code**
  - display_backtrace [-level num] [numlevels] [options]
  - print [-slice {exp}] {variable | exp}
- **Memory**
  - check_guard_blocks
  - list_allocations
  - list_leaks
  - save_html_heap_status_source_view
  - save_memory_debugging_file
  - save_text_heap_status_source_view

Command syntax

- **General syntax**
  - tvscript [options] [filename] –a [program_args]
- **MPI Options**
  - -mpi *starter*   starter comes from Parallel tab dropdown
  - -starter_args “args for starter program”
  - -nodes
  - -np or –procs or –tasks

Command syntax

- ## Action options
  - -create_actionpoint “src_expr[=>action1[,action2] …]”
    - Repeat on command line for each actionpoint
  - -event_action “event_action_list”
    - event1=action1,event2=action2 or event1=>action1,action2
    - Can repeat on command line for multiple actions
- ## General options
  - -display_specifiers “display_specifiers_list”
  - -maxruntime “hh:mm:ss”
  - -script_file scriptFile
  - -script_log_filename logFilename
  - -script_summary_log_filename summaryLogFilename

Command syntax

- **Memory debugging options**
  - -memory_debugging (must use for debugging memory)
  - -mem_detect_leaks
  - -mem_detect_use_after_free
  - -mem_guard_blocks
  - -mem_hoard_freed_memory
  - -mem_hoard_low_memory_threshold *nnnn*
  - -mem_paint_all
  - -mem_paint_on_alloc
  - -mem_paint_on_dealloc

Command syntax

- Memory debugging red zone options
  - -mem_red_zone_overruns
  - -mem_red_zones_size_ranges min:max[,min:max]…
    - Ranges can be
      - min:max
      - min:
      - :max
      - fixed
  - -mem_red_zones_underruns

Script Files

- **Instead of putting everything on the command line, you can also write and use script files**
- **Script files can also include TCL**
- **Logging functions**
  - tvscript_log *msg* – logs msg to the log file
  - tvscript_slog *msg* – logs msg to the summary log file
- **Property functions**
  - tvscript_get_process_property *process_id property*
  - tvscript_get_thread_property *thread_id property*

Script Files

- **Action point and event functions**
  - tvscript_create_actionpoint *source_loc_expr*
    - [[##image#]filename#]line_number
    - *function_name*
    - class *class_name*
    - virtual *class:signature*
  - tvscript_add_actionpoint_handler *id handler*
  - tvscript_add_event_handler *event handler*
    - Passes an array to handler, event will either be error or actionpoint
    - Other information relevant to event
- **Handlers are written in TCL**

LAB 7: BATCH MODE DEBUGGING WITH TVSCRIPT

REVERSE DEBUGGING WITH REPLAYENGINE

What is ReplayEngine?

- **Provides record for deterministic replay**
- **Records program changes as they happen**
- **Captures input**
  - Function calls
  - Network and file I/O
- **Captures non-determinism**
  - Forces single threaded execution
  - Records context switches
- **Allows stepping back in execution, like a DVR for your programs**
- **Use breakpoints and watchpoints**
- **Support for MPI on Ethernet, Infiniband, Cray XE Gemini**
- **Support for Pthreads, and OpenMP**

ReplayEngine Support

- **Replay on Demand: enable it when you want it**
- **Supported on Linux for x86 and x86_64**
- **Cluster interconnects**
  - IP (any interconnect): MPICH, MPICH2, OpenMPI, Intel MPI, SGI MPT, Cray XT-MPT, MVAPICH, MVAPICH2
  - Mellanox Infiniband
    - IB verb: MVAPICH, MVAPICH2, OpenMPI, Intel MPI
  - Qlogic Infiniband
    - PSM: MVAPICH, MVAPICH2, OpenMPI, Intel MPI

ReplayEngine

- **Editing during record mode**
  - Allows modification of variables during record mode (eval breakpoints, click/edit of variable values)
  - Modifications are recorded along with the rest of the execution
  - Not allowed to change values when in playback mode
  - Don’t attempt to step into recorded edits, but correct values show up on either side

ReplayEngine

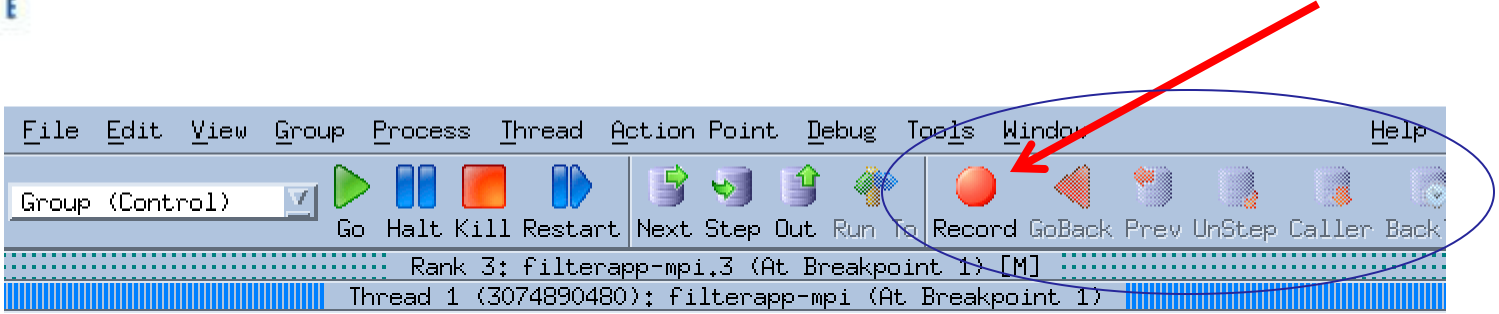
An Intuitive User Interface

Next	Step forward over functions	Prev	Step *backward* over functions
Step	Step forward into functions	UnStep	Step *backward* into functions
Out	Advance forward out of current Function, after the call	Caller	*Advance backward* out of current Function, to before the call
Run To	Advance forward to selected line	BackTo	Advance backward to selected line
Live	Advance forward to "live" session		

ReplayEngine

Replay not running

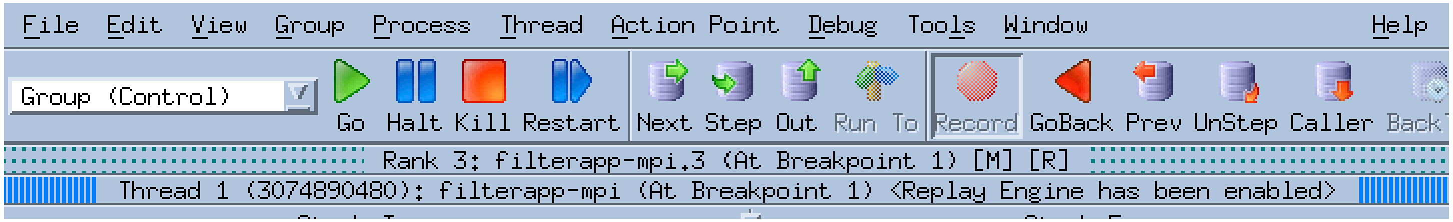
Enable Replay “On Demand”



```
Rank 3: filterapp-mpi.3 (At Breakpoint 1) [M]
Thread 1 (3074890480): filterapp-mpi (At Breakpoint 1)
```

No Active Buttons!

Replay running



```
Rank 3: filterapp-mpi.3 (At Breakpoint 1) [M] [R]
Thread 1 (3074890480): filterapp-mpi (At Breakpoint 1) <Replay Engine has been enabled>
```

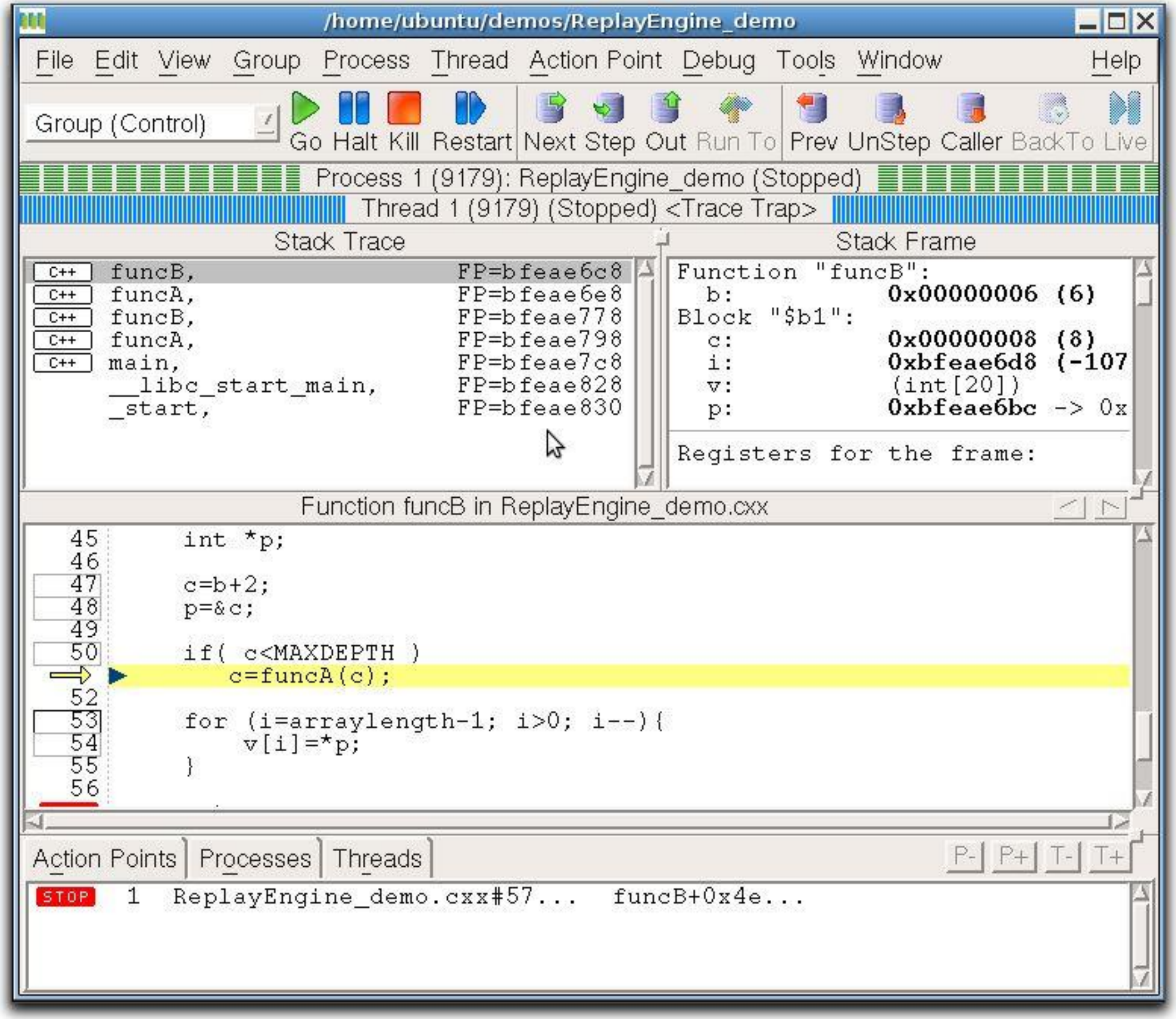
Active Buttons

Example

ReplayEngine

Consider the following very difficult program scenario:

- A crash occurs that destroys the stack backtrace, giving no information leading up to the problem
- ReplayEngine can be used to work backwards from the crash, and even to observe the stack recreate itself, providing the critical information on where and how the problem began.
- The ReplayEngine provides the ability to review any part of the program execution… to see all variables and function calls, from the beginning of the run to the current time



Process 1 (9179): ReplayEngine_demo (Stopped)

Thread 1 (9179) (Stopped) <Trace Trap>

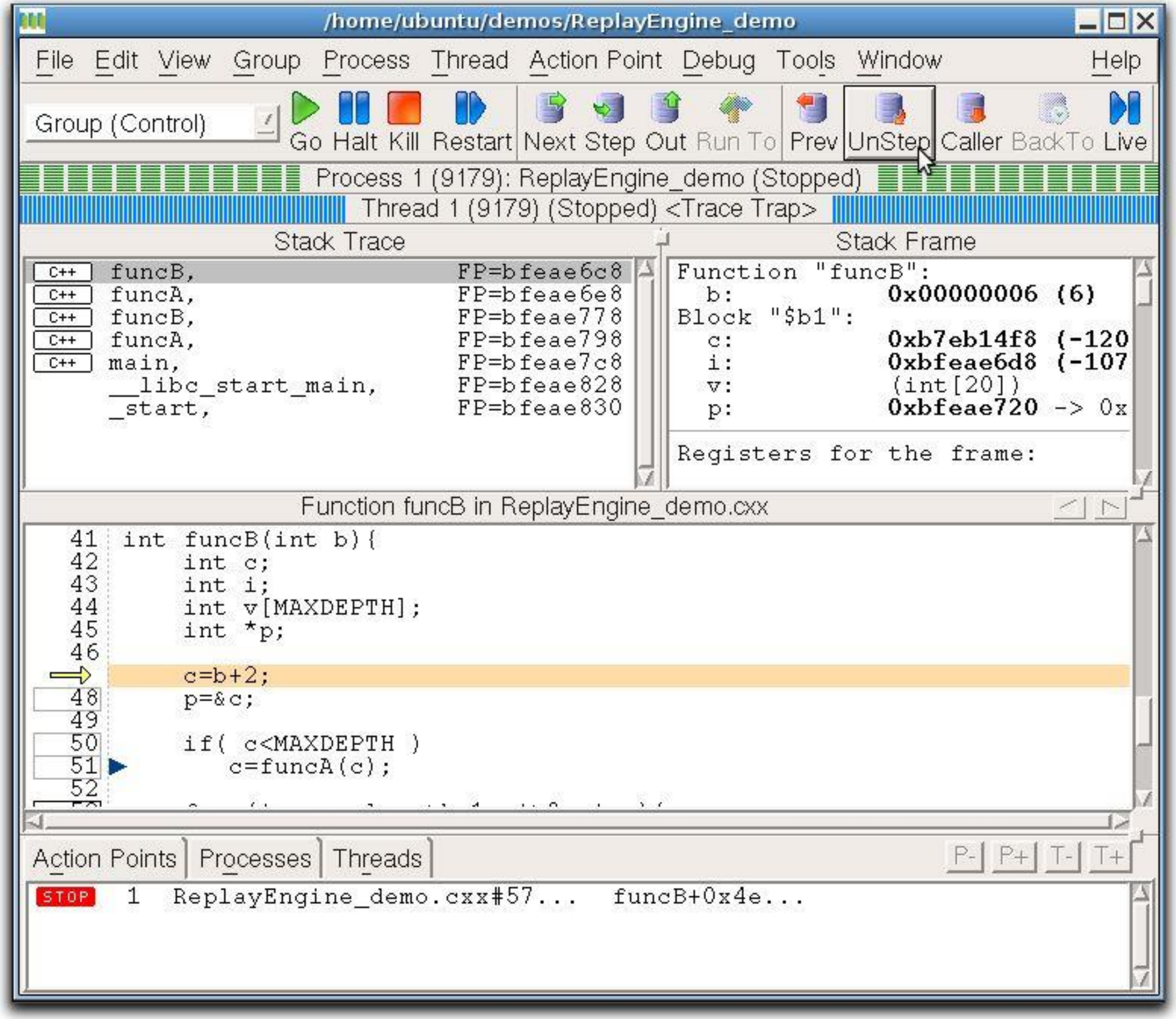
```
C++  funcB,                 FP=bfeae6c8
C++  funcA,                 FP=bfeae6e8
C++  funcB,                 FP=bfeae778
C++  funcA,                 FP=bfeae798
C++  main,                  FP=bfeae7c8
     __libc_start_main,     FP=bfeae828
     _start,                FP=bfeae830
```

```
Function "funcB":
  b:          0x00000006 (6)
Block "$b1":
  c:          0x00000008 (8)
  i:          0xbfeae6d8 (-107
  v:          (int[20])
  p:          0xbfeae6bc -> 0x

Registers for the frame:
```

Function funcB in ReplayEngine_demo.cxx

```
    int *p;

    c=b+2;
    p=&c;

    if( c<MAXDEPTH )
       c=funcA(c);

    for (i=arraylength-1; i>0; i--){
       v[i]=*p;
    }
```

```
STOP   1   ReplayEngine_demo.cxx#57...   funcB+0x4e...
```


/home/ubuntu/demos/ReplayEngine_demo
File Edit View Group Process Thread Action Point Debug Tools Window Help
Group (Control)
Go Halt Kill Restart Next Step Out Run To Prev UnStep Caller BackTo Live

Process 1 (9179): ReplayEngine_demo (Stopped)

Thread 1 (9179) (Stopped) <Trace Trap>

Stack Trace

```
C++  funcB,               FP=bfeae6c8
C++  funcA,               FP=bfeae6e8
C++  funcB,               FP=bfeae778
C++  funcA,               FP=bfeae798
C++  main,                FP=bfeae7c8
     __libc_start_main,   FP=bfeae828
     _start,              FP=bfeae830
```

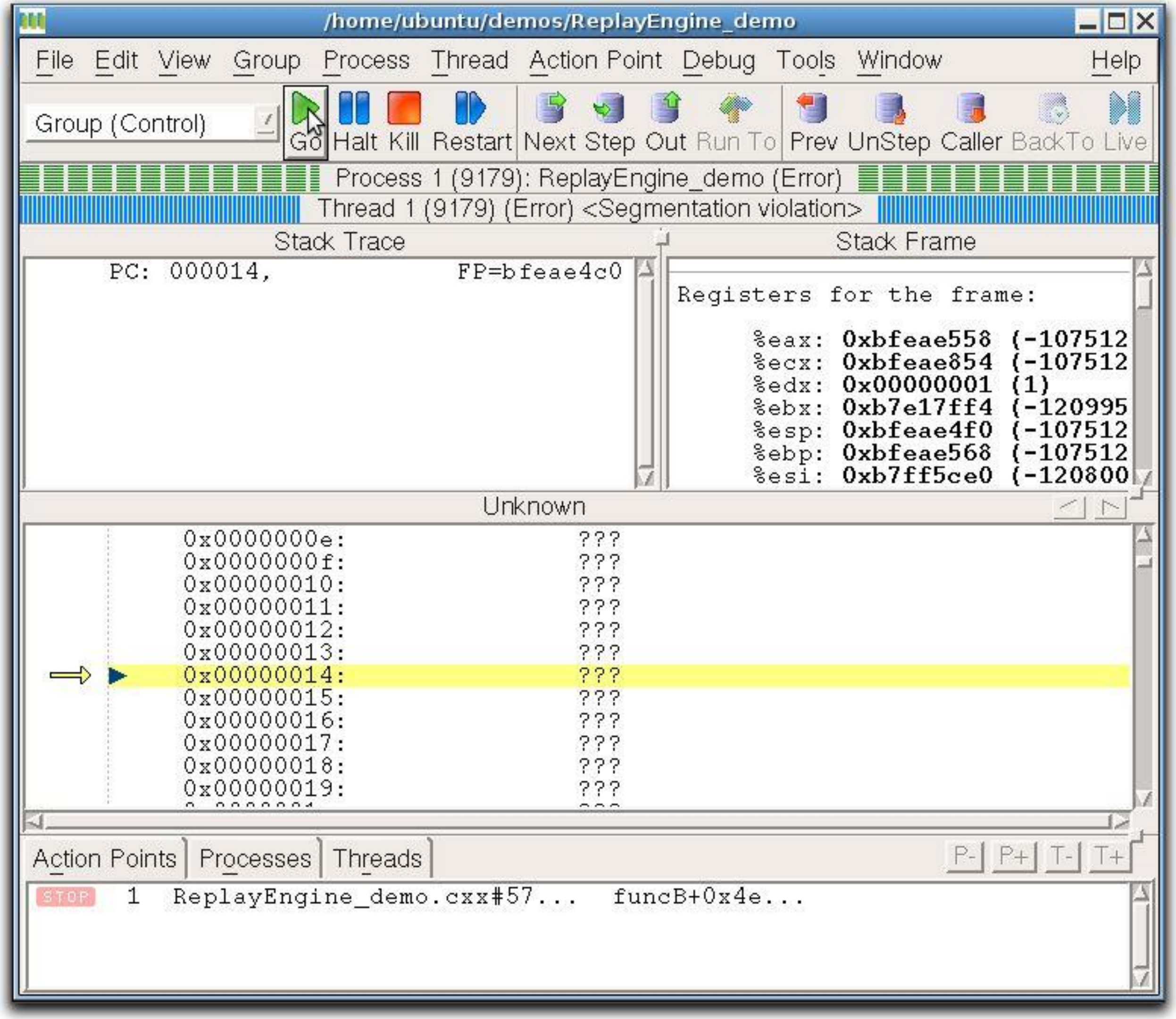
Stack Frame

```
Function "funcB":
  b:             0x00000006 (6)
Block "$b1":
  c:             0xb7eb14f8 (-120
  i:             0xbfeae6d8 (-107
  v:             (int[20])
  p:             0xbfeae720 -> 0x

Registers for the frame:
```

Function funcB in ReplayEngine_demo.cxx

```
int funcB(int b){
    int c;
    int i;
    int v[MAXDEPTH];
    int *p;

    c=b+2;
    p=&c;

    if( c<MAXDEPTH )
       c=funcA(c);
```

```
STOP  1  ReplayEngine_demo.cxx#57...  funcB+0x4e...
```



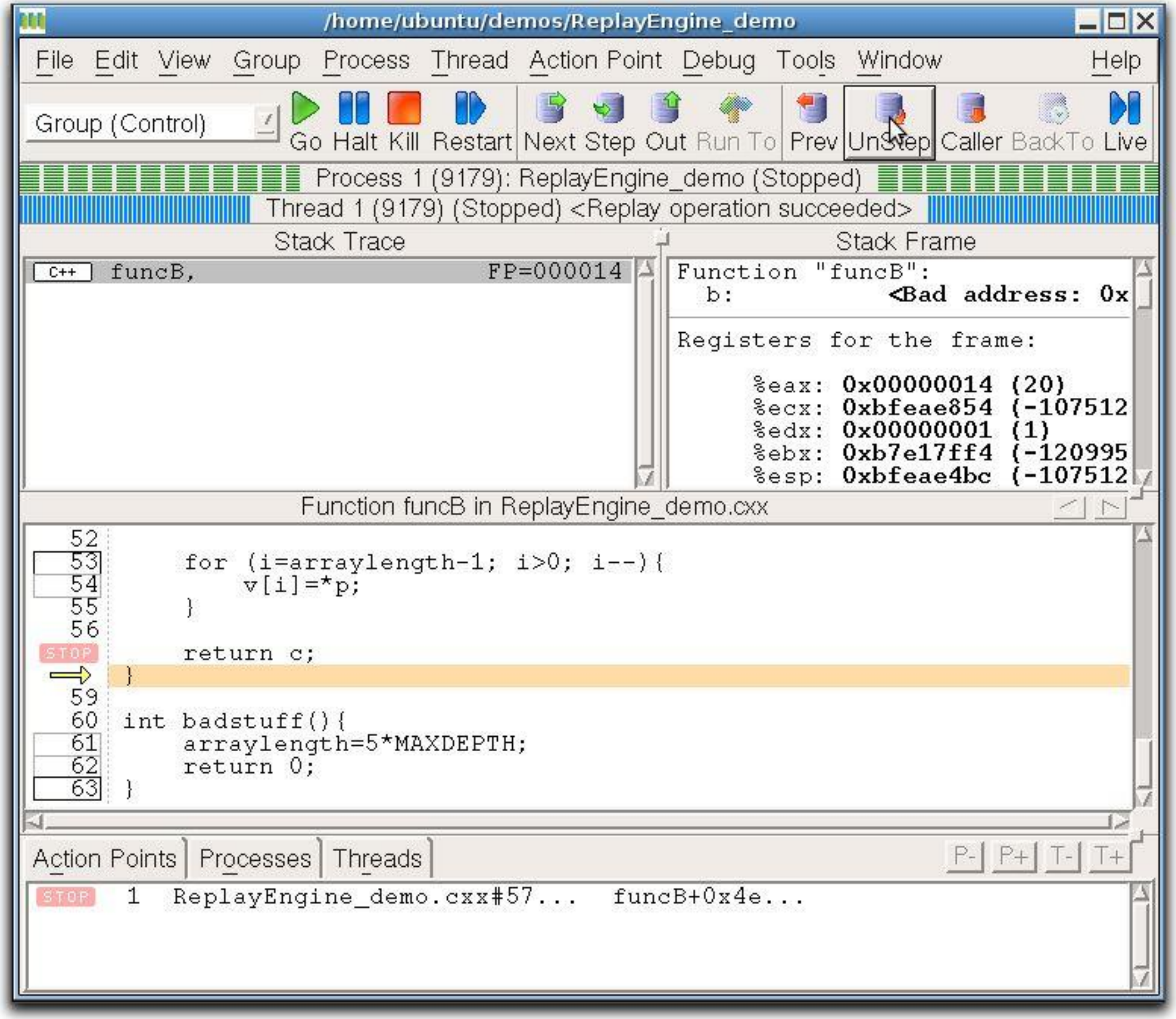
```
Process 1 (9179): ReplayEngine_demo (Error)
Thread 1 (9179) (Error) <Segmentation violation>
```

```
PC: 000014,               FP=bfeae4c0
```

```
Registers for the frame:

      %eax: 0xbfeae558 (-107512
      %ecx: 0xbfeae854 (-107512
      %edx: 0x00000001 (1)
      %ebx: 0xb7e17ff4 (-120995
      %esp: 0xbfeae4f0 (-107512
      %ebp: 0xbfeae568 (-107512
      %esi: 0xb7ff5ce0 (-120800
```

```
      0x0000000e:          ???
      0x0000000f:          ???
      0x00000010:          ???
      0x00000011:          ???
      0x00000012:          ???
      0x00000013:          ???
⇒ ▶  0x00000014:          ???
      0x00000015:          ???
      0x00000016:          ???
      0x00000017:          ???
      0x00000018:          ???
      0x00000019:          ???
```

```
STOP  1  ReplayEngine_demo.cxx#57...  funcB+0x4e...
```



Process 1 (9179): ReplayEngine_demo (Stopped)

Thread 1 (9179) (Stopped) <Replay operation succeeded>

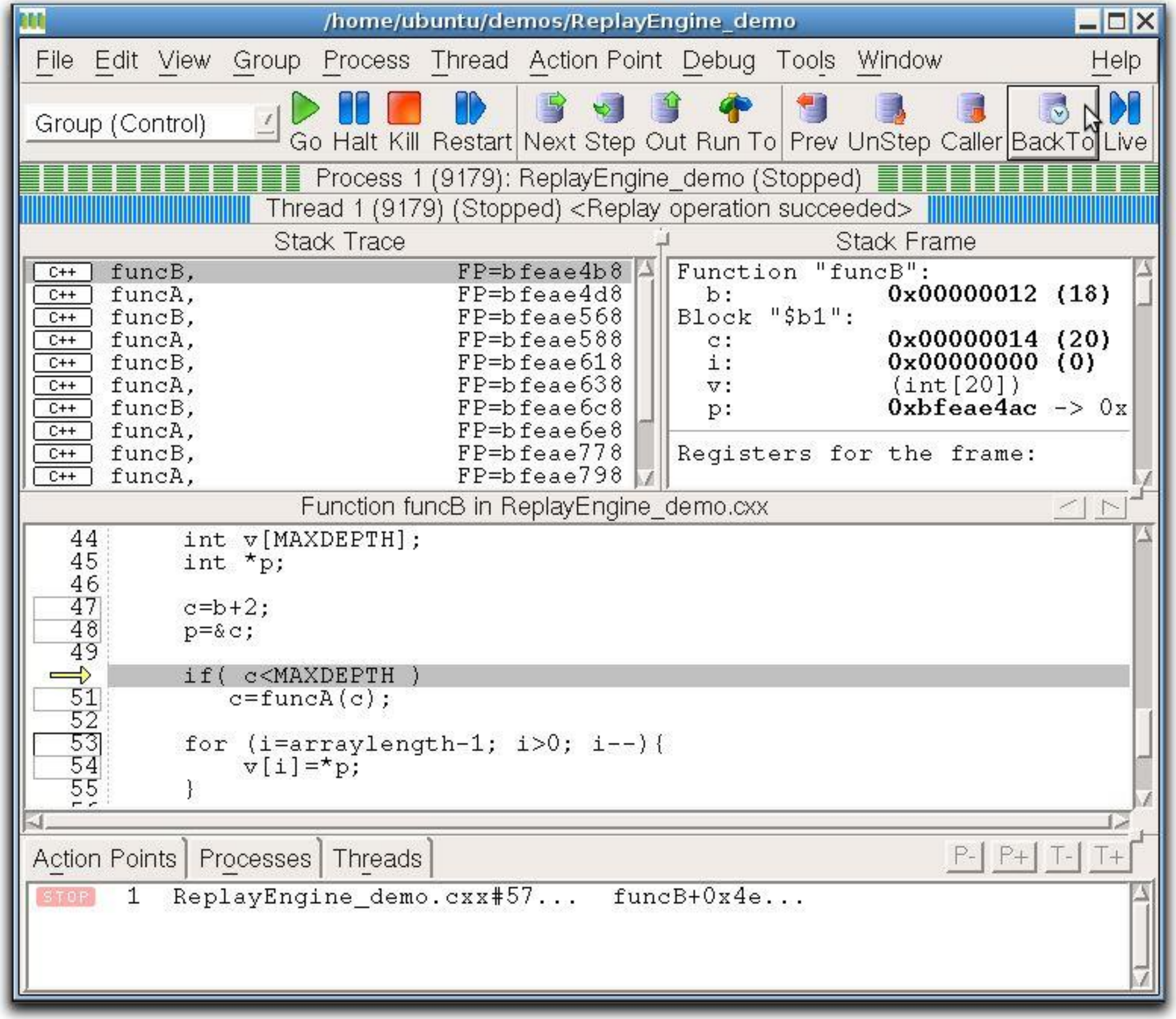
```
C++  funcB,                      FP=000014
```

```
Function "funcB":
  b:              <Bad address: 0x

Registers for the frame:

      %eax: 0x00000014 (20)
      %ecx: 0xbfeae854 (-107512
      %edx: 0x00000001 (1)
      %ebx: 0xb7e17ff4 (-120995
      %esp: 0xbfeae4bc (-107512
```

Function funcB in ReplayEngine_demo.cxx

```

    for (i=arraylength-1; i>0; i--){
        v[i]=*p;
    }

    return c;
}

int badstuff(){
    arraylength=5*MAXDEPTH;
    return 0;
}
```

```
STOP   1   ReplayEngine_demo.cxx#57...   funcB+0x4e...
```



/home/ubuntu/demos/ReplayEngine_demo
File Edit View Group Process Thread Action Point Debug Tools Window Help
Group (Control)
Go Halt Kill Restart Next Step Out Run To Prev UnStep Caller BackTo Live
Stack Trace
Stack Frame
Action Points Processes Threads
P- P+ T- T+

Process 1 (9179): ReplayEngine_demo (Stopped)

Thread 1 (9179) (Stopped) <Replay operation succeeded>

```
C++  funcB,    FP=bfeae4b8
C++  funcA,    FP=bfeae4d8
C++  funcB,    FP=bfeae568
C++  funcA,    FP=bfeae588
C++  funcB,    FP=bfeae618
C++  funcA,    FP=bfeae638
C++  funcB,    FP=bfeae6c8
C++  funcA,    FP=bfeae6e8
C++  funcB,    FP=bfeae778
C++  funcA,    FP=bfeae798
```

```
Function "funcB":
  b:          0x00000012 (18)
Block "$b1":
  c:          0x00000014 (20)
  i:          0x00000000 (0)
  v:          (int[20])
  p:          0xbfeae4ac -> 0x

Registers for the frame:
```

Function funcB in ReplayEngine_demo.cxx

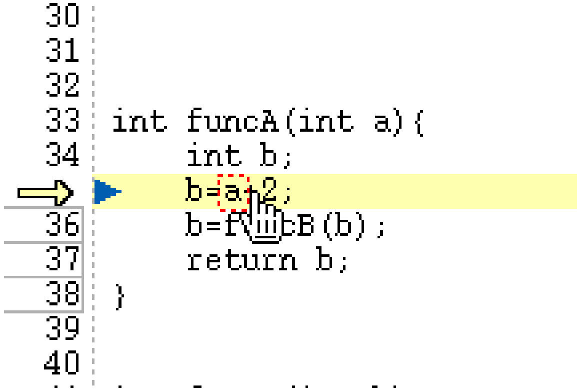
```
    int v[MAXDEPTH];
    int *p;

    c=b+2;
    p=&c;

    if( c<MAXDEPTH )
       c=funcA(c);

    for (i=arraylength-1; i>0; i--){
        v[i]=*p;
    }
```

```
STOP  1  ReplayEngine_demo.cxx#57...   funcB+0x4e...
```

LAB 8: REVERSE DEBUGGING WITH REPLAY ENGINE

New Capabilities in TotalView 8.10

- **CUDA 4.1**
- **Reverse Debugging**
  - Replay on Demand
  - C++View and ReplayEngine interop
- **Visual Dive Indicator**
- **Cray-specific enhancements**
  - Improved Cray Compiler Edition Support
  - ReplayEngine on Cray XE
  - CUDA support on Cray XK
  - Early Access Preview for OpenACC on the Cray with CCE 8 compiler
- **TVScript Scalability Improvements**
- **32 user bugs fixed**



```
int funcA(int a){
    int b;
    b=a+2;
    b=funcB(b);
    return b;
}
```

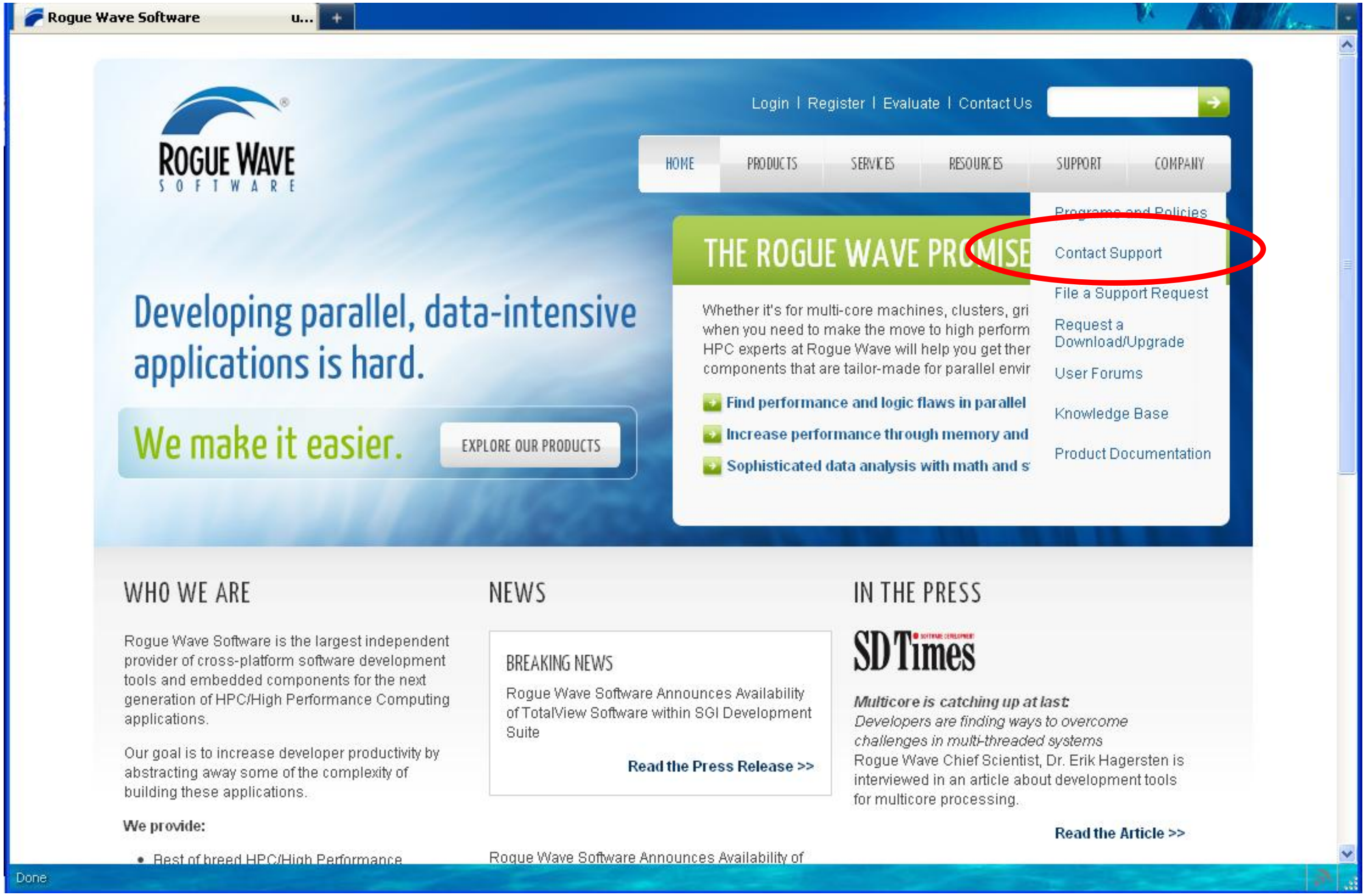
SUPPORT AND QUESTIONS

TotalView Customer Support

- **Email: tvsupport@roguewave.com**
- **Use our web site for documentation, demos, FAQs and to contact support**

TotalView Customer Support

http://www.roguewave.com



Developing parallel, data-intensive applications is hard.

We make it easier.

THE ROGUE WAVE PROMISE

Whether it's for multi-core machines, clusters, gri
when you need to make the move to high perform
HPC experts at Rogue Wave will help you get ther
components that are tailor-made for parallel envir

- Find performance and logic flaws in parallel
- Increase performance through memory and
- Sophisticated data analysis with math and s

WHO WE ARE

Rogue Wave Software is the largest independent provider of cross-platform software development tools and embedded components for the next generation of HPC/High Performance Computing applications.

Our goal is to increase developer productivity by abstracting away some of the complexity of building these applications.

We provide:

- Best of breed HPC/High Performance

NEWS

BREAKING NEWS

Rogue Wave Software Announces Availability of TotalView Software within SGI Development Suite

Read the Press Release >>

Rogue Wave Software Announces Availability of

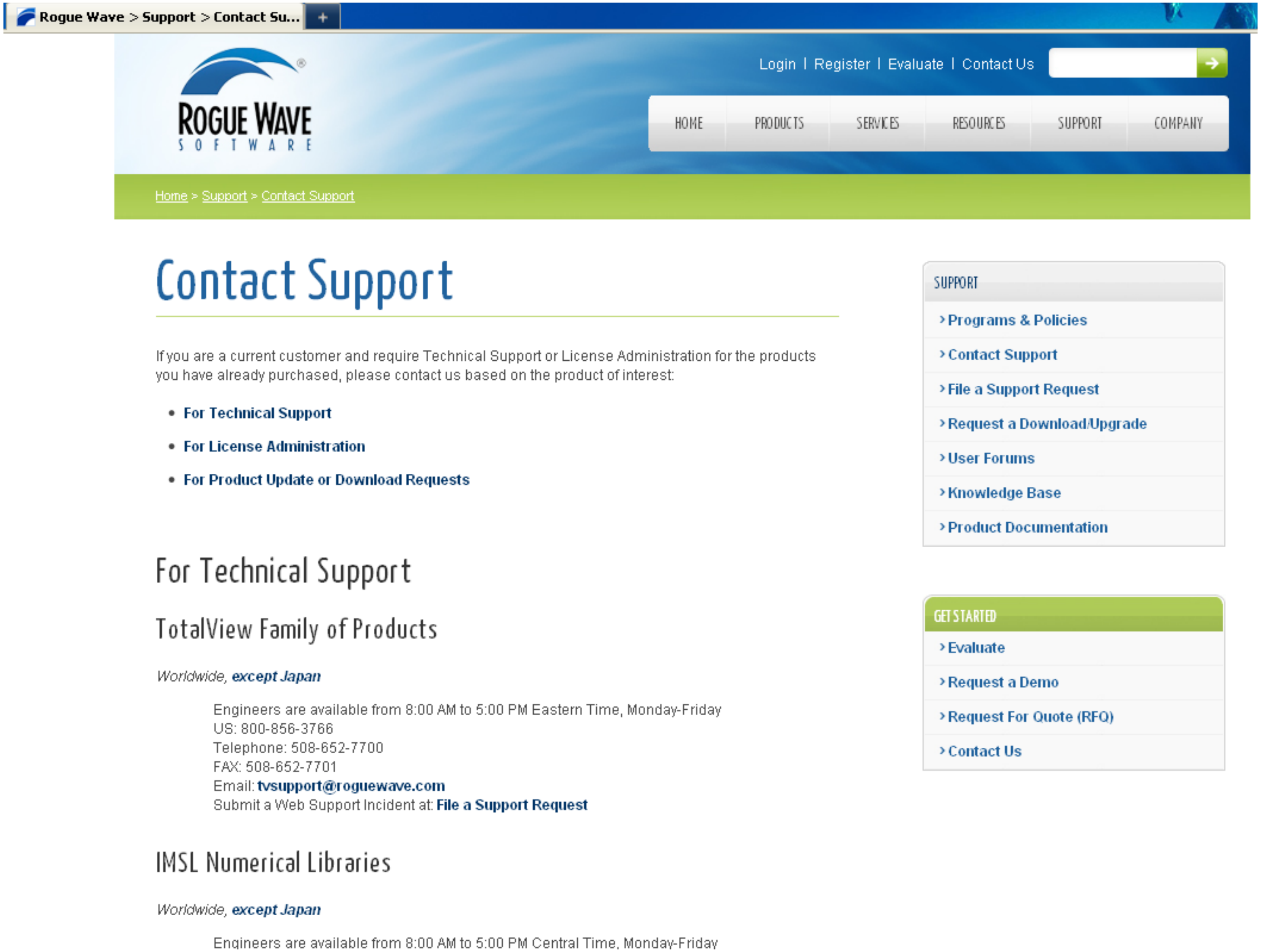
IN THE PRESS

SD Times

Multicore is catching up at last:
Developers are finding ways to overcome challenges in multi-threaded systems
Rogue Wave Chief Scientist, Dr. Erik Hagersten is interviewed in an article about development tools for multicore processing.

Read the Article >>

TotalView Customer Support



Rogue Wave > Support > Contact Su...
Login | Register | Evaluate | Contact Us
HOME
PRODUCTS
SERVICES
RESOURCES
SUPPORT
COMPANY
Home > Support > Contact Support

Contact Support

If you are a current customer and require Technical Support or License Administration for the products you have already purchased, please contact us based on the product of interest:

- For Technical Support
- For License Administration
- For Product Update or Download Requests

For Technical Support

TotalView Family of Products

*Worldwide, **except Japan***

Engineers are available from 8:00 AM to 5:00 PM Eastern Time, Monday-Friday
US: 800-856-3766
Telephone: 508-652-7700
FAX: 508-652-7701
Email: **tvsupport@roguewave.com**
Submit a Web Support Incident at: **File a Support Request**

IMSL Numerical Libraries

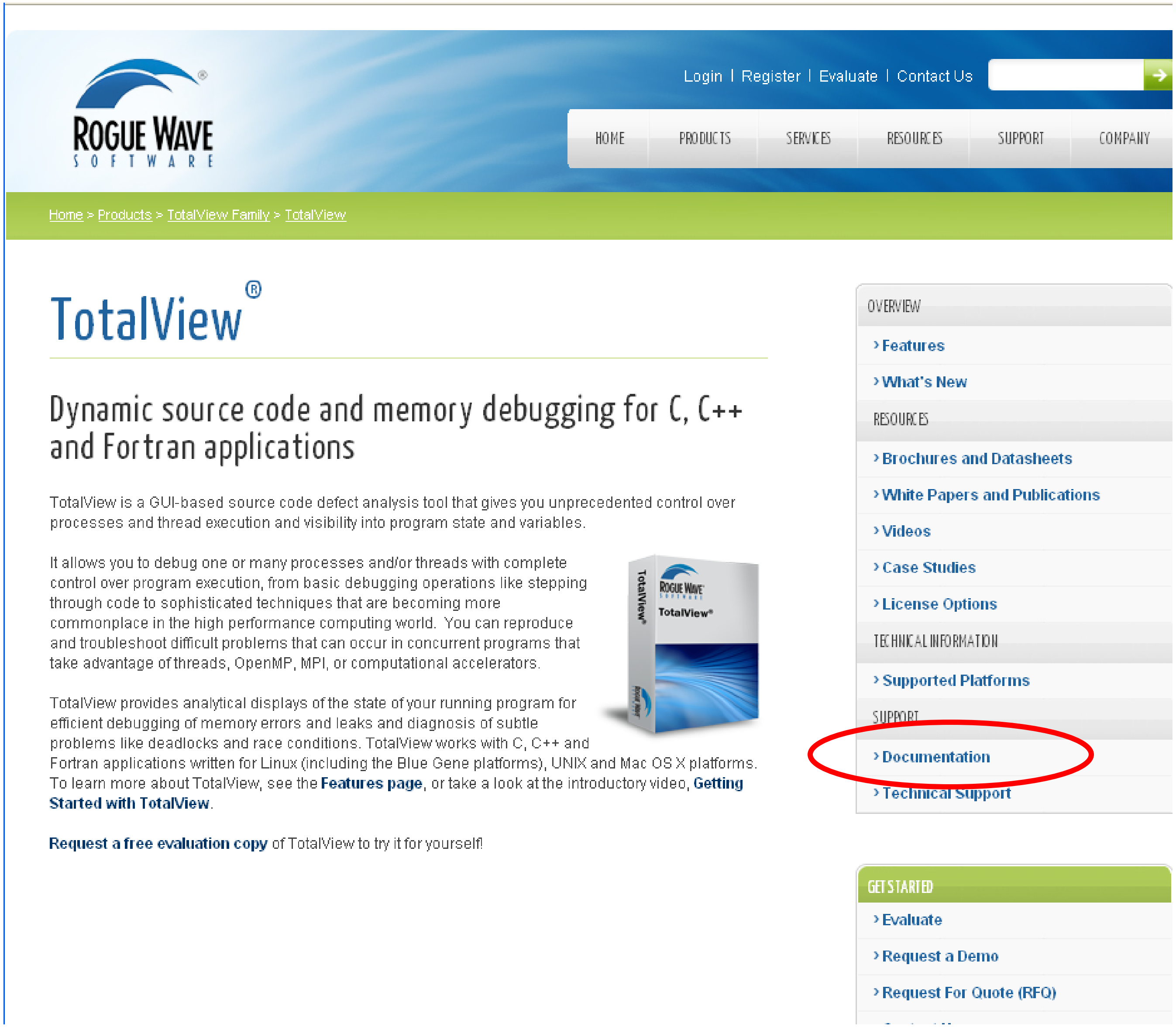
*Worldwide, **except Japan***

Engineers are available from 8:00 AM to 5:00 PM Central Time, Monday-Friday

SUPPORT
- Programs & Policies
- Contact Support
- File a Support Request
- Request a Download/Upgrade
- User Forums
- Knowledge Base
- Product Documentation

GET STARTED
- Evaluate
- Request a Demo
- Request For Quote (RFQ)
- Contact Us

TotalView Documentation



TotalView®

Dynamic source code and memory debugging for C, C++ and Fortran applications

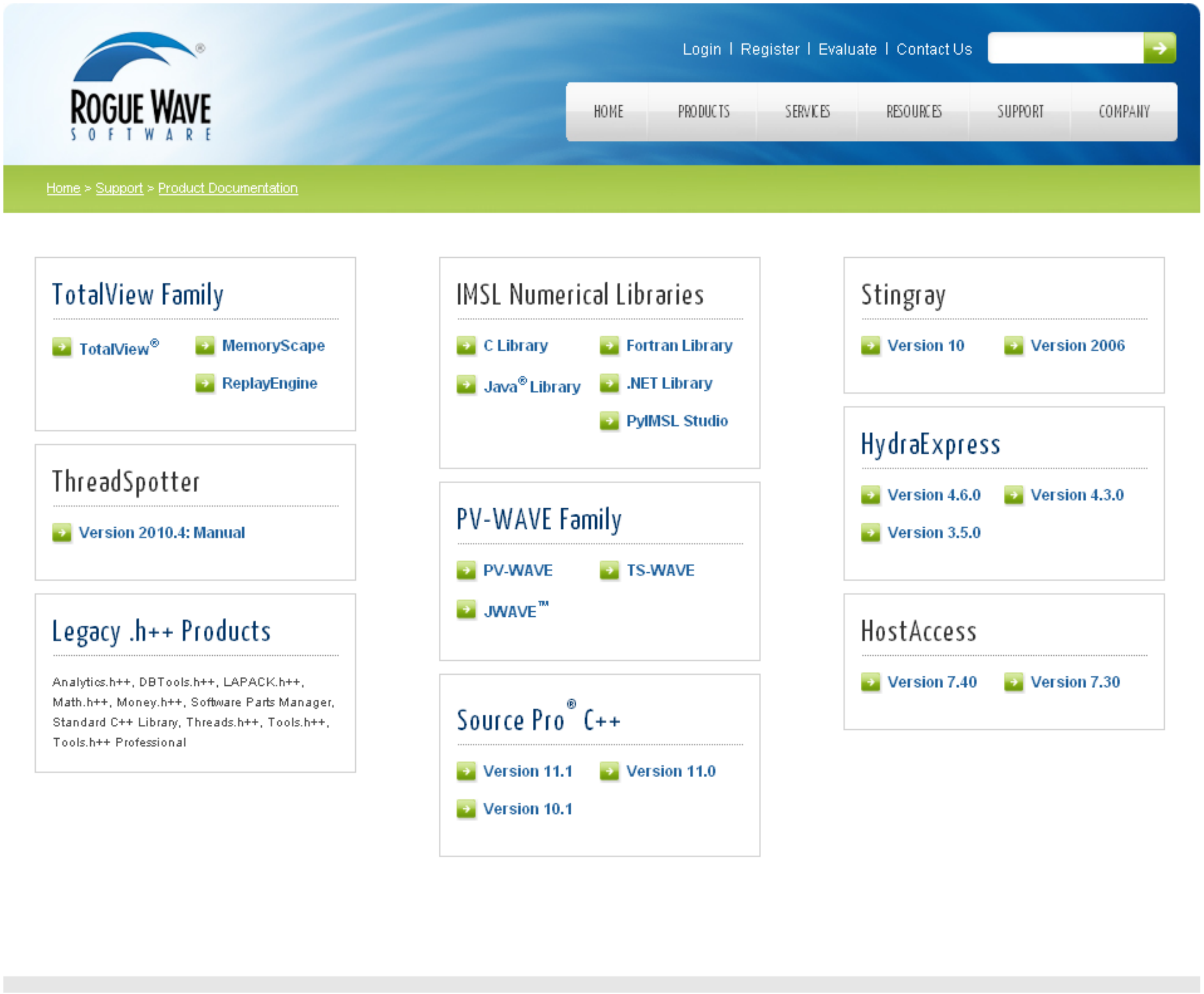
TotalView is a GUI-based source code defect analysis tool that gives you unprecedented control over processes and thread execution and visibility into program state and variables.

It allows you to debug one or many processes and/or threads with complete control over program execution, from basic debugging operations like stepping through code to sophisticated techniques that are becoming more commonplace in the high performance computing world. You can reproduce and troubleshoot difficult problems that can occur in concurrent programs that take advantage of threads, OpenMP, MPI, or computational accelerators.

TotalView provides analytical displays of the state of your running program for efficient debugging of memory errors and leaks and diagnosis of subtle problems like deadlocks and race conditions. TotalView works with C, C++ and Fortran applications written for Linux (including the Blue Gene platforms), UNIX and Mac OS X platforms. To learn more about TotalView, see the **Features page**, or take a look at the introductory video, **Getting Started with TotalView**.

Request a free evaluation copy of TotalView to try it for yourself!

TotalView Documentation


Login | Register | Evaluate | Contact Us
HOME
PRODUCTS
SERVICES
RESOURCES
SUPPORT
COMPANY
Home > Support > Product Documentation

TotalView Family

- TotalView®
- MemoryScape
- ReplayEngine

ThreadSpotter

- Version 2010.4: Manual

Legacy .h++ Products

Analytics.h++, DBTools.h++, LAPACK.h++, Math.h++, Money.h++, Software Parts Manager, Standard C++ Library, Threads.h++, Tools.h++, Tools.h++ Professional

IMSL Numerical Libraries

- C Library
- Fortran Library
- Java® Library
- .NET Library
- PyIMSL Studio

PV-WAVE Family

- PV-WAVE
- TS-WAVE
- JWAVE™

Source Pro® C++

- Version 11.1
- Version 11.0
- Version 10.1

Stingray

- Version 10
- Version 2006

HydraExpress

- Version 4.6.0
- Version 4.3.0
- Version 3.5.0

HostAccess

- Version 7.40
- Version 7.30

Thanks!

QUESTIONS?