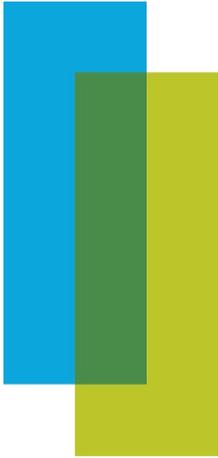


TotalView Product Review:

Memory Debugging

Remote Display Debugging



TotalView

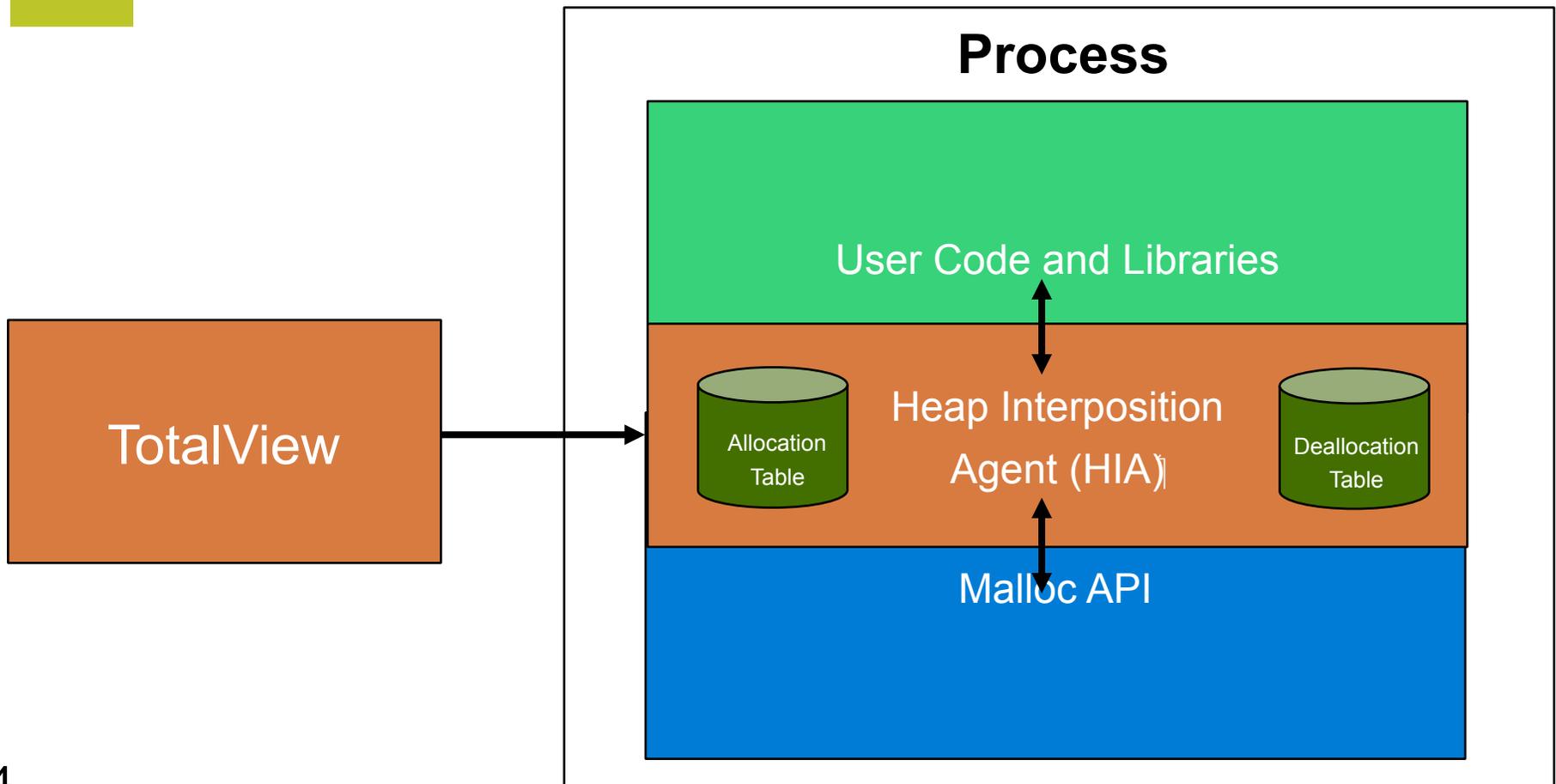
Memory Debugging

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
 - Over running array bounds

The Agent and Interposition

TotalView's HIA resides *between* your software and the OS



TotalView HIA Technology

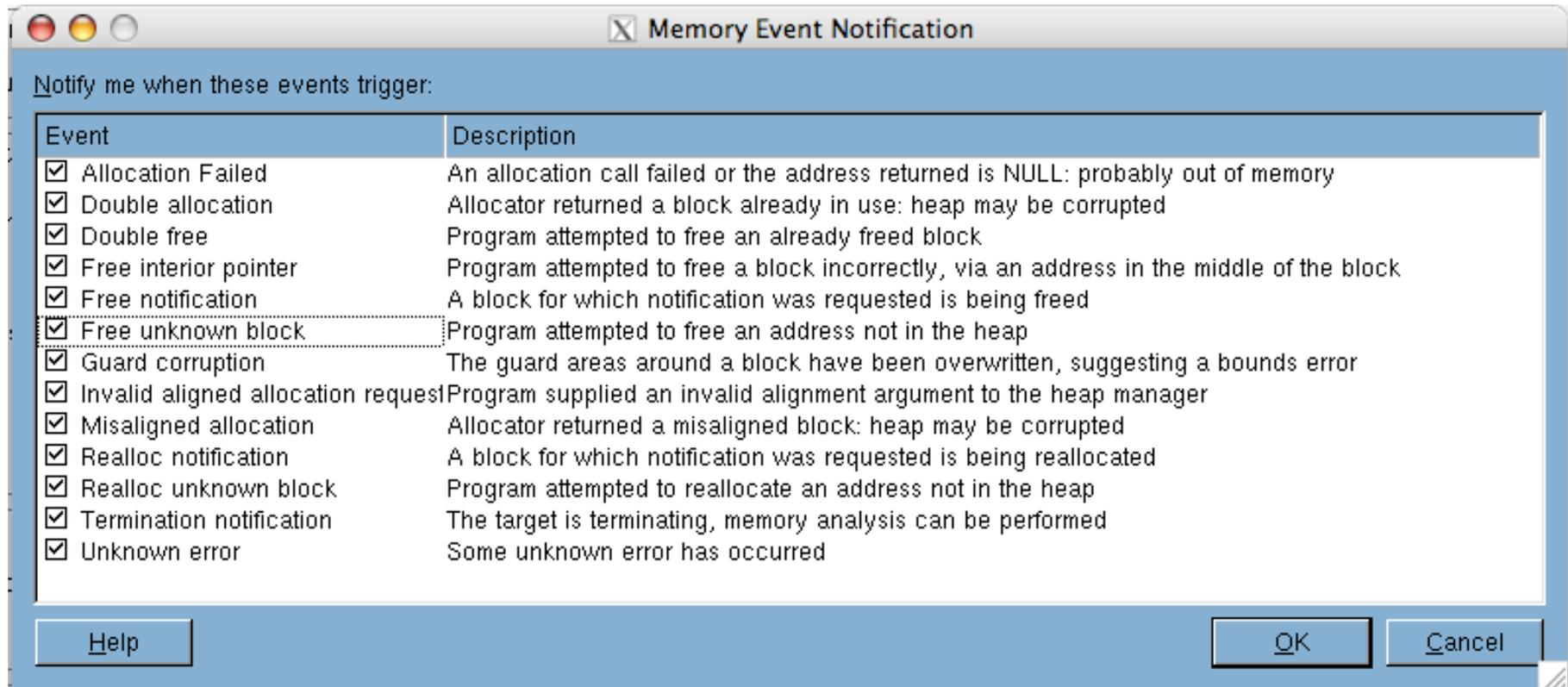
- **Advantages of TotalView HIA Technology**
 - Use it with your existing builds
 - No Source Code or Binary Instrumentation
 - Programs run nearly full speed
 - Low performance overhead
 - Low memory overhead
 - Efficient memory usage
 - Support wide range of platforms and compilers



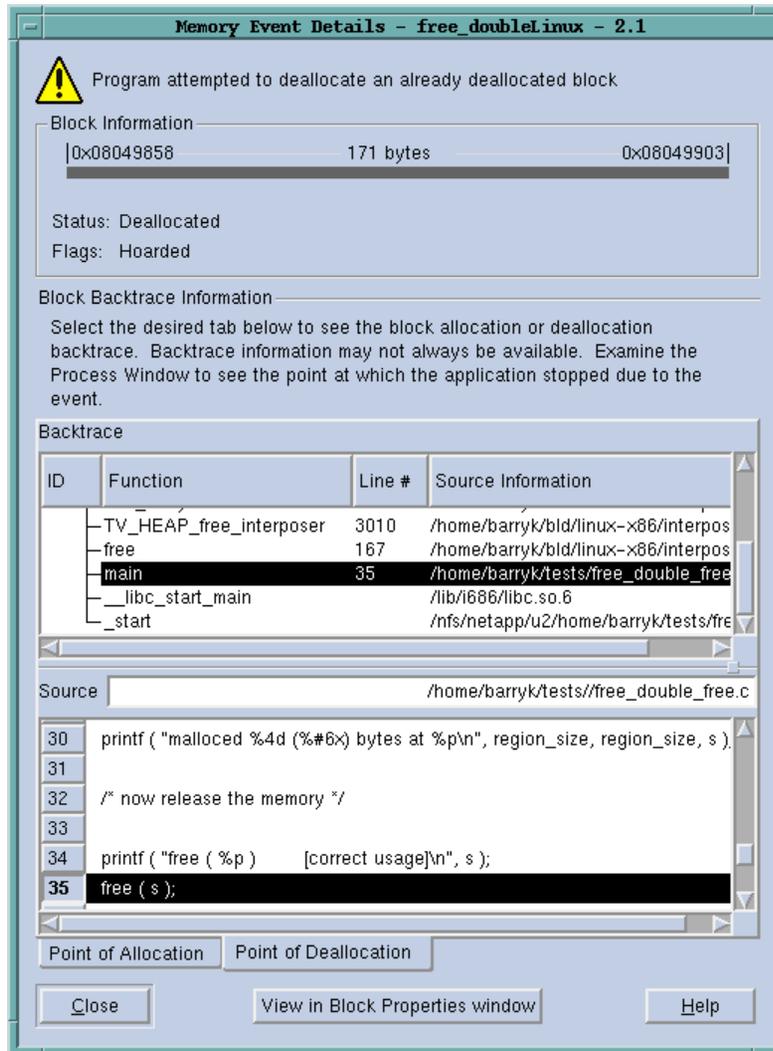
Memory Debugger Features

- **Automatically detect allocation problems**
- **View the heap**
- **Leak detection**
- **Block painting**
- **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 Details Window



Memory Event Details - free_doubleLinux - 2.1

 Program attempted to deallocate an already deallocated block

Block Information
|0x08049858 | 171 bytes | 0x08049903|

Status: Deallocated
Flags: Hoarded

Block Backtrace Information
Select the desired tab below to see the block allocation or deallocation backtrace. Backtrace information may not always be available. Examine the Process Window to see the point at which the application stopped due to the event.

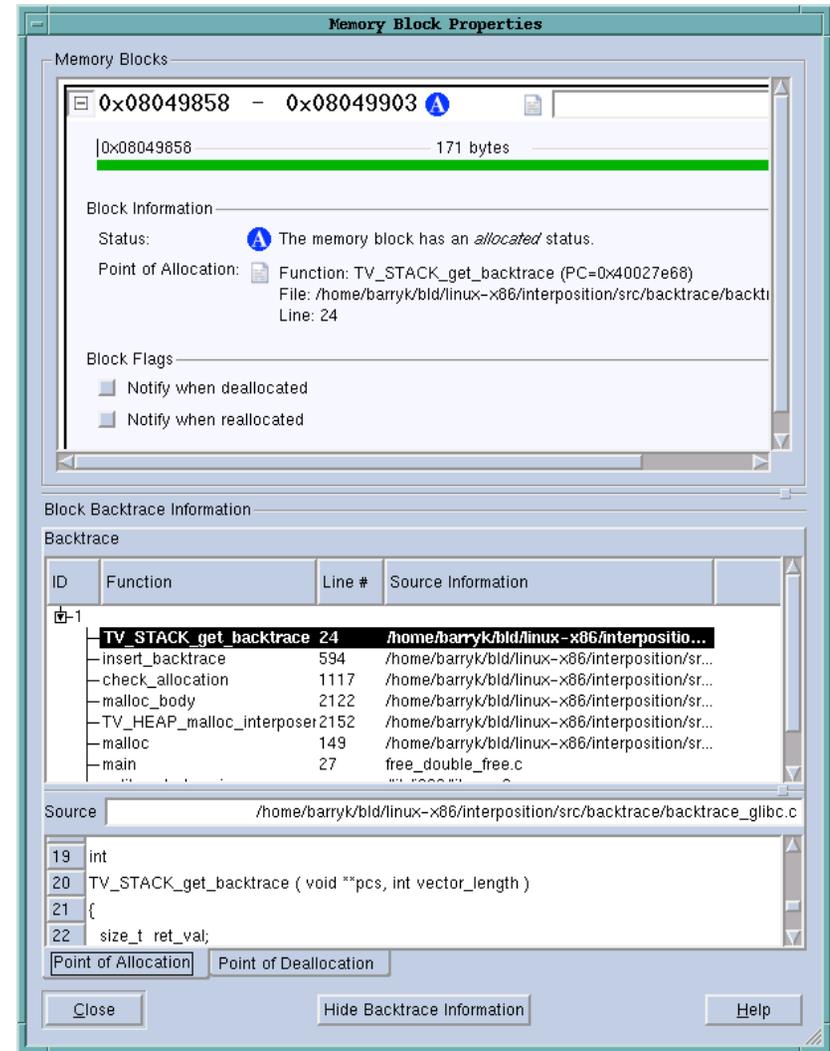
Backtrace

ID	Function	Line #	Source Information
	TV_HEAP_free_interposer	3010	/home/barryk/bld/linux-x86/interpos
	free	167	/home/barryk/bld/linux-x86/interpos
	main	35	/home/barryk/tests/free_double_free
	__libc_start_main		/lib/1686/libc.so.6
	_start		/nfs/netapp/u2/home/barryk/tests/fre

Source /home/barryk/tests/free_double_free.c

```
30 printf ("malloced %4d (%#6x) bytes at %p\n", region_size, region_size, s );
31
32 /* now release the memory */
33
34 printf ("free ( %p ) [correct usage]\n", s );
35 free ( s );
```

Point of Allocation | Point of Deallocation



Memory Block Properties

Memory Blocks
|0x08049858 - 0x08049903 |

Block Information
Status:  The memory block has an *allocated* status.
Point of Allocation:  Function: TV_STACK_get_backtrace (PC=0x40027e68)
File: /home/barryk/bld/linux-x86/interposition/src/backtrace/backtr
Line: 24

Block Flags
 Notify when deallocated
 Notify when reallocated

Block Backtrace Information

Backtrace

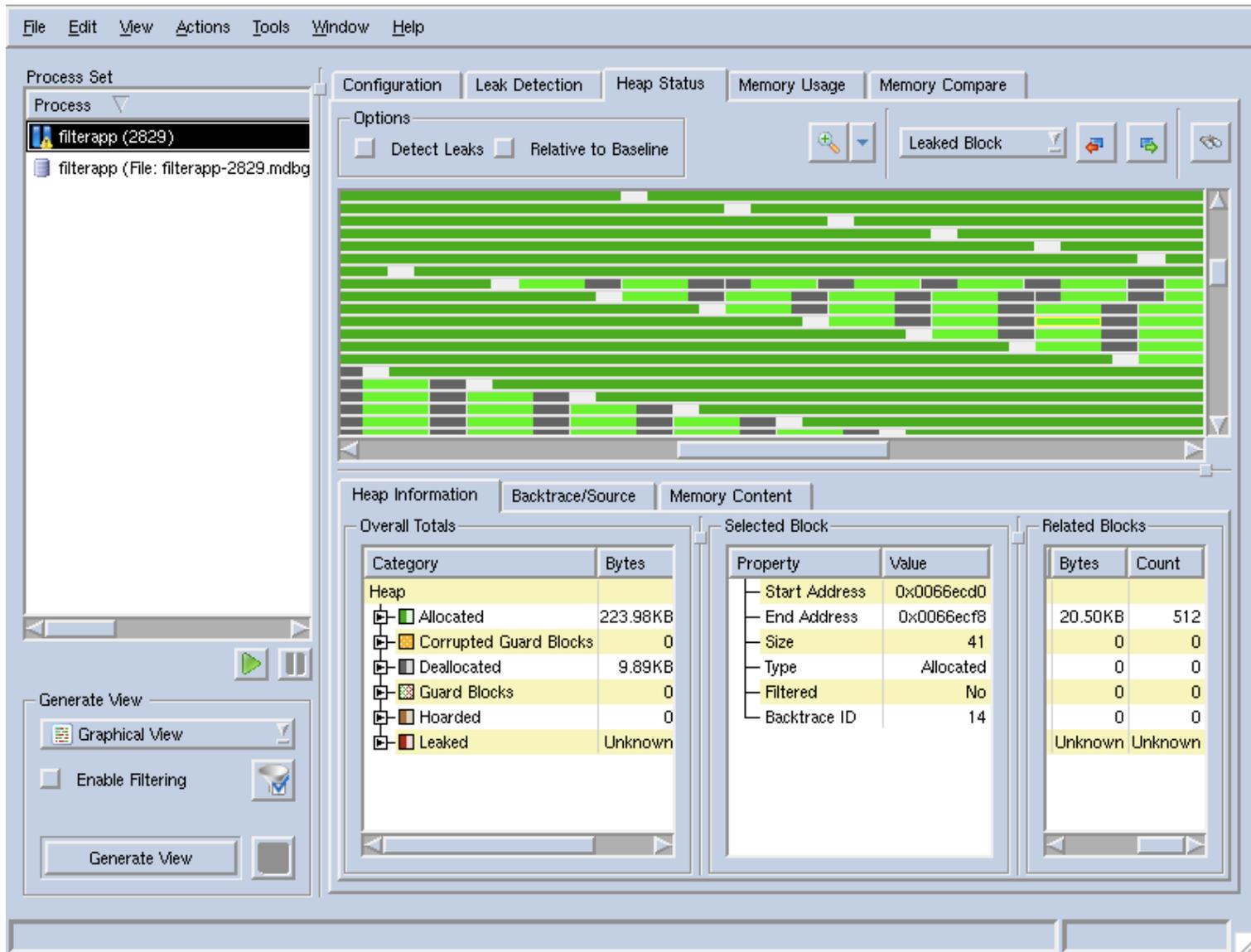
ID	Function	Line #	Source Information
-1	TV_STACK_get_backtrace	24	/home/barryk/bld/linux-x86/interpositio...
	insert_backtrace	534	/home/barryk/bld/linux-x86/interposition/sr...
	check_allocation	1117	/home/barryk/bld/linux-x86/interposition/sr...
	malloc_body	2122	/home/barryk/bld/linux-x86/interposition/sr...
	TV_HEAP_malloc_interposer	2152	/home/barryk/bld/linux-x86/interposition/sr...
	malloc	149	/home/barryk/bld/linux-x86/interposition/sr...
	main	27	free_double_free.c

Source /home/barryk/bld/linux-x86/interposition/src/backtrace/backtrace_glibc.c

```
19 int
20 TV_STACK_get_backtrace ( void **pcs, int vector_length )
21 {
22 size_t ret_val;
```

Point of Allocation | Point of Deallocation

Heap Graphical View



The screenshot displays the TOTALVIEW software interface in the 'Heap Graphical View' for the process 'filterapp (2829)'. The interface is divided into several sections:

- Process Set:** Lists the active process 'filterapp (2829)' and its file path 'filterapp (File: filterapp-2829.mdbg)'.
- Configuration:** Includes tabs for 'Configuration', 'Leak Detection', 'Heap Status', 'Memory Usage', and 'Memory Compare'. Under 'Options', there are checkboxes for 'Detect Leaks' and 'Relative to Baseline', and a 'Leaked Block' dropdown menu.
- Graphical View:** A large area showing a horizontal bar chart of memory blocks. Green bars represent allocated memory, and grey bars represent deallocated or other states.
- Heap Information:** A table showing overall totals for different heap categories.
- Selected Block:** A table showing properties for a specific memory block.
- Related Blocks:** A table showing related memory blocks.

Overall Totals Table:

Category	Bytes
Heap	
Allocated	223.98KB
Corrupted Guard Blocks	0
Deallocated	9.89KB
Guard Blocks	0
Hoarded	0
Leaked	Unknown

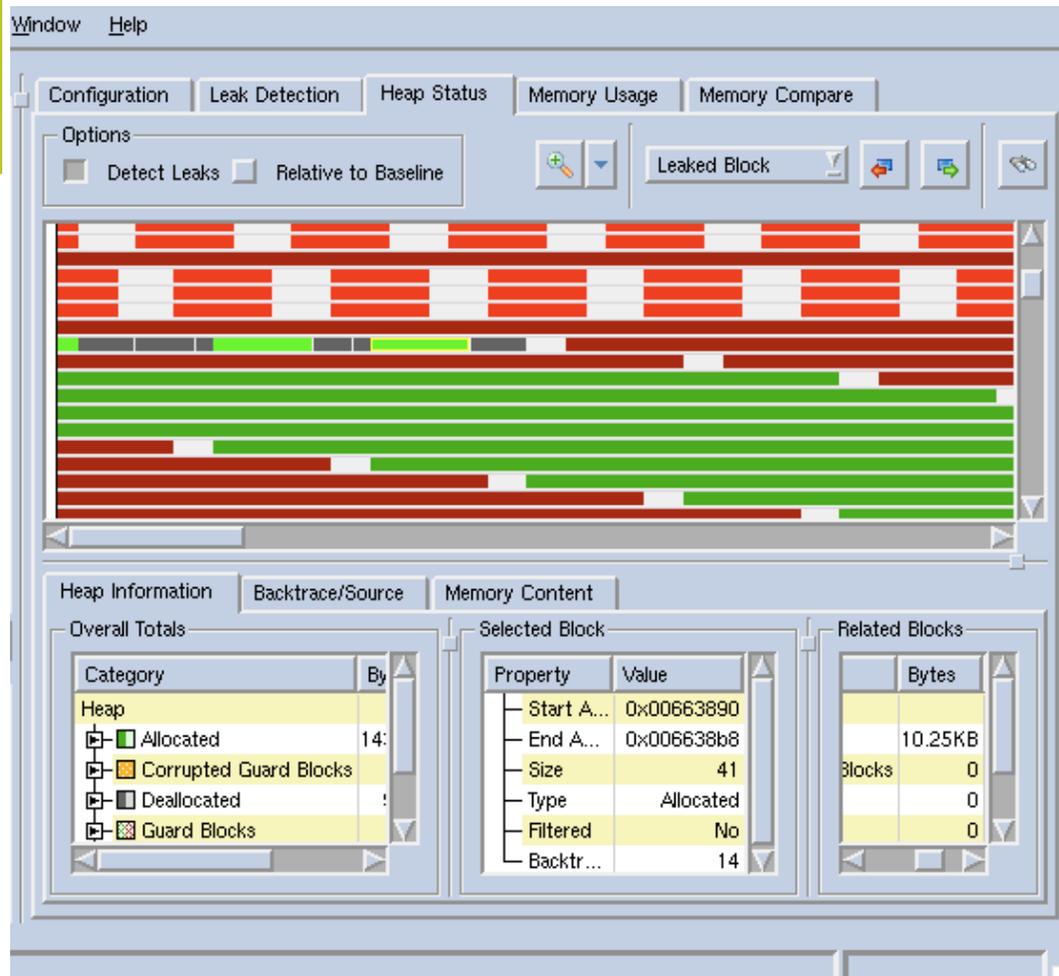
Selected Block Table:

Property	Value
Start Address	0x0066ecd0
End Address	0x0066ecf8
Size	41
Type	Allocated
Filtered	No
Backtrace ID	14

Related Blocks Table:

Bytes	Count
20.50KB	512
0	0
0	0
0	0
0	0
Unknown	Unknown

Leak Detection



The screenshot displays the TotalView Leak Detection tool interface. At the top, there are tabs for 'Configuration', 'Leak Detection', 'Heap Status', 'Memory Usage', and 'Memory Compare'. The 'Leak Detection' tab is active, showing options for 'Detect Leaks' (checked) and 'Relative to Baseline' (unchecked). A search bar contains 'Leaked Block'. The main area shows a memory dump with red and green bars representing memory blocks. Below the dump are three panels: 'Overall Totals', 'Selected Block', and 'Related Blocks'.

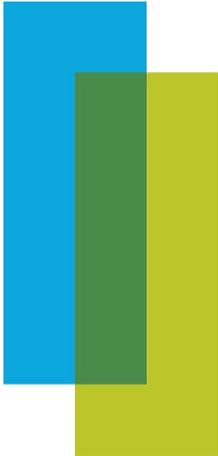
Category	By
Heap	
Allocated	14:
Corrupted Guard Blocks	
Deallocated	:
Guard Blocks	

Property	Value
Start A...	0x00663890
End A...	0x006638b8
Size	41
Type	Allocated
Filtered	No
Backtr...	14

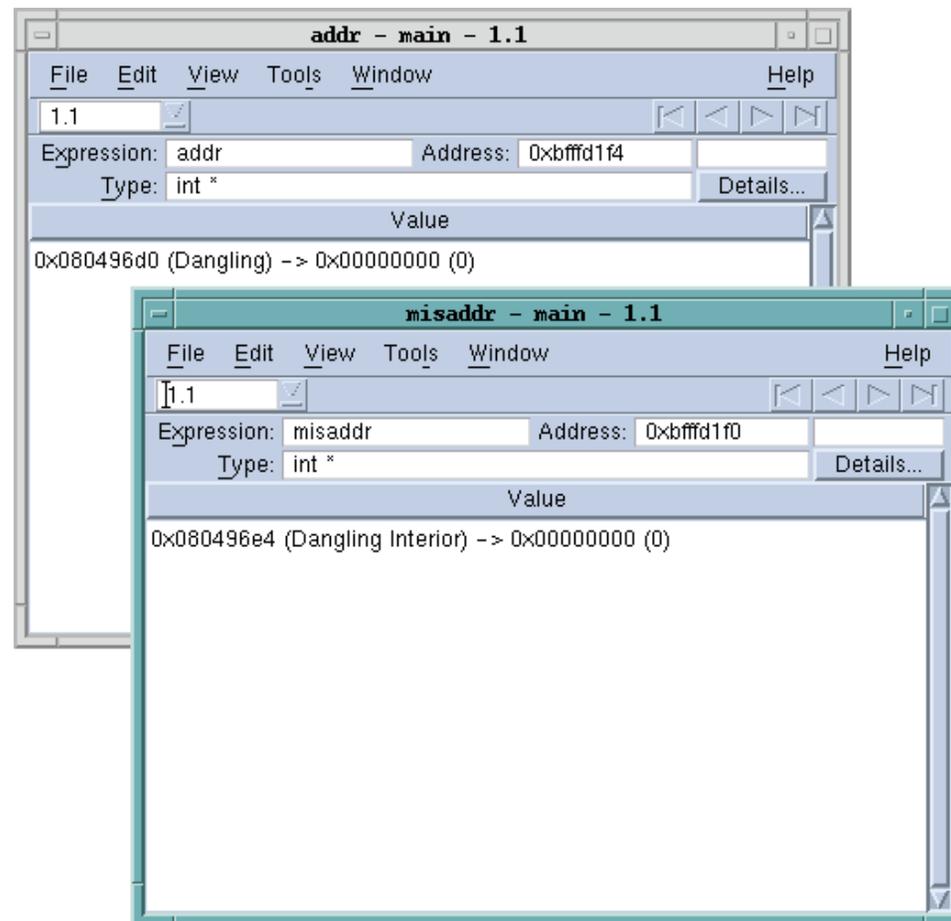
	Bytes
	10.25KB
Blocks	0
	0

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



The image displays two debugger windows from TotalView Technologies, illustrating dangling pointer detection. Both windows are titled "main - 1.1".

The top window, titled "addr - main - 1.1", shows the variable "addr" at memory address 0xbfffd1f4. Its type is "int *". The value is displayed as "0x080496d0 (Dangling) -> 0x00000000 (0)", indicating a pointer to a null value.

The bottom window, titled "misaddr - main - 1.1", shows the variable "misaddr" at memory address 0xbfffd1f0. Its type is "int *". The value is displayed as "0x080496e4 (Dangling Interior) -> 0x00000000 (0)", indicating a pointer to a null value.

Memory Corruption Detection (Guard Blocks)

Guard Blocks

Guard Blocks

Pre-Guard Size: 8 bytes

Pattern: 0x77777777

Post-Guard Size: 8 bytes

Pattern: 0x99999999

Maximum Guard Size: 0 bytes

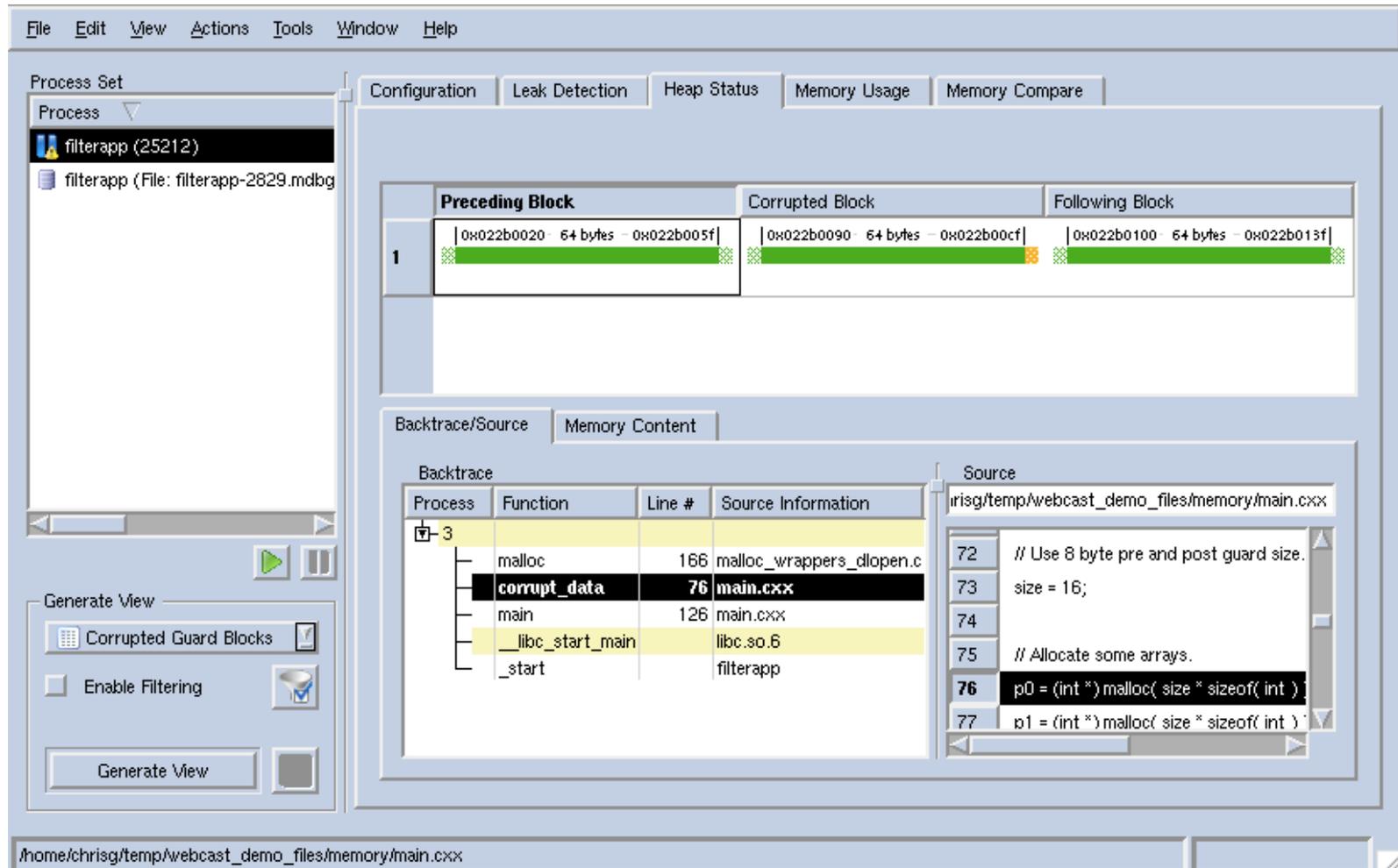
Configuration Leak Detection Heap Status Memory Usage Memory Compare

Preceding Block	Corrupted Block	Following Block
1 0x080+a028 - 171 bytes - 0x080+a0d2	0x080+a0e8 - 32 bytes - 0x080+a107	

Backtrace

Process	Function	Line #	Source Information	Source
1	malloc	149	malloc_wrappe	/home/barryk/fred/hia_events.c
	main	106	hia_events.c	101 102 break; 103 104 case notify_guard_corrupti 105 106 s3 = (char *)malloc (0xab 107
	__libc_start_main		libc.so.6	
	_start		hia_events	

Memory Corruption Report



The screenshot shows the TotalView Memory Corruption Report interface. The main window displays a memory corruption report for the process 'filterapp (25212)'. The report is organized into several sections:

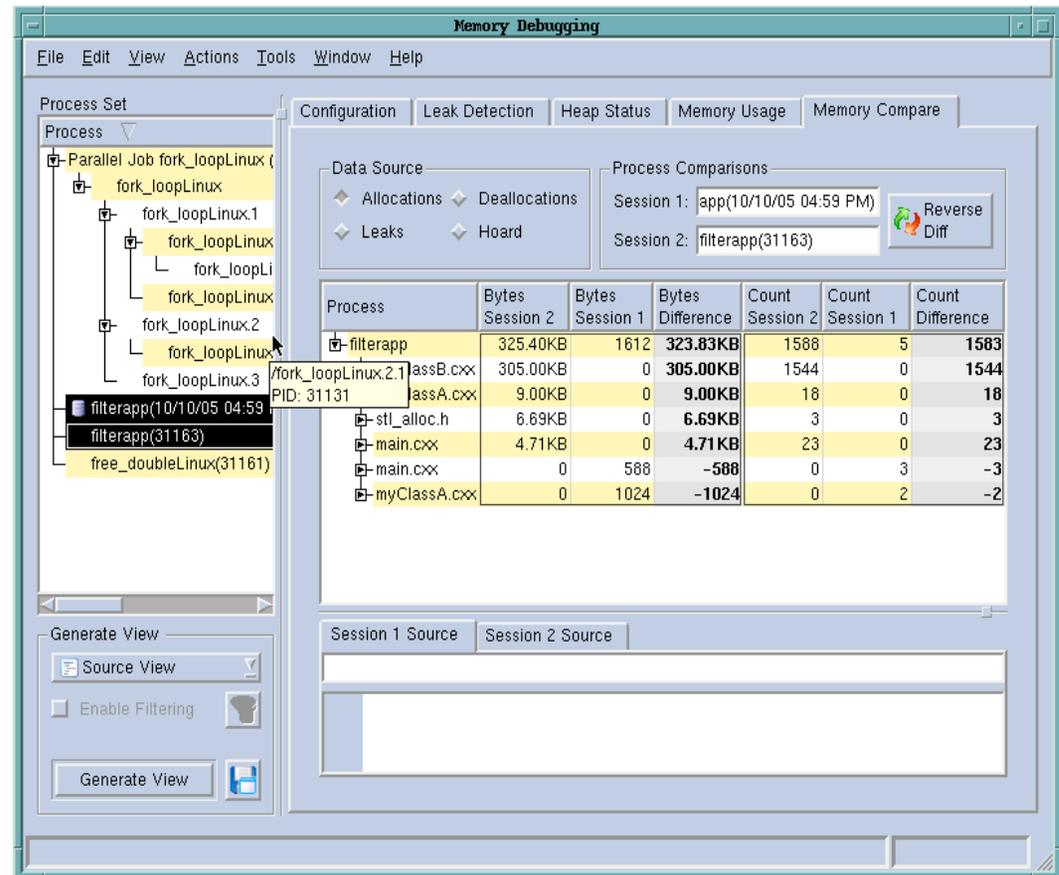
- Process Set:** Shows the process 'filterapp (25212)' and the file 'filterapp (File: filterapp-2629.mdbg)'.
- Configuration:** Includes tabs for 'Leak Detection', 'Heap Status', 'Memory Usage', and 'Memory Compare'.
- Memory Corruption Details:** A table showing the 'Preceding Block', 'Corrupted Block', and 'Following Block'. The corrupted block is highlighted in orange.
- Backtrace/Source:** A table showing the backtrace of the corruption, including the process, function, line number, and source information. The source code is displayed in a separate window.

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

```
irisg/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 ) );
```

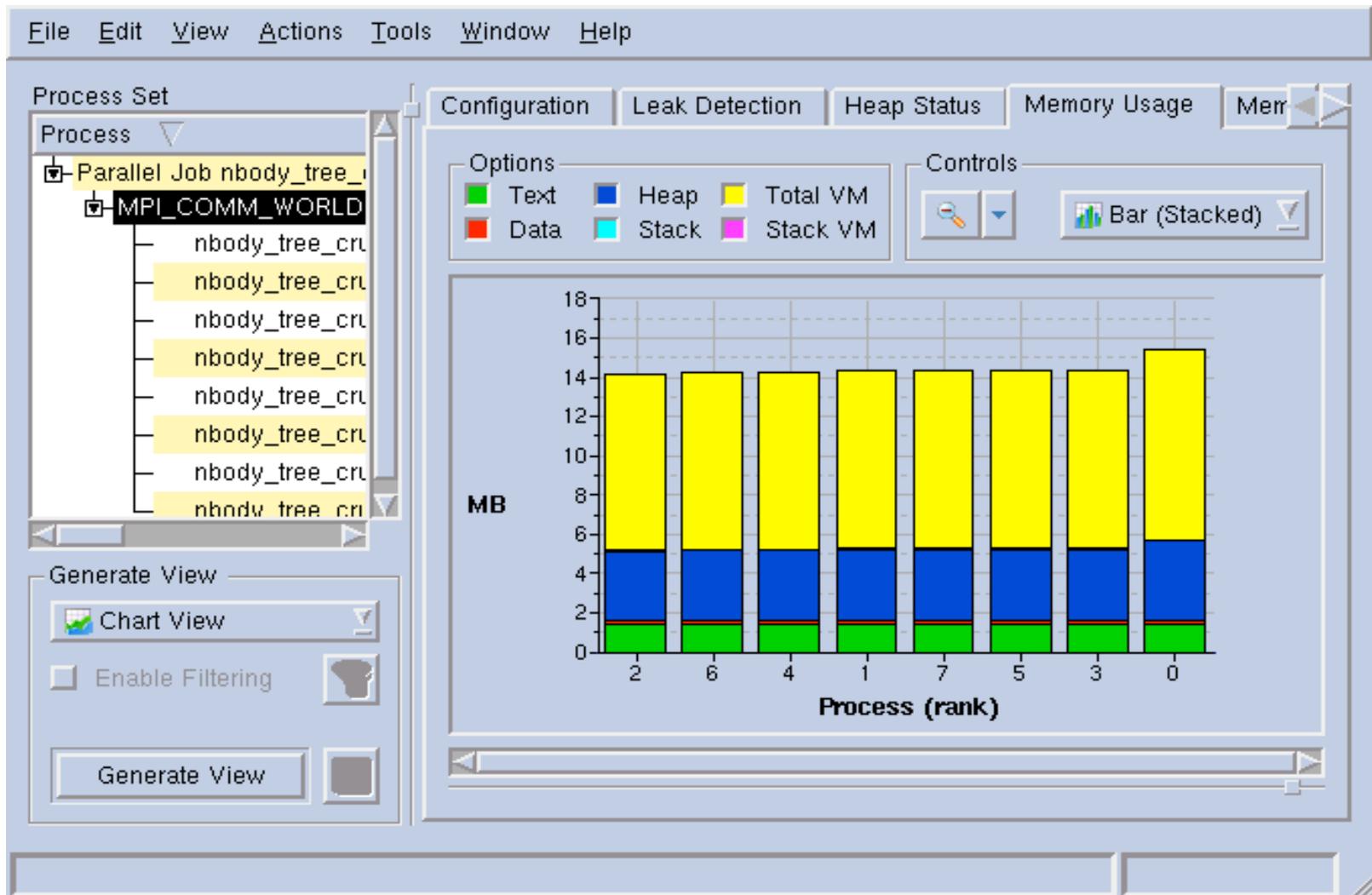
Memory Comparisons

- **“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



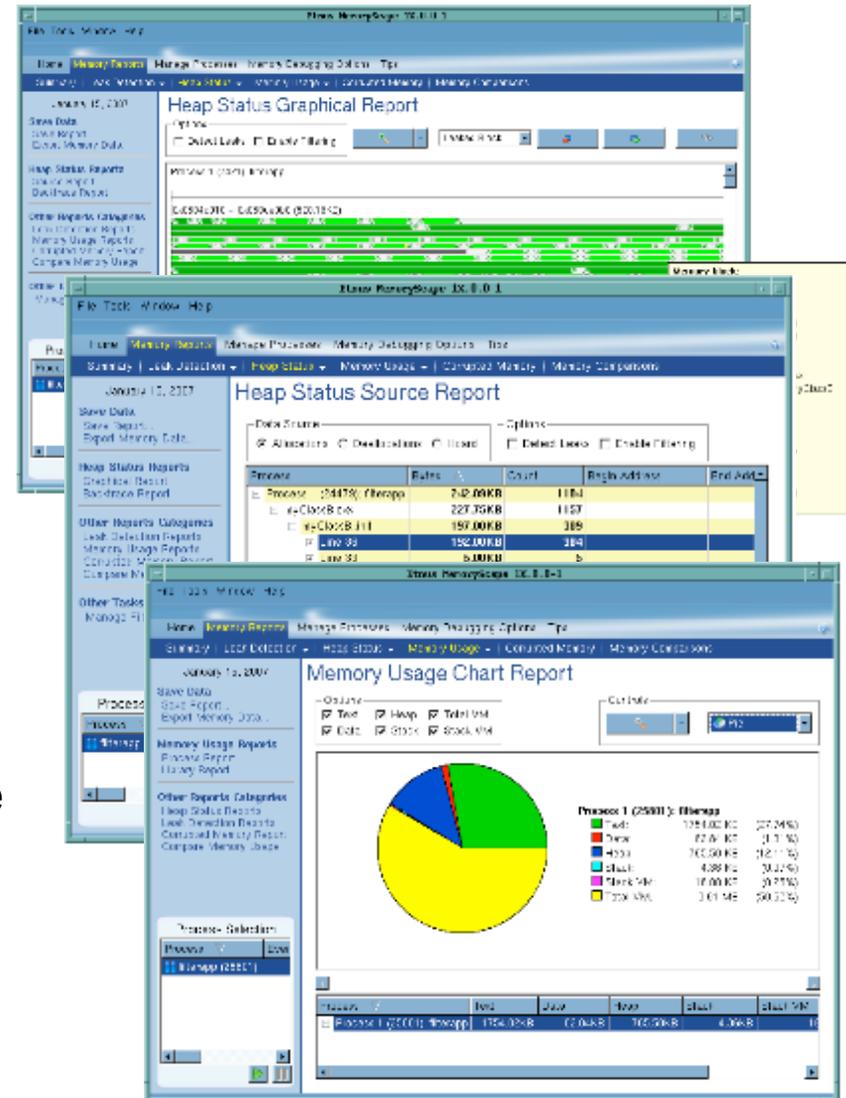
Process	Bytes Session 2	Bytes Session 1	Bytes Difference	Count Session 2	Count Session 1	Count Difference
-filterapp	325.40KB	1612	323.83KB	1588	5	1583
fork_loopLinux.2.1\assB.cxx	305.00KB	0	305.00KB	1544	0	1544
fork_loopLinux.2.1\assA.cxx	9.00KB	0	9.00KB	18	0	18
-stl_alloc.h	6.69KB	0	6.69KB	3	0	3
-main.cxx	4.71KB	0	4.71KB	23	0	23
-main.cxx	0	588	-588	0	3	-3
-myClassA.cxx	0	1024	-1024	0	2	-2

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



The image displays three overlapping screenshots of the TotalView Memory Debugger interface, illustrating various memory analysis reports:

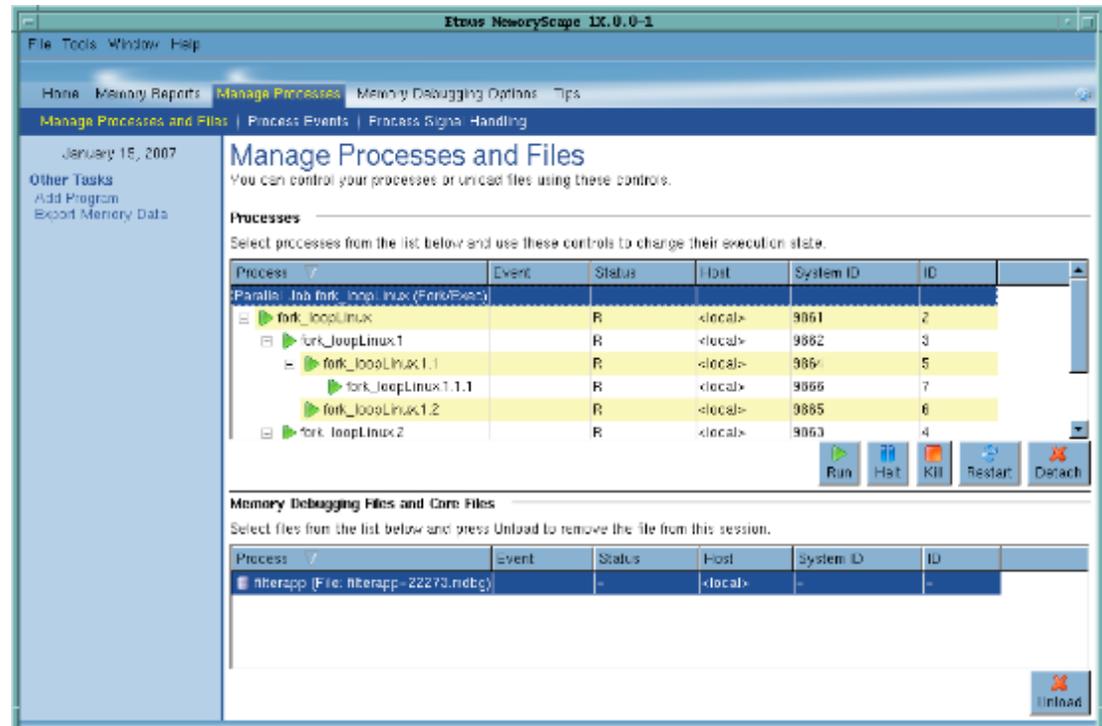
- Heap Status Graphical Report:** Shows a horizontal bar chart representing memory allocation and deallocation over time for a specific process.
- Heap Status Source Report:** Provides a detailed table of memory allocations, including process ID, file name, line number, size, and address.

Process	File	Line	Size	Count	Begin Address	End Address
Process 1 (25478): libexp	libexp	242	242.00KB	1164		
Process 1 (25478): libexp	libexp	227	227.75KB	1157		
Process 1 (25478): libexp	libexp	157	157.00KB	389		
Process 1 (25478): libexp	libexp	33	152.00KB	384		
Process 1 (25478): libexp	libexp	3	8.00KB	8		
- Memory Usage Chart Report:** Features a pie chart showing the distribution of memory usage across different categories for a selected process.

Category	Size	Percentage
Heap	102.00 KB	27.24%
Text	22.00 KB	5.71%
Stack	703.50 KB	18.11%
Free	4.88 KB	0.12%
Stack VM	17.00 KB	0.43%
Text VM	1.61 MB	40.89%

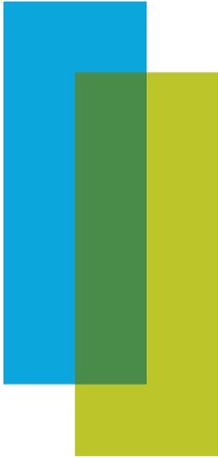
Multi-Process and Multi-Thread

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



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



Support

[Home](#) | [Support](#) | [TotalView Documentation](#)

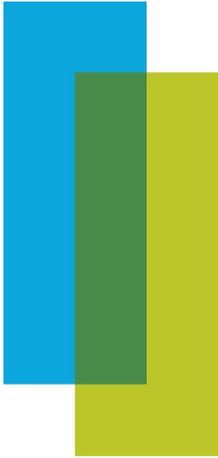
- [New Features](#)
- [Full User Guide](#)
- [Reference Guide](#)
- [GUI Reference Guide](#)
- [Platforms and System Requirements](#)
- [Installing TotalView](#)
- [Getting Started](#)
- [Using Threads Processes and Threads](#)
- [Using the Remote Display Client](#)
- [Setting Up a Debugging Session](#)
- [Setting Up MPI Debugging Sessions](#)
- [Setting Up Parallel Debugging Sessions](#)
- [Setting Up tvdsrv](#)
- [Command Line Syntax](#)
- [Using TotalView Windows](#)
- [Visualizing Programs and Data](#)
- [Debugging Programs](#)
- [Examining and Changing Data](#)
- [Examining Arrays](#)
- [Setting Action Points](#)
- [Evaluating Expressions](#)
- [Debugging Memory Problems](#)
- [Using the CLI](#)
- [Creating Type Transformations](#)

 **Download Free Trial**

NEXT STEPS

-  [Contact Sales](#)
-  [View License Options](#)
-  [Request Web Demo](#)
-  [Sign up for The View](#)





Developers & Support

[Home](#) | [Developers & Support](#) | [Video Tutorials](#)

Video Tutorials for TotalView, MemoryScope and ReplayEngine

These on demand video tutorials provide you with an easy and efficient way to learn the key capabilities of our products. Whether you are currently evaluating TotalView, ReplayEngine or MemoryScope, or are looking for a brief update on the latest versions, these short videos are an ideal self paced learning tool.

» **C/C++ Support**

6 MINUTES

» **Deterministic Replay with ReplayEngine**

5 MINUTES

» **Getting Started with TotalView**

14 MINUTES

» **Memory Debugging**

5 MINUTES

» **Asynchronous Threads**

2 MINUTES

» **Threads Navigation**

2.5 MINUTES

» **Setting Breakpoint Preferences for Threads**

2.5 MINUTES

» **Viewing Data Across Threads**

2.5 MINUTES

» **Debugging OpenMP with TotalView**

2.5 MINUTES

» **Seeing Threads and Process Status**

2 MINUTES

On Demand Webinars

» [TotalView Technologies ReplayEngine demo](#)

» [TotalView Technologies TotalView demo](#)

 **Download
Free Trial**

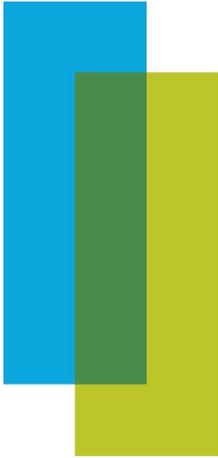
NEXT STEPS

 [Contact Sales](#)

 [View License Options](#)

 [Request Web Demo](#)

 [Sign up for The View](#)



TotalView

Remote Display Debugging

Easy, Secure and Fast

Remote Display Client

- **The Remote Display Client offers users the ability to easily set up and operate a TotalView debug session that is running on another system.**
- **TotalView Remote Display has two components:**
 - a Client, which runs on the remote system
 - a Server, which runs on any system supported by TotalView, “invisibly” manages the connections between the host and client
- **No license is needed to run the Client.**
- **The Remote Display Client is available for:**
 - Linux x86
 - Linux x86-64
 - Windows XP
 - Windows Vista
- **TotalView Remote Display presents a window on your machine that will display TotalView executing on a remote system.**

TotalView 8.6

Remote Display Client

For Windows, TotalView provides a Set up Wizard



TotalView 8.6

Remote Display Client

- **With the Remote Display Client window a user provides the information necessary for connectivity to the host, and saves the information in a profile.**
- **The information includes:**
 - User name, public key file, or other ssh options
 - The directory for TotalView or MemoryScape
 - The path name of the executable (full or relative to home directory)
 - For indirect connections the client provides for multiple intermediate host jumps, each by:
 - Host Name
 - Access type (User name, Public Key file, other SSH)
 - Access value
- **The Client also provides for submission of jobs to batch queuing systems PBS Pro and LoadLeveler**

TotalView 8.6

TV Remote Display Client

File Help



Session Profiles:

- arrays_with_sub
- stovepipe build

1. Enter the Remote Host to run your debug session:

Remote Host: User Name:

2. As needed, enter hosts in access order to reach the Remote Host:

	Host	Access By	Access Value
1		User Name	
2		User Name	

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

TotalView MemoryScope

Path to TotalView on Remote Host:

Arguments for TotalView:

Your Executable (path & name):

Arguments for Your Executable:

Submit job to Batch Queueing System:

4. Enter batch submission settings for the Remote Host:

PBS Submit Command:

TotalView PBS Script to run:

Additional PBS Options:

No session running

Remote Display

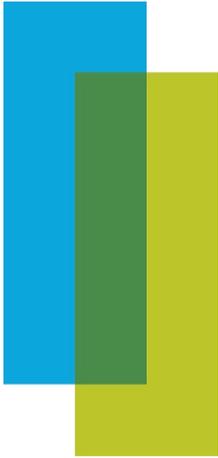
Remote Display

Session Profile Management

- Profiles are saved with profile names
- Multiple profiles can be generated for various environments
- Profiles can be exported and shared
- Shared Profile files can be imported and shared by other users

Remote Display Security

- Remote Display uses SSH
- Remote Display Server allows only RFB (Remote Frame Buffer) connections to and from the host
- No incoming access to the Server is allowed
- The Server can only connect back to the Viewer via SSH
- Only one Viewer connection is allowed to the Server
- As Remote Display connects to systems, the user is prompted for password information as required
 - Note: ssh and X Windows are required



Thanks!

QUESTIONS?

totalviewtech.com