Debugging Memory Problems on Cray XT3 Supercomputers with TotalView Debugger

Chris Gottbrath, Ariel Burton

TotalView Technologies

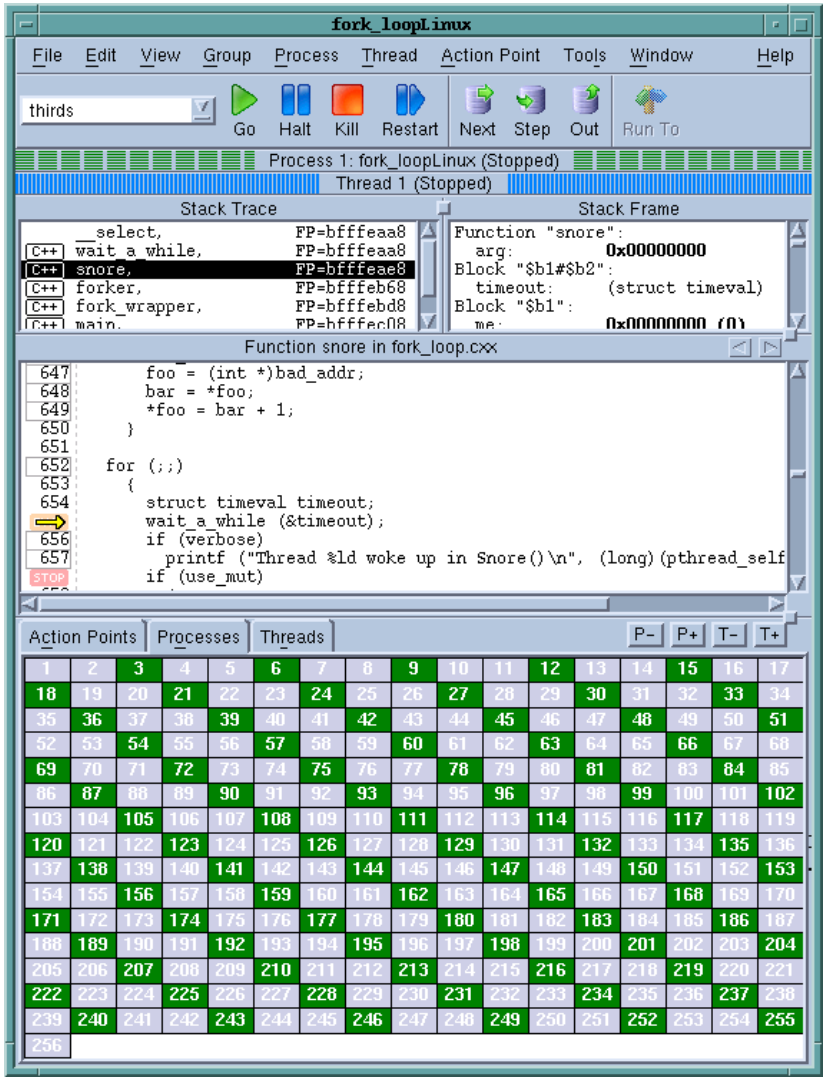
Robert Moench, Luiz DeRose

Cray

TOTALVIEW TECHNOLOGIES

What is TotalView?

- **Source Code Debugger**
  - C, C++, Fortran 77, Fortran90, UPC
    - Complex Language Features
  - Wide Compiler and Platform Support
  - Multi-Threaded Debugging
  - Parallel Debugging
    - MPI, PVM, Others
  - Remote Debugging
  - Memory Debugging capabilities
    - Integrated into the debugger
  - Powerful and Easy GUI
    - Visualization
  - CLI for Scripting

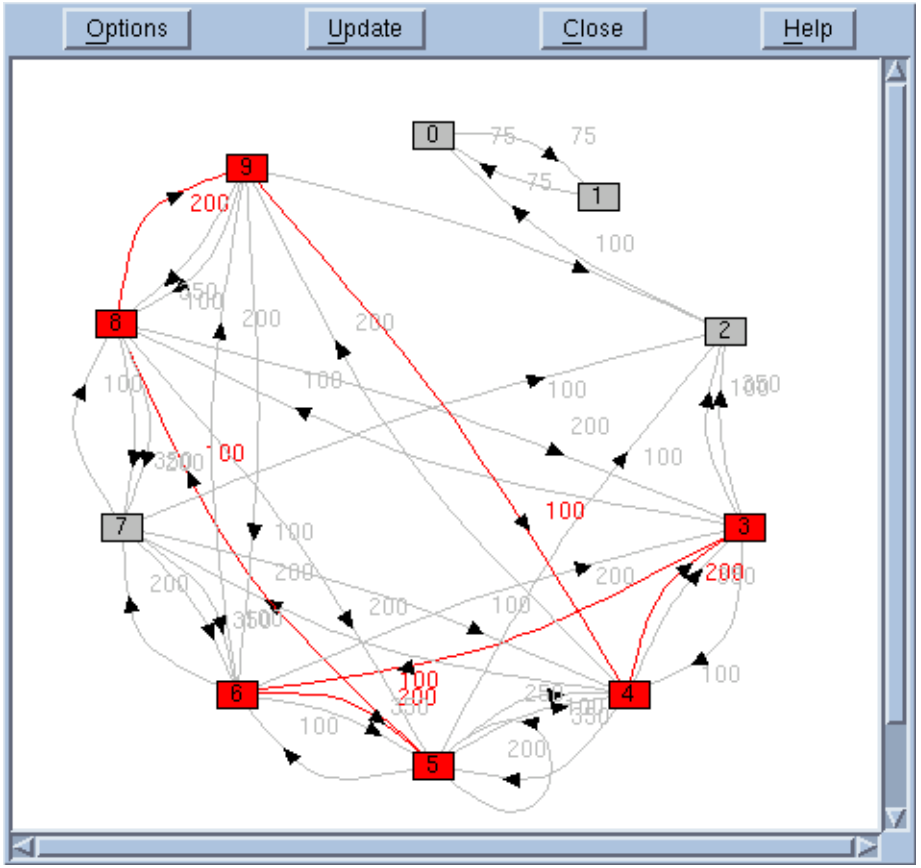


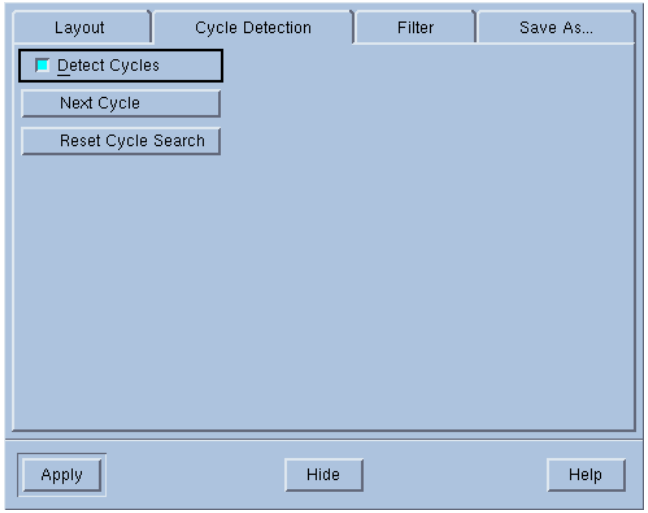
Supported Compilers, Distributions and Architectures

- **Platform Support**
  - Linux x86, x86-64, ia64, Power
  - Mac Power and Intel
  - Solaris Sparc and AMD64
  - AIX, Tru64, IRIX
  - Cray X1, XT3, IBM BGL
- **Languages / Compilers**
  - C/C++, Fortran, UPC, Assembly
  - Many Commercial & Open Source Compilers
- **Parallel Environments**
  - MPI (MPICH1 & 2, LAM, Open MPI, poe, MPT, Quadrics, MVAPICH, & many others)
  - UPC

Message Queue Debugging

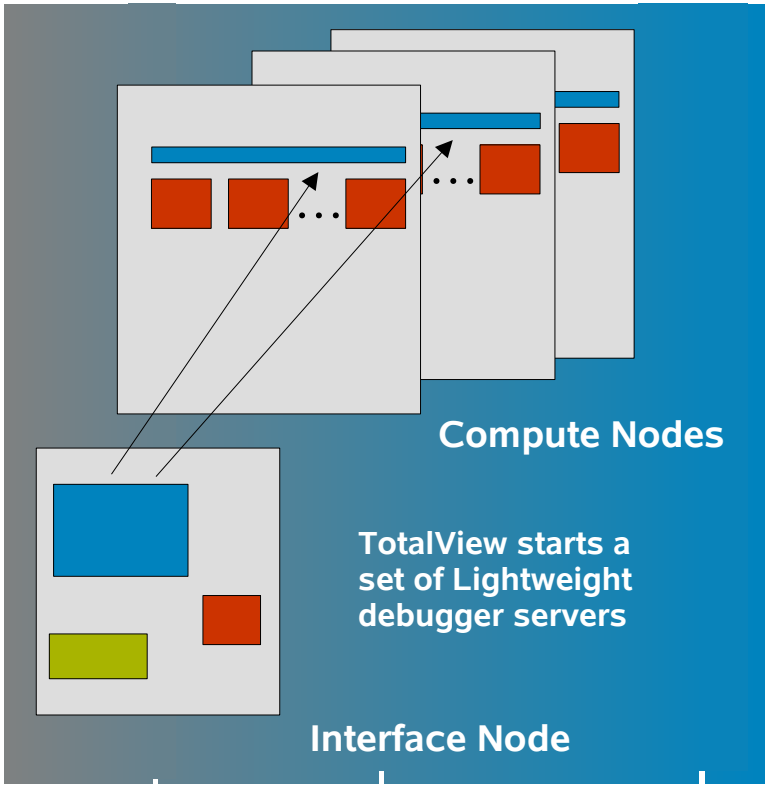
- **Message Queue Graph**
- **Message Inspection**
- **Cycle detection**
  - Find deadlocks





TotalView Parallel Debugger Architecture for Cluster Debugging

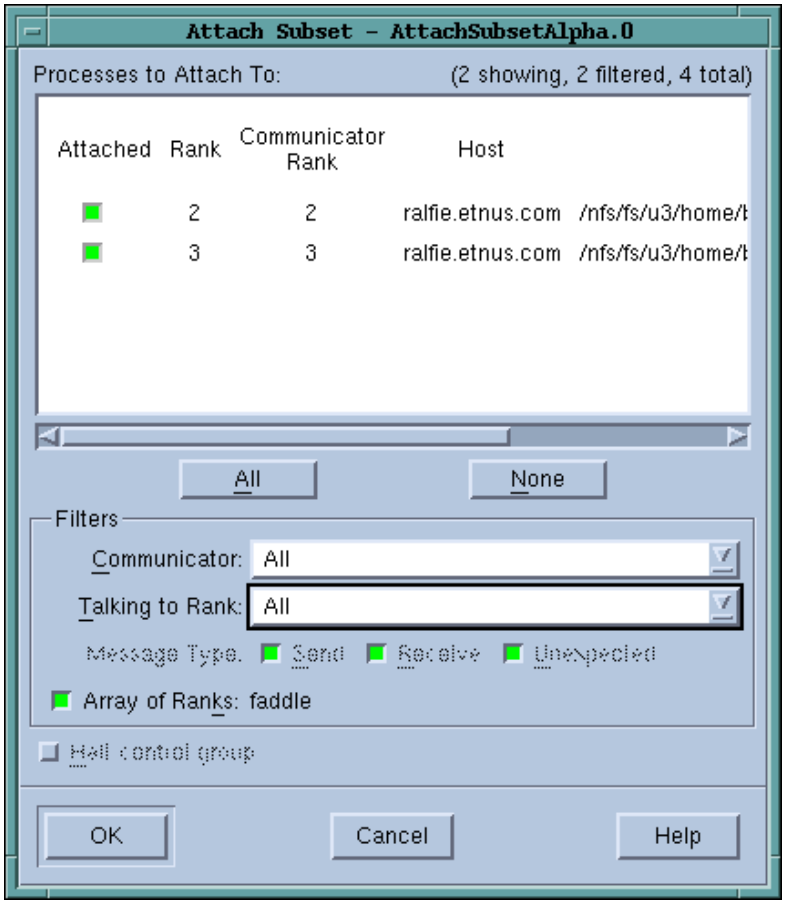
- **Cluster Architecture**
  - Single Front End (TotalView)
    - GUI and debug engine
  - Debugger Agents (tvdsvr)
    - Low overhead, 1 per node
    - Traces multiple rank processes
  - TotalView communicates directly with tvdsvrs
    - Not using MPI
    - Optimized Protocol



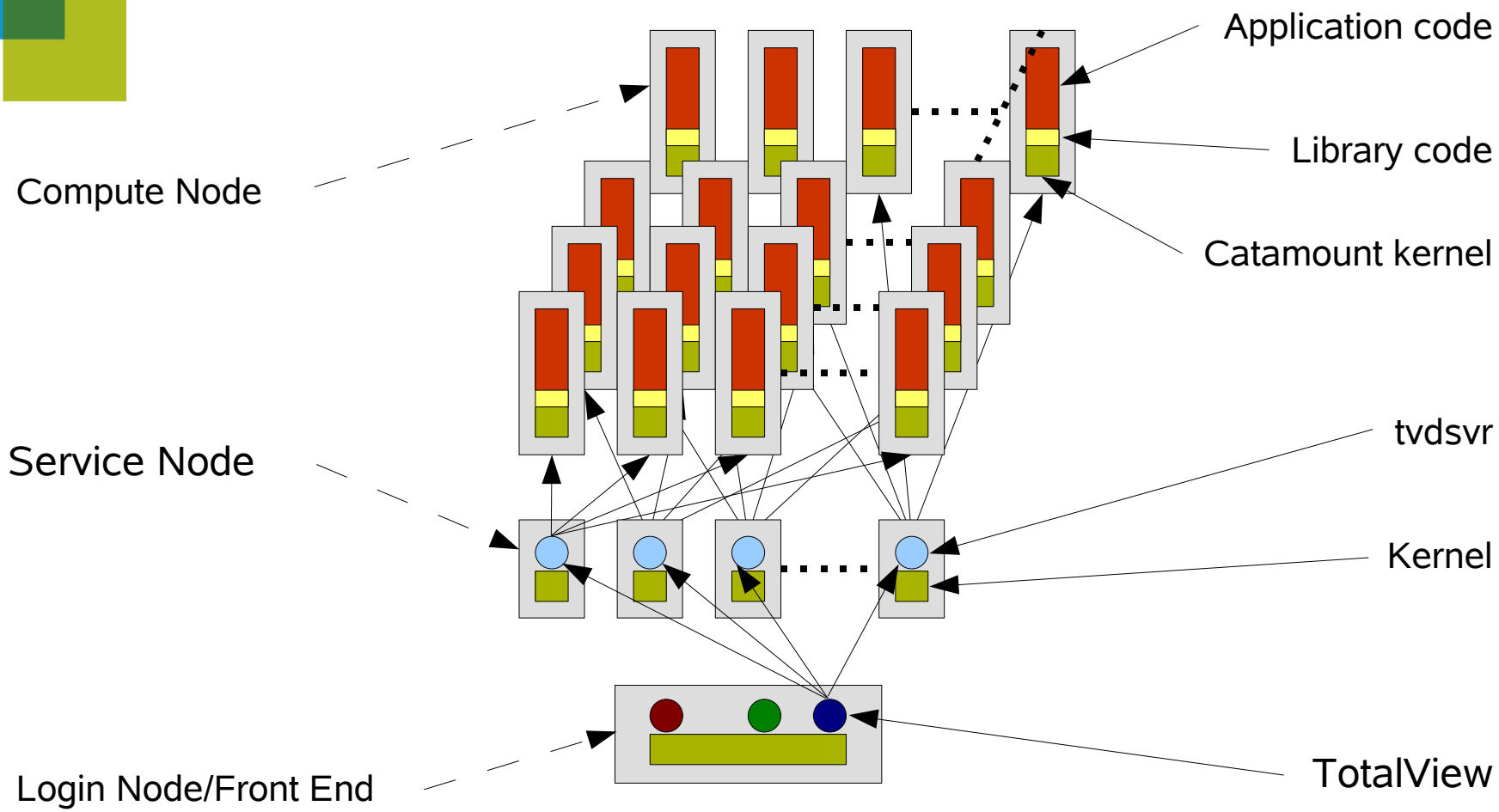
- **Provides: Robust, Scalable, Minimal Interaction**

Subset Attach

- **TotalView does not need to be attached to the entire job**
  - You can be attached to different subsets at different times through the run
    - You can attach to a subset, run till you see trouble and then 'fan out' to look at more processes if necessary.
  - This greatly reduces overhead
    - There is a danger of missing things



TotalView Parallel Debugger Architecture for the Cray XT3



TotalView Technologies

Memory Debugging with TotalView

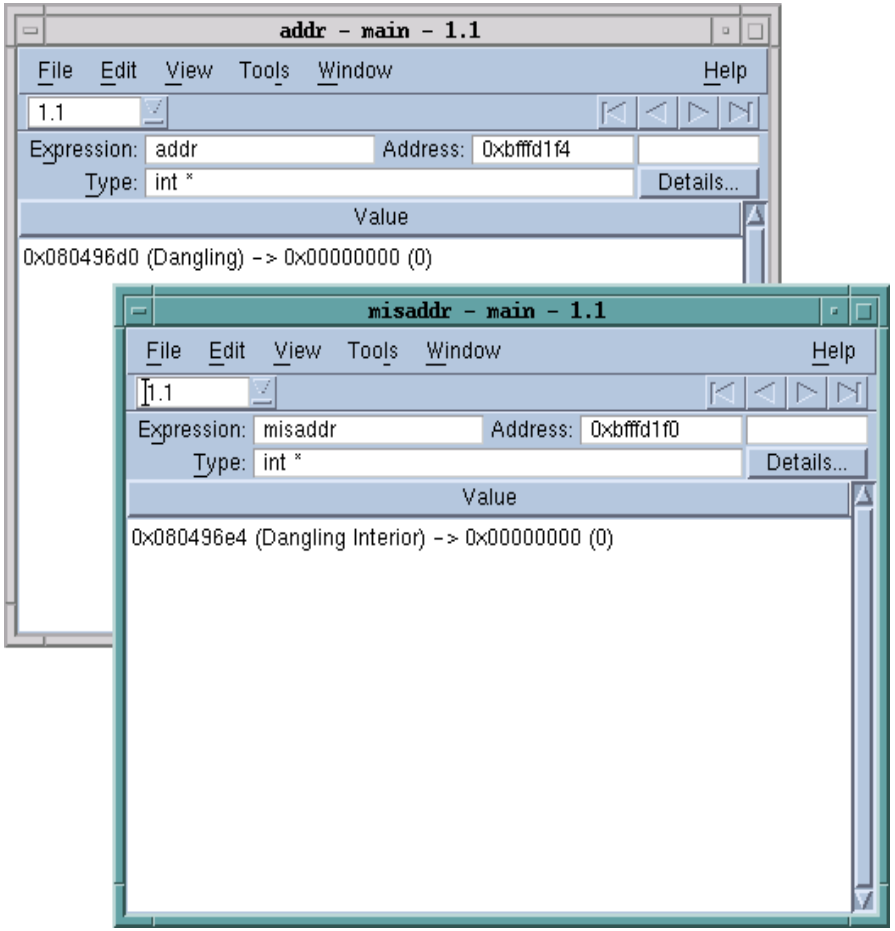
- **Application runs with a component called the Heap Interposition Agent (HIA)**
- **NO source code modification**
- **Usually engaged automatically by TotalView**
  - simple as starting the application under TotalView and enabling Memory Debugging in the GUI
  - sometimes more explicit steps are required
- **Monitors the application's interactions with the Heap Manager**
- **Integrated with the Debugger**
  - data displays annotated with information from the HIA
  - error and event notification
  - view the current state of the heap, compare with earlier state
- **Low overhead**

Enabling TotalView Memory Debugging on the Cray XT3

- **Cray XT3 Compute Node executables are statically linked**
  - executable must be linked with the HIA:
    - `cc -g app.c`
    - `cc -o app app.o -L`*`path`* `-ltvheap_xt3 -lgmalloc`
- **Normally a parallel job is started using the yod launcher:**
    - `yod -sz=256 app`
- **Instead, start TotalView on yod:**
    - `totalview yod -a -sz=256 app`

Integration with TotalView --- Pointer Annotation

- **Based on information from the HIA**
- **Shows**
  - Allocated
  - Allocated Interior
  - Deallocated
  - Deallocated Interior
  - Corrupted Guard Block(s)

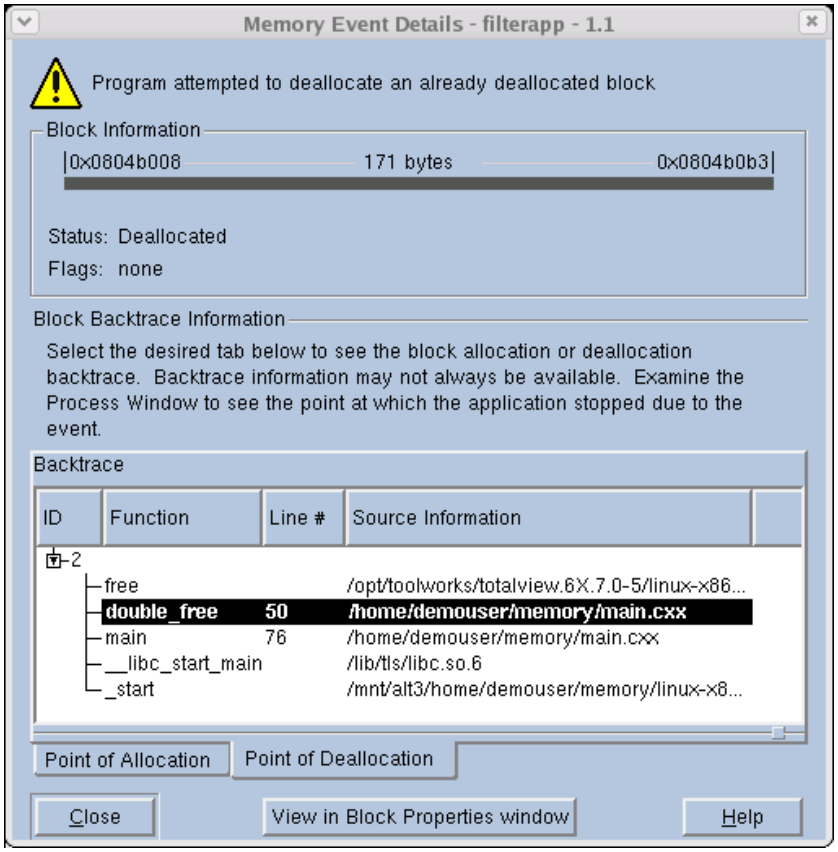


Memory Debugging with TotalView

- Heap Manager API Errors
- Read-before-Write --- reading uninitialized data
- Use-after-free --- dangling pointers
- Bounds Errors
- Leaks

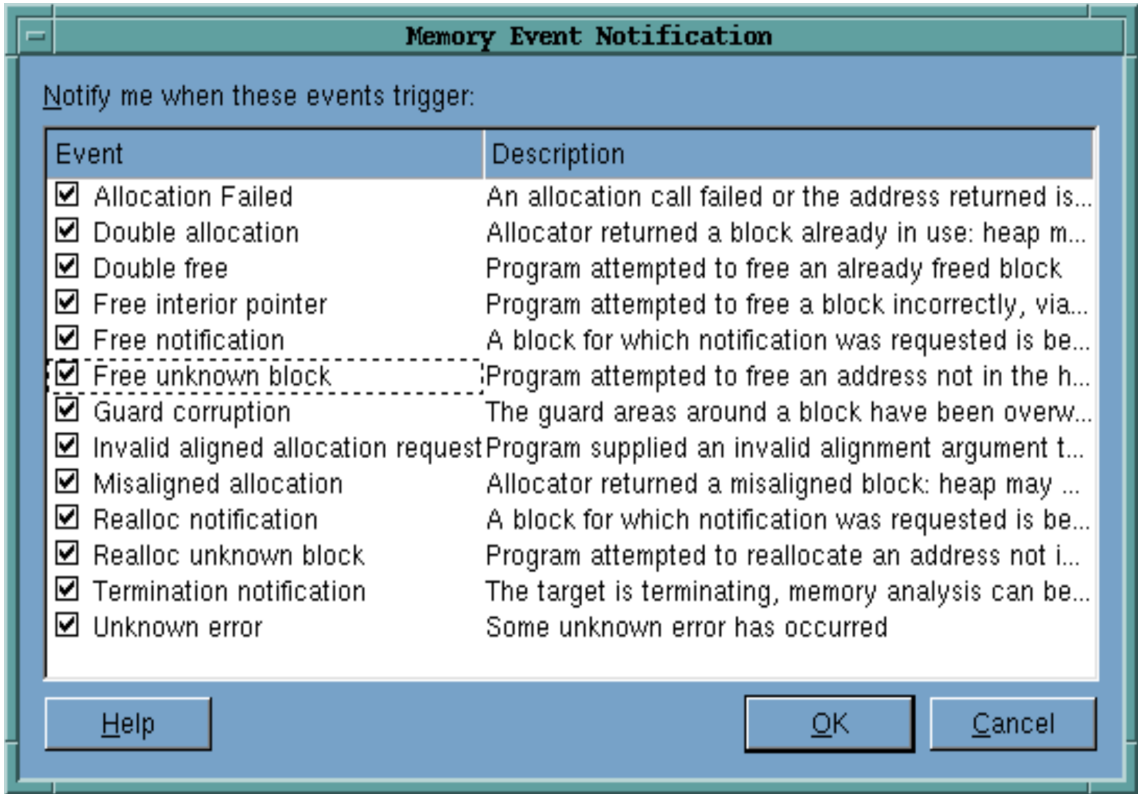
Heap Manager API Errors

- HIA monitors calls to the Heap Manager
- Checks arguments and return values
- Updates its tables
- Checks for errors, e.g.:
  - Double `free()`
  - `free()` interior
  - `free()` unknown
  - `realloc()` errors
  - Invalid alignment
  - Checks guards (more later)
- Notifies TotalView



Event Filtering

- **Notification can be restricted to a set of events of interest**



Read-before-Write --- Reading Uninitialized Data

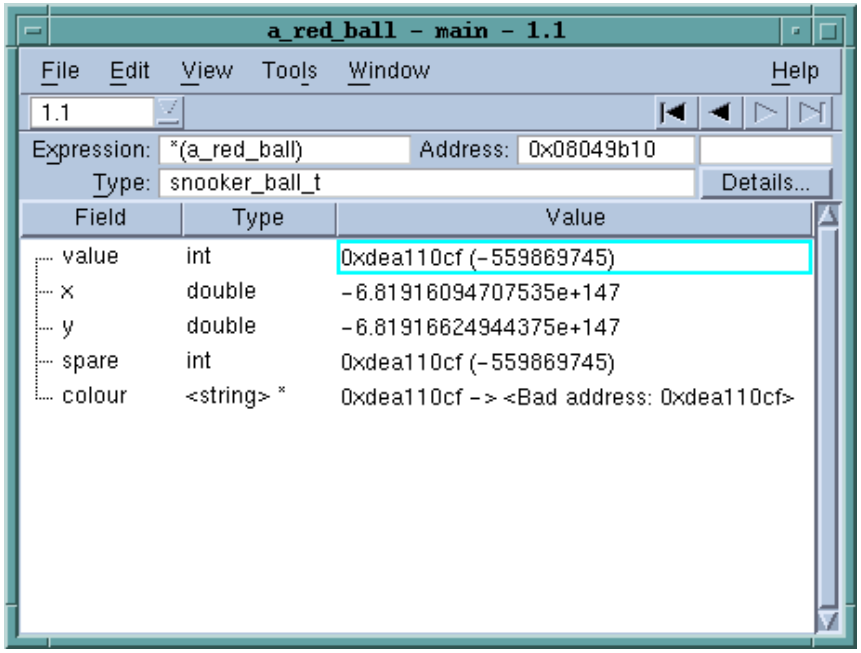
- Program reads from a newly allocated area before initializing its contents
- Can be difficult to find because a program may have worked in the past, or appears to fail non-deterministically
- Trivial example:

```
snooker_ball_t  *red   = malloc (  sizeof ( *red ) );
int              value = red->value;

current_score += value;
```

Painting

- The HIA can paint blocks on
  - allocation
  - deallocation
- Paint Pattern
  - defaults are unlikely values
  - can be customized
- Look for pattern
- Trigger fault on dereference
- Intended to provoke noticeable and consistent numerical errors in arithmetic, or trigger exception
- Temporarily fix problem



Use-after-Free --- Dangling Pointers

- Application continues to use a block after it has been released back to the Memory Manager
- Confusion over block ownership in complex codes with many libraries
- Can be difficult to find because point failure may depend on when block is reused
- TotalView can help:
  - annotations on data displays
  - painting
  - tagging
  - hoarding

Tagging and Hoarding

- **Tagging**
  - tag an allocation so that when it is passed to the Heap Manager for reuse, an event is raised
  - use when you know which block is being used-after-free, but don't know where the block is being freed
- **Hoarding**
  - released blocks are not immediately passed to the Heap Manager for reuse, but retained by the HIA
  - allows the application to run safely for a while after the premature deallocation

Bounds Errors

- **TotalView can help find certain bounds errors by adding *guard* regions to allocations**
  - optionally 'pre' and/or 'post' guards
  - sizes and patterns can be specified
  - alignment constraints are preserved
- **Guards checked by the HIA when a block is deallocated**
  - if a guard is found to be have been corrupted, an error is raised
- **Full guard check can be initiated at any time from TotalView**
- **Choice of patterns may trigger errors earlier (ala painting)**

Bounds errors/...

The guard areas around a block have been overwritten, suggesting a bounds error

Block Information

0x0804c068 — 64 bytes — 0x0804c0a7

Status: Allocated

Flags: Operation in Progress

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
-4			
	malloc	149	malloc_wrappers_dlopen.c
	corrupt_data	77	main.cxx
	main	126	main.cxx
	__libc_start_main		libc.so.6
	_start		filterapp

Source /home/demouser/memory/main.cxx

```
73  size = 16;
74
75  // Allocate some arrays.
76  p0 = (int *) malloc( size * sizeof( int ) );
77  p1 = (int *) malloc( size * sizeof( int ) );
78  p2 = (int *) malloc( size * sizeof( int ) );
```

Point of Allocation | Point of Deallocation

Close | View in Block Properties window | Help

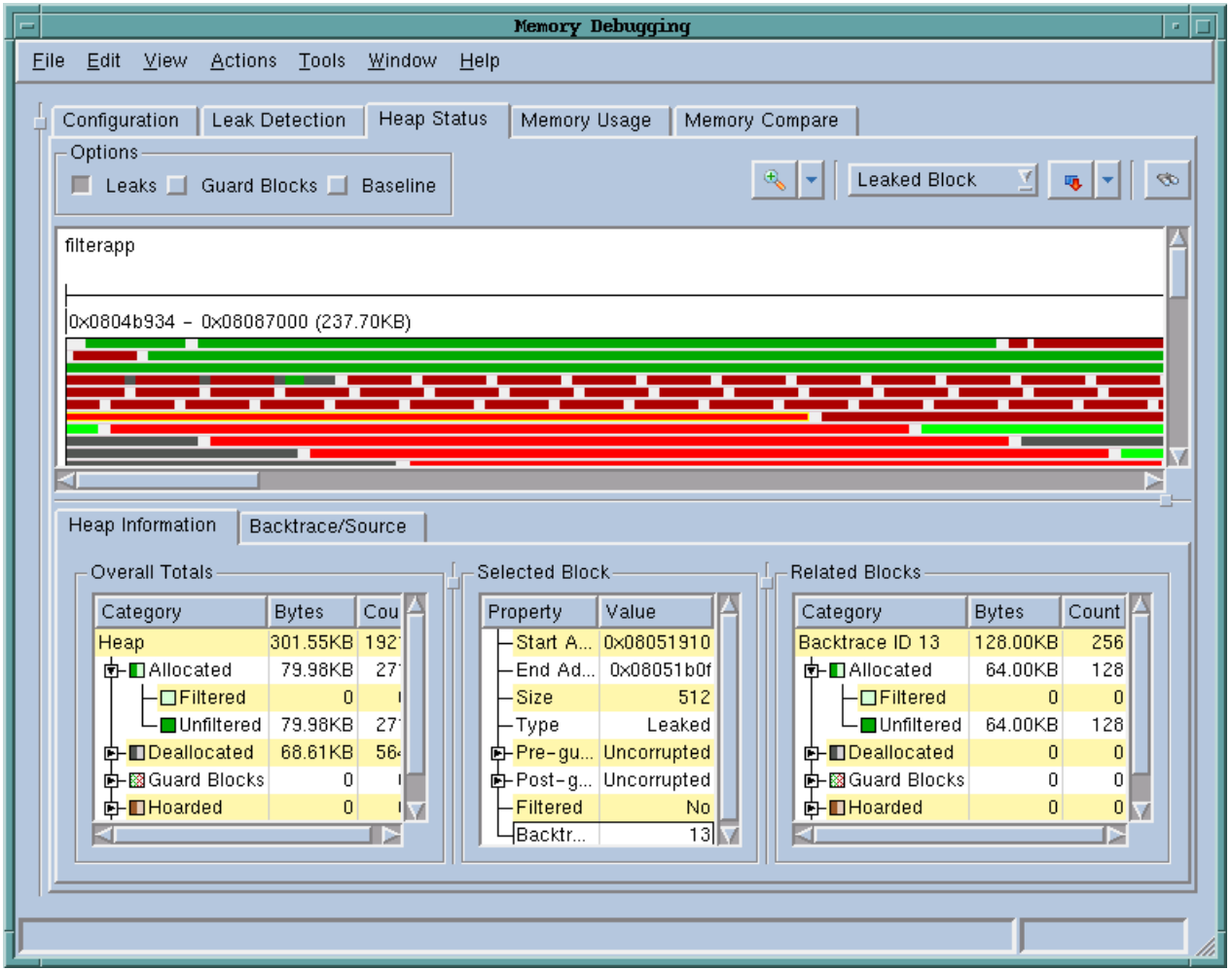
Leaks

- **Application deletes, or overwrites the last reference to a block before releasing the block**
- **Memory can no longer be accessed by the program, and cannot be reused by the Heap Manager**
- **Confusion over block ownership in complex codes with many libraries**
- **Performance loss, increase in resource usage**
- **TotalView can help:**
  - find leaks
  - heap reports and analysis
  - heap state comparisons

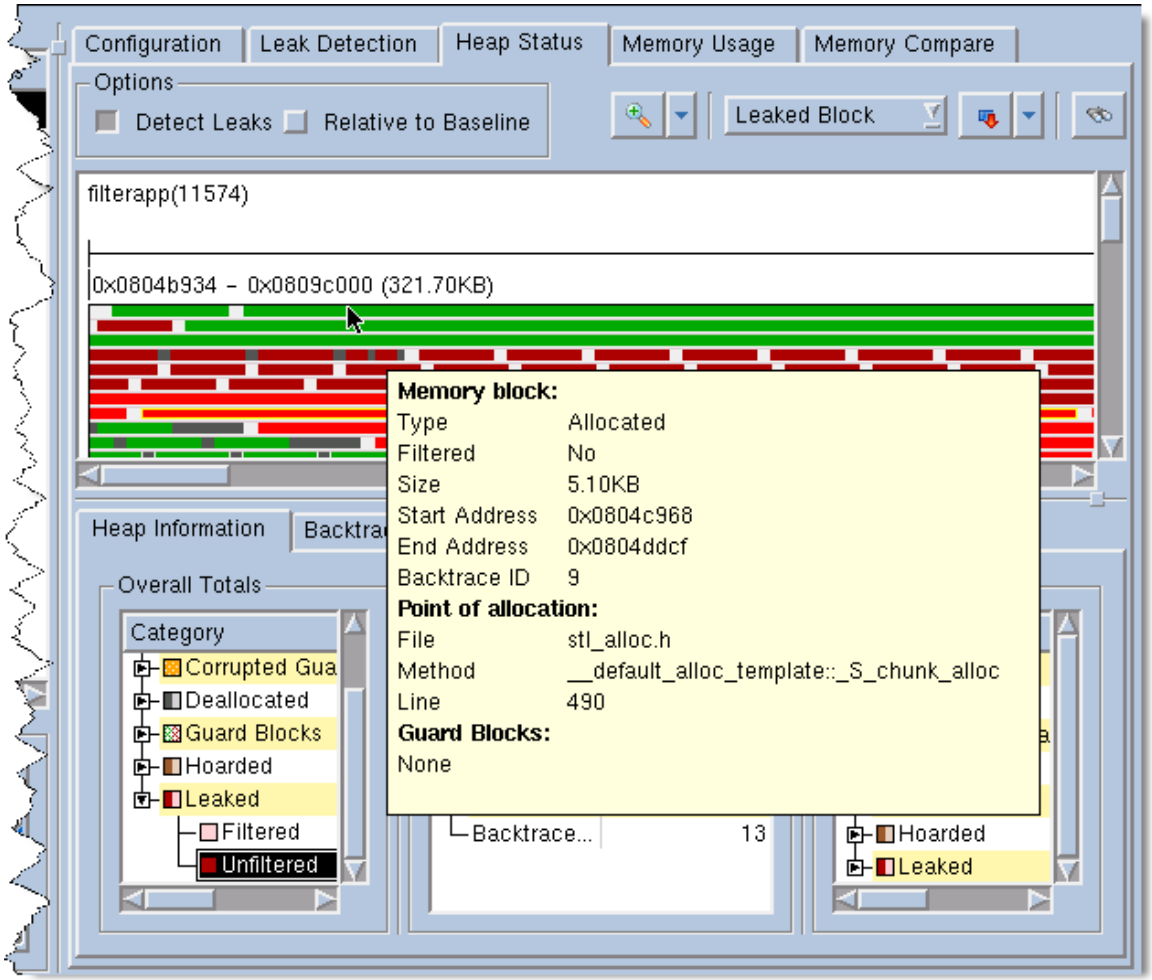
Leak Detection

- **Performed by TotalView at the request of the user**
- **Performs analysis similar to the first phases of a 'Mark-and-Sweep' Garbage Collector**
- **Conservative --- will not report anything active as a leak**
- **Results presented in TotalView's Heap Views:**
  - Heap Graphical View
  - Heap Source View

Heap Graphical View



Heap Graphical View/...



Heap Source View

Configuration | Leak Detection | Heap Status | Memory Usage | Memory Compare

Data Source: Allocations, Deallocations, Hoard

Options: Leaks, Baseline

Process	Bytes	Count	Begin Address	End Address
filterapp	164.32KB	793		
myClassB.cxx	151.50KB	771		
myClassB::init	131.00KB	259		
Line 36	128.00KB	256		
Line 33	3.00KB	3		
Block 3	1024	1	0x08071f00	0x080
Block 2	1024	1	0x08071af8	0x0807
Block 1	1024	1	0x08050ef0	0x0805
myClassB::myCla...	20.50KB	512		

Backtrace

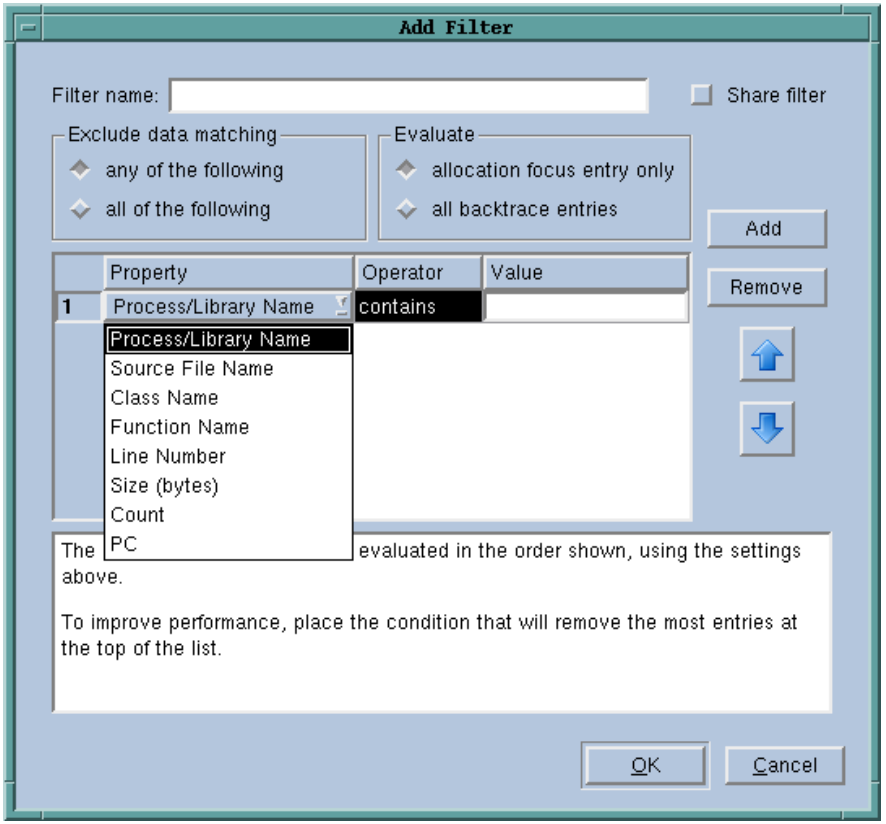
ID	Function	Line #	Source Informatic
12			
	malloc	149	malloc_wrappers_
	myClassB::init	33	myClassB.cxx
	myClassB::myClassB	11	myClassB.cxx
	main	19	main.cxx
	__libc_start_main		libc.so.6
	_start		filterapp

Source: ../myClassB.cxx

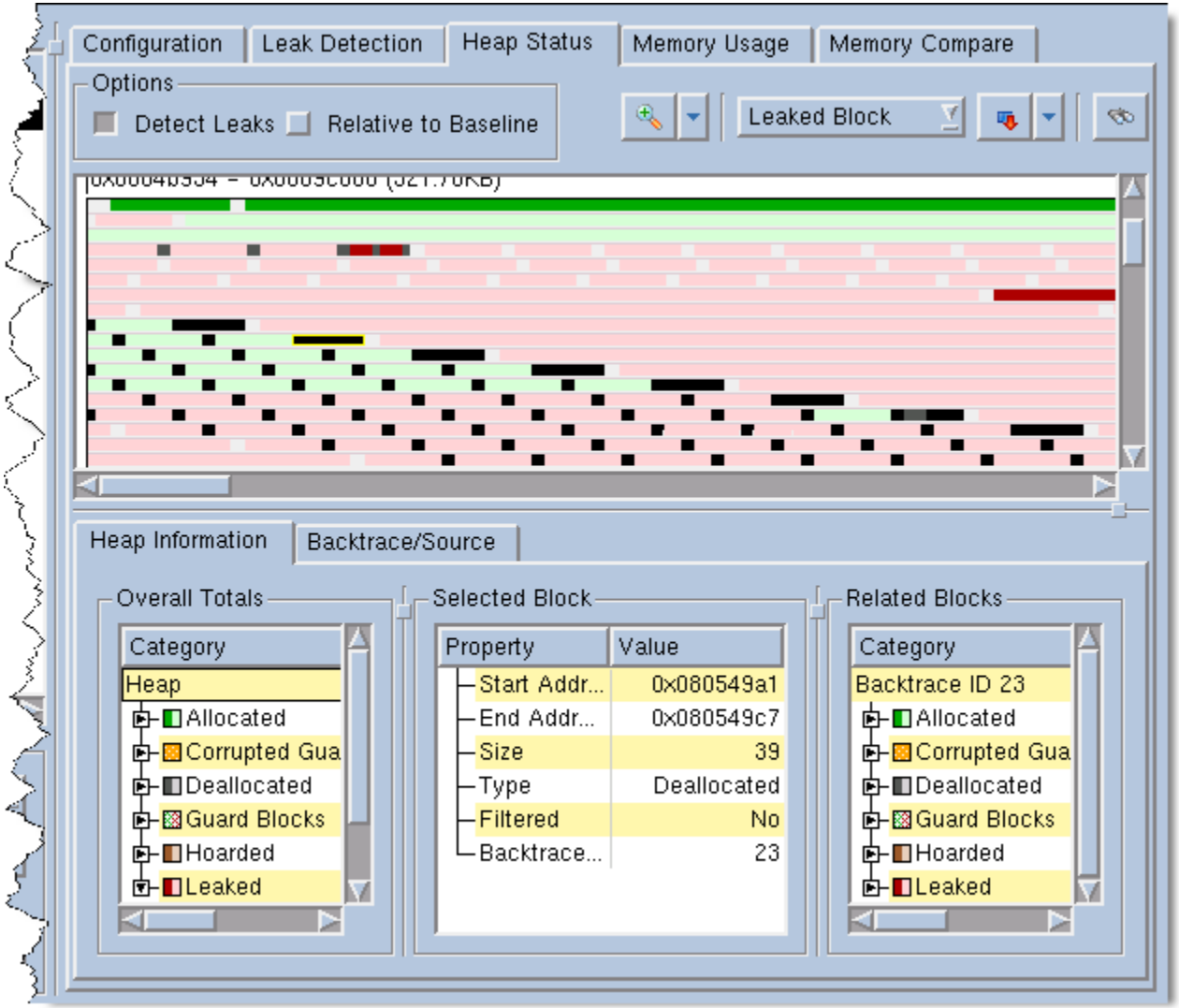
```
29  }
30
31  void myClassB::init(void) {
32
33    b_pp = (int **) malloc (size * si
34
```

Heap View Filters

- Filter views so that only blocks with certain properties are shown



Filtered Heap Graphical View



Heap Comparisons

- **At any point, save the state of the heap, including:**
  - allocated and deallocated blocks
  - leaks
  - guard states
  - full stack backtraces and source code snippets
- **Read in at a later time**
  - process may have terminated
- **Compare different snapshots**

Heap Comparisons/...

Memory Debugging

File Edit View Actions Tools Window Help

Process Set

Process

- Parallel Job fork_loopLinux (
  - fork_loopLinux
    - fork_loopLinux.1
      - fork_loopLinux
        - fork_loopLi
      - fork_loopLinux
    - fork_loopLinux.2
      - fork_loopLinux
    - fork_loopLinux.3
- filterapp(10/10/05 04:59
- filterapp(31163)
- free_doubleLinux(31161)

/fork_loopLinux.2.1
PID: 31131

Generate View

Source View

Enable Filtering

Generate View

Configuration | Leak Detection | Heap Status | Memory Usage | Memory Compare

Data Source

Allocations | Deallocations

Leaks | Hoard

Process Comparisons

Session 1: app(10/10/05 04:59 PM)

Session 2: filterapp(31163)

Reverse Diff

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**
assB.cxx	305.00KB	0	**305.00KB**	1544	0	**1544**
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**

Session 1 Source | Session 2 Source

TotalView Technologies

Try it Yourself!

- **Kick the Tires**
  - Sign up for a 15 day evaluation at http://www.totalviewtech.com
- **Get more Info**
  - Full Documentation available on line at http://www.totalviewtech.com
  - Watch a webcast at http://www.totalviewtech.com
    - Introduction to TotalView Source Code Debugger
    - Introduction to Memory Debugging
  - Contact us at info@totalviewtech.com