ROGUE WAVE®
SOFTWARE

# Leap to Petascale

## Developing Developer Tools
## TotalView and Blue Gene/Q

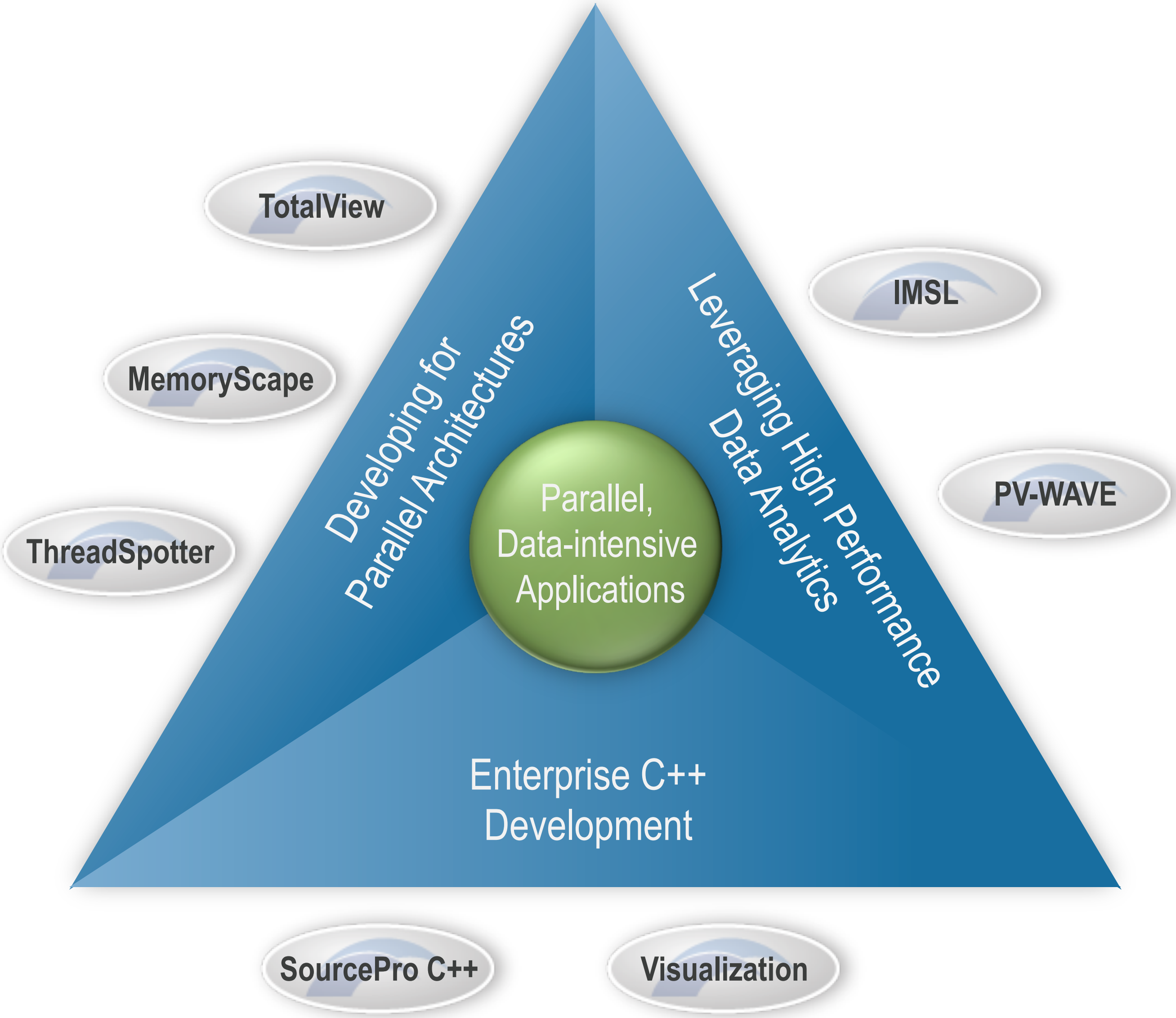
May 23 2012

Ed Hinkel,
Sr Sales Engineer
Rogue Wave Software

# Agenda

- Who is Rogue Wave?
- Early Blue Gene Days with TotalView
- Blue Gene/Q Advancements
- Techniques for Debugging Challenges
- What's New with TotalView

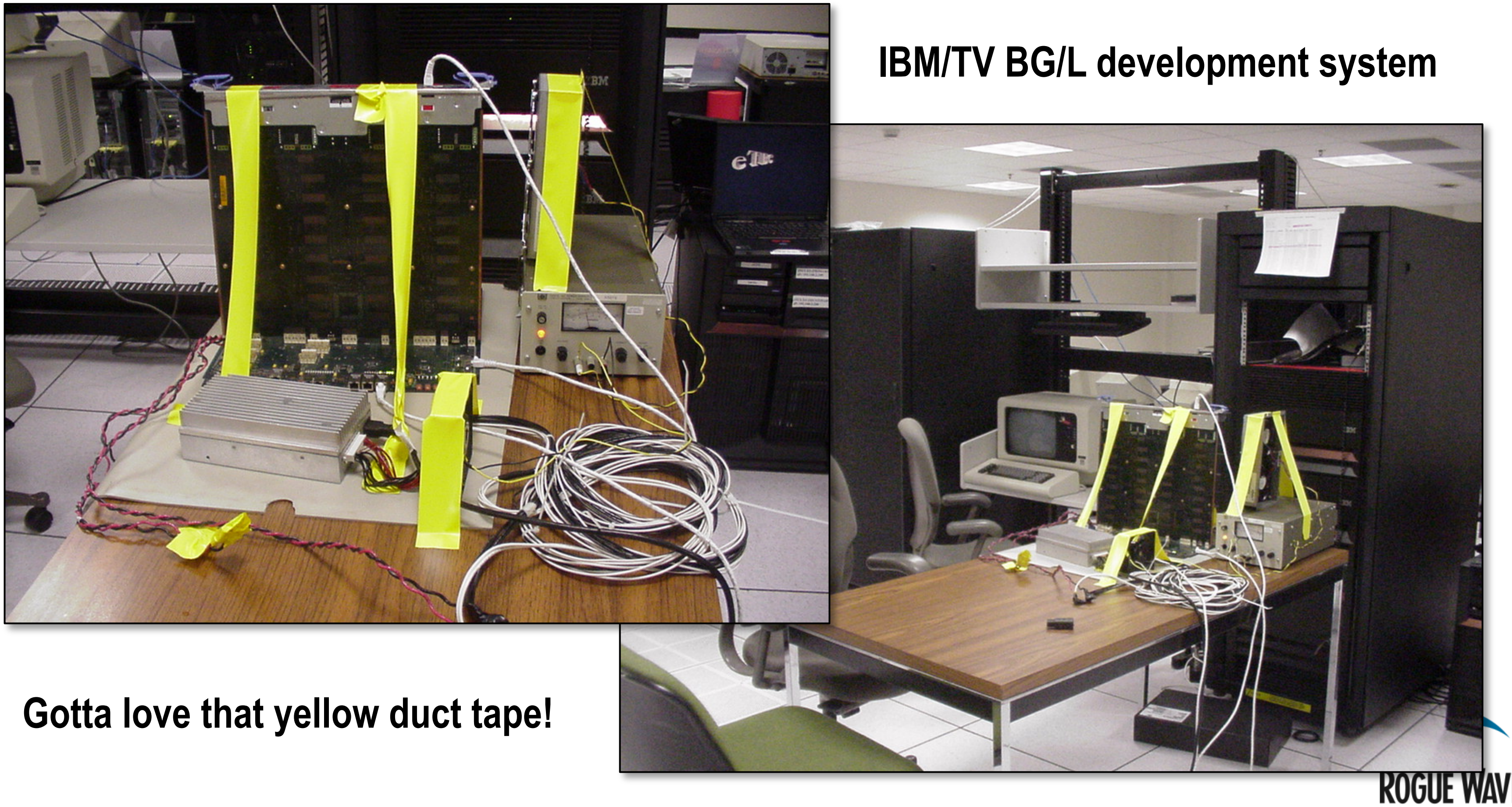
# Who is Rogue Wave Software?
# Solution Portfolio


TotalView
MemoryScape
ThreadSpotter
Developing for Parallel Architectures
IMSL
PV-WAVE
Leveraging High Performance Data Analytics
Parallel, Data-intensive Applications
Enterprise C++ Development
SourcePro C++
Visualization

# Early Blue Gene Days with TotalView

# TotalView Blue Gene Support

- **TotalView involvement started in 2003 on BG/L**

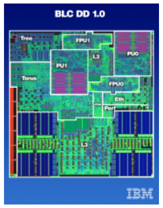


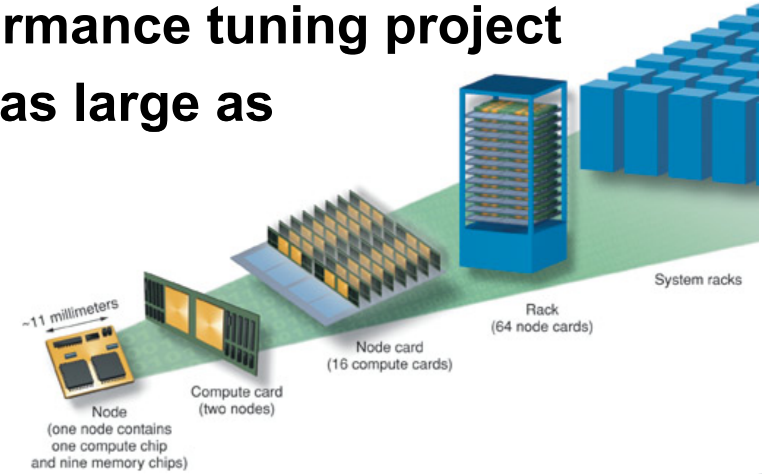
IBM/TV BG/L development system

Gotta love that yellow duct tape!

# TotalView Blue Gene/L Support

- Support for Blue Gene/L since 2005
- Debugging interfaces developed via close collaboration with IBM (CIOD)
- Used on DOE/NNSA/LLNL's Blue Gene/L system containing 212 K cores
  - Heap memory debugging support added
  - Blue Gene/L scaling and performance tuning project
  - TotalView has debugged jobs as large as 32,768 processes

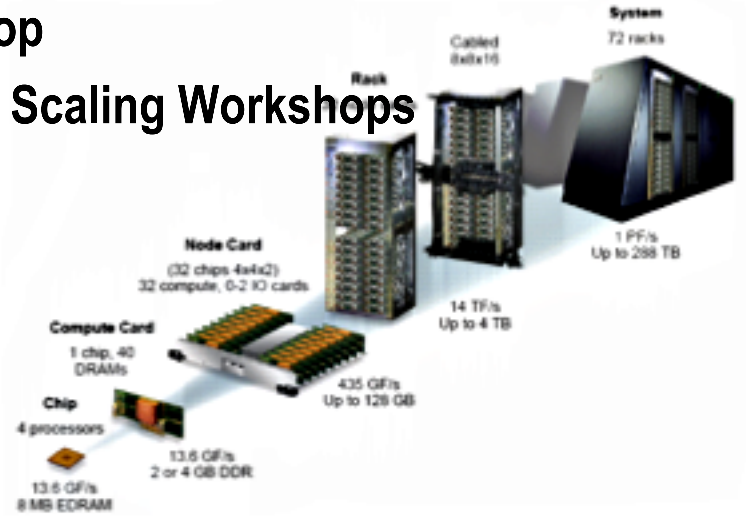




Blue Gene/L work facilitated Blue Gene/P support

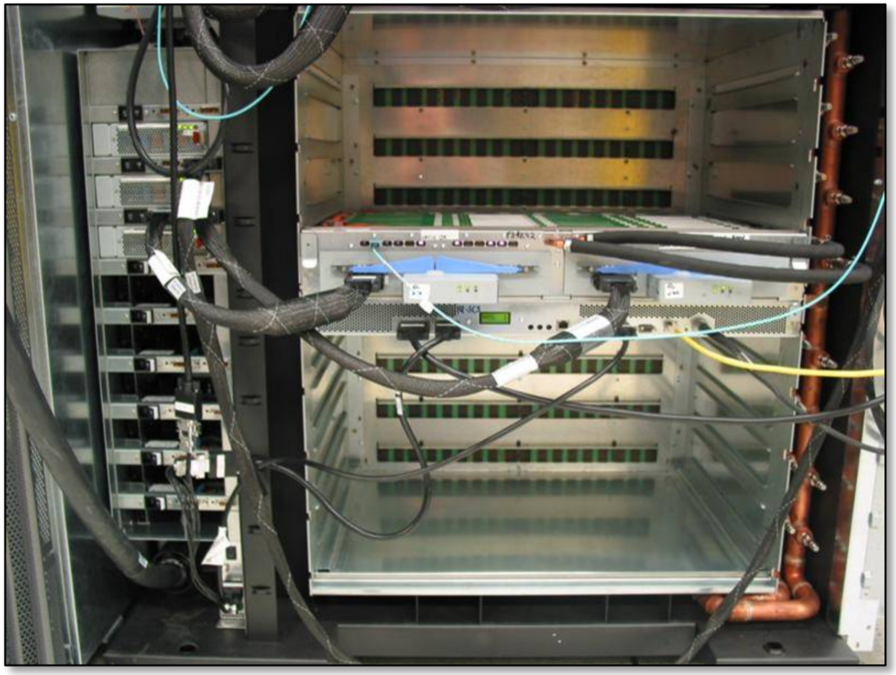
# TotalView Blue Gene/P Support

- Continued close collaboration with IBM
- Currently running on several BG/P installations in Germany, France, the UK, and the US.
- Support for shared libraries, threads, and OpenMP
- TotalView has debugged jobs as large as 32,768
- Active workshop participation through the development
  - ANL’s ALCF INCITE Performance Workshop
  - Jülich’s Blue Gene/P Porting, Tuning, and Scaling Workshops



# TotalView Blue Gene/Q Support

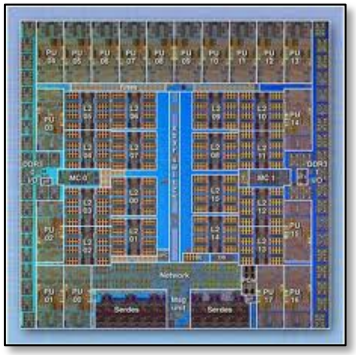
- Porting TotalView began in June 2011
- Access to Q32 at IBM began in August
- Basic debugging operations in October
- Used in Synthetic Workload Testing in December
- Fully functional in March 2012

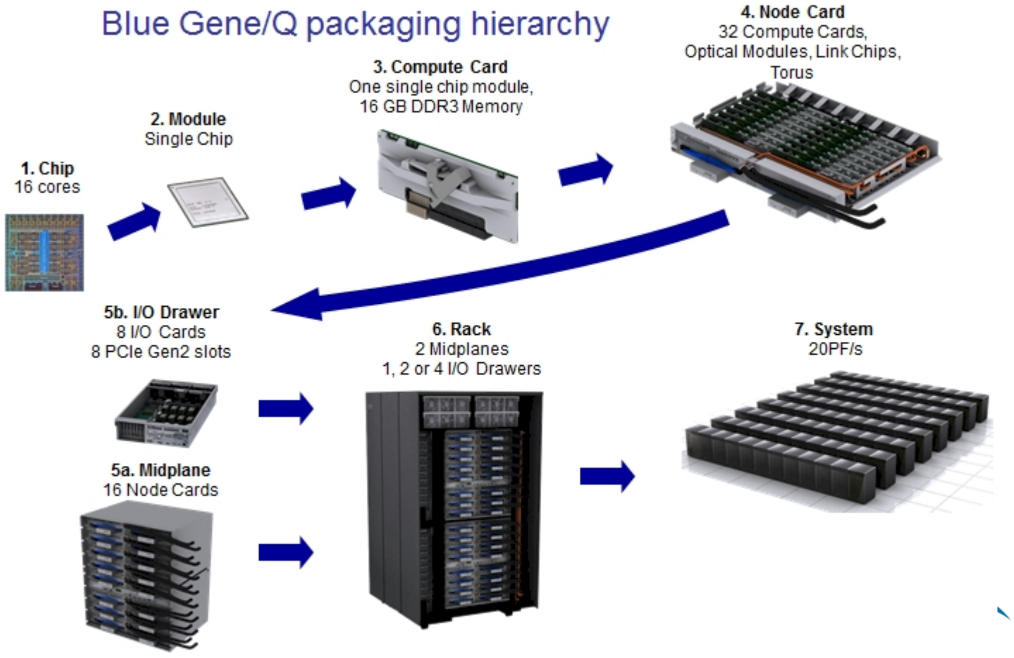


IBM's Q32

# TotalView Blue Gene/Q Support (cont)

- Thanks to the ongoing collaboration with IBM and the BG Kernel Team, early access versions of TotalView for BG/Q isavailable
- Argonne National Laboratory
- Lawrence Livermore National Laboratory





# Blue Gene/Q Advancements

# with

# TotalView

# TotalView on BG/Q Support

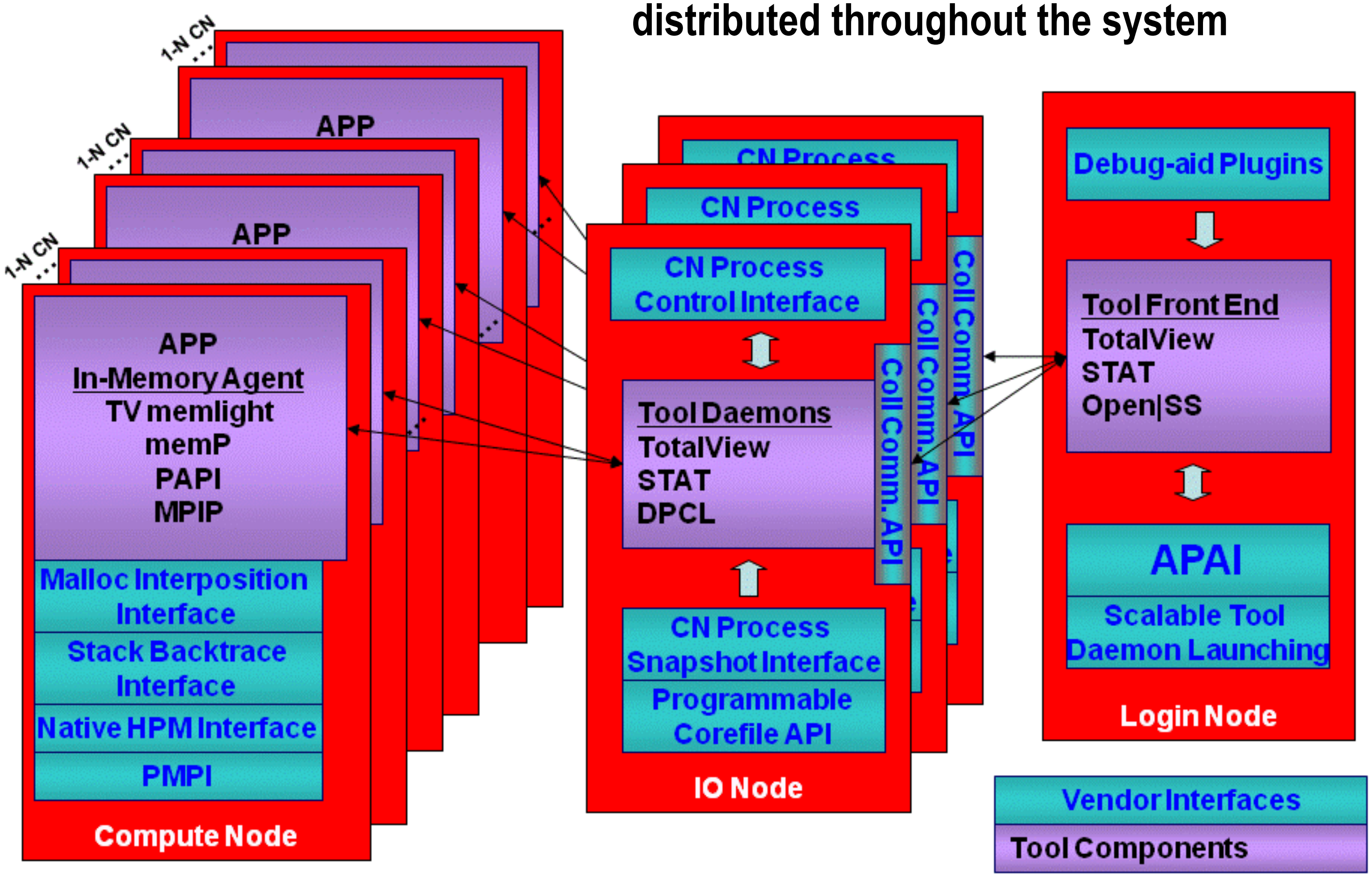
- BG/Q TotalView is as functional as BG/P TotalView
  - MPI, OpenMP, pthreads, hybrid MPI+threads
  - C, C++, Fortran, assembler; IBM and GNU compilers
  - Basics: source code, variables, breakpoints, watchpoints, stacks, single stepping, read/write memory/registers, conditional breakpoints, etc.
  - Memory debugging, message queues, binary core files, etc.
- PLUS, features unique to BG/Q TotalView
  - QPX (floating point) instruction set and register model
  - Fast compiled conditional breakpoints and watchpoints
  - Asynchronous thread control
- Working with IBM on debugging interfaces for TM/SE regions
  - TM == transactional memory; SE == speculative execution

# Advanced TotalView Features on BG/Q

- Asynchronous thread control
  - A feature on Linux, and other TotalView platforms, ported to BG/Q
  - Allows you to individually control the execution of threads
  - Run and halt individual threads
  - Single-step a group of threads in lockstep
  - Hold and release the execution of individual threads
  - Create stop-thread and thread barrier breakpoints
- Fast compiled conditional breakpoints and watchpoints
  - A feature on AIX and other TotalView platforms, ported to BG/Q
  - Conditional breakpoints and watchpoints execute in as little as 7 μsecs
  - Conditional expressions are compiled and dynamically patched into the process
  - Evaluation is performed by the triggering thread, in parallel

# Blue Gene Code Development Tools Interface (CDTI)

Hierarchical infrastructure components are distributed throughout the system



1-N CN

1-N CN

1-N CN

APP

APP

APP

In-Memory Agent

TV memlight

memP

PAPI

MPIP

Malloc Interposition Interface

Stack Backtrace Interface

Native HPM Interface

PMPI

Compute Node

CN Process

CN Process

CN Process Control Interface

Tool Daemons

TotalView

STAT

DPCL

CN Process Snapshot Interface

Programmable Corefile API

Coll Comm. API

Coll Comm. API

Coll Comm. API

IO Node

Debug-aid Plugins

Tool Front End

TotalView

STAT

Open|SS

APAI

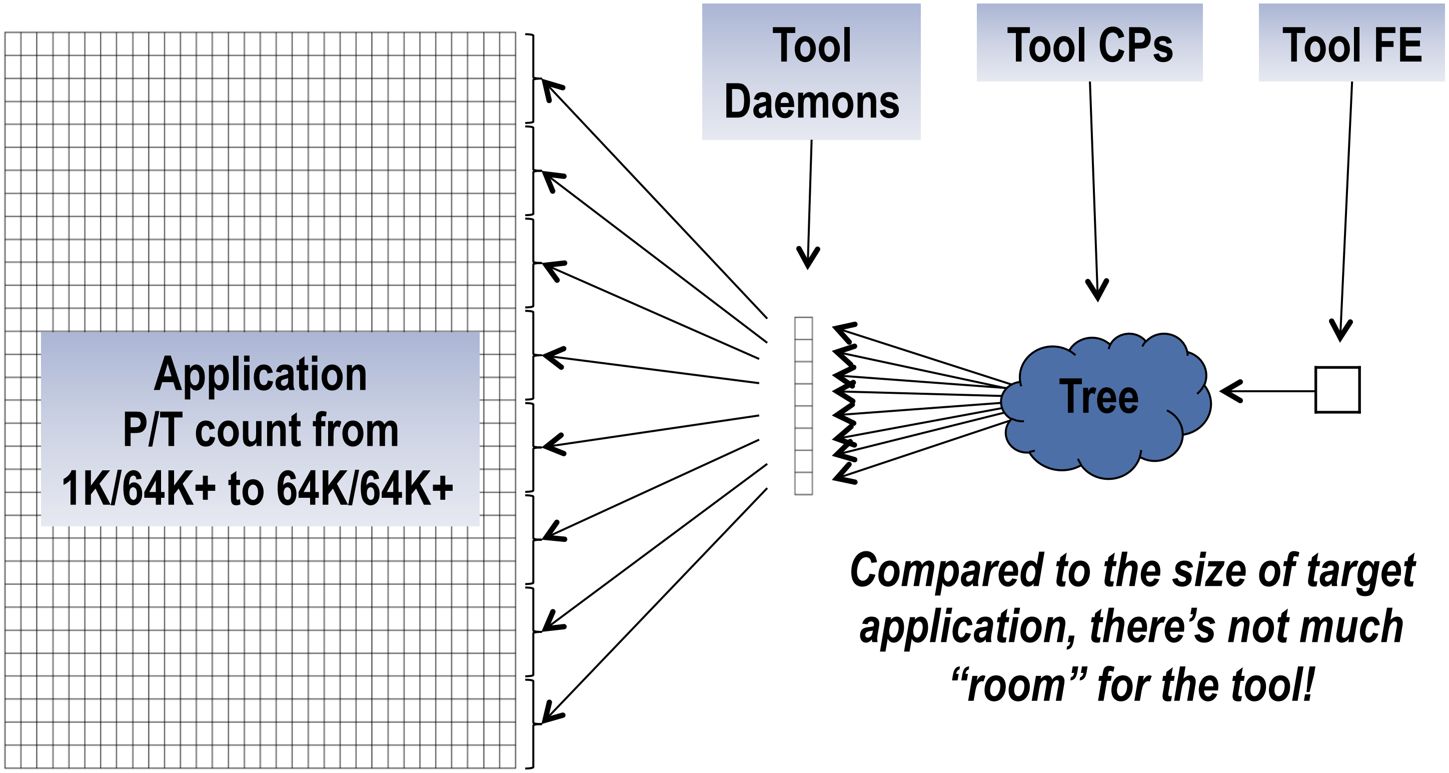
Scalable Tool Daemon Launching

Login Node

Vendor Interfaces

Tool Components

# Tool Challenges



Tool Daemons

Tool CPs

Tool FE

Application
P/T count from
1K/64K+ to 64K/64K+

Tree

*Compared to the size of target application, there's not much "room" for the tool!*

One rack of BG/Q:
1K CNs, 16K Cores,
64K HW Threads

A "generous"
128:1 CN:ION
ratio: 8 IONs

A "beefy" FEN
P7, 3 GHz+, 32 GB+

# Overcoming High CN:ION Ratios

- On BG/Q, at a given ratio, on *each* IO node, tool daemons may be responsible for up to

| CN:ION | Processes | Threads |
|---|---|---|
| 64:1 | 4,096 | 20,480 |
| 128:1 | 8,192 | 40,960 |
| 256:1 | 16,384 | 81,920 |
| 512:1 | 32,768 | 163,840 |

- But each IO node has
  - 1.6 GHz A2 17 core processor (not too swift)
  - 16 GB (limited physical memory)

# Where to put the “weight” of the debugger?

- Most of the “weight” of the debugger is in the symbol table
- Real-world applications are huge and complex
- A recently analyzed mission critical application revealed
  - 1.5 million function definitions
  - 16 million line number definitions
  - DWARF symbol information in excess of 2 GB
  - 100s or 1000 of shared libraries
- You don’t want to be big in the back end!
- And nothing too compute intensive either

# TotalView's Architecture

- Extremely lightweight back-end daemon processes
  - Small footprint plus a few hundred bytes per CN process or thread
  - Each daemon can handle thousands of processes and threads
  - The daemons do not store the symbol table!
- The "weight" of the debugger is on the front-end node
- Symbol tables are indexed and stored on the FEN
  - Debugger has exactly *one* copy of the symbol table for each image file
  - Symbol tables are *shared* across all processes and thread
  - Aggregate memory consumption is minimal

- Well suited to Blue Gene!

# There’s still the high P/T count per IO node problem

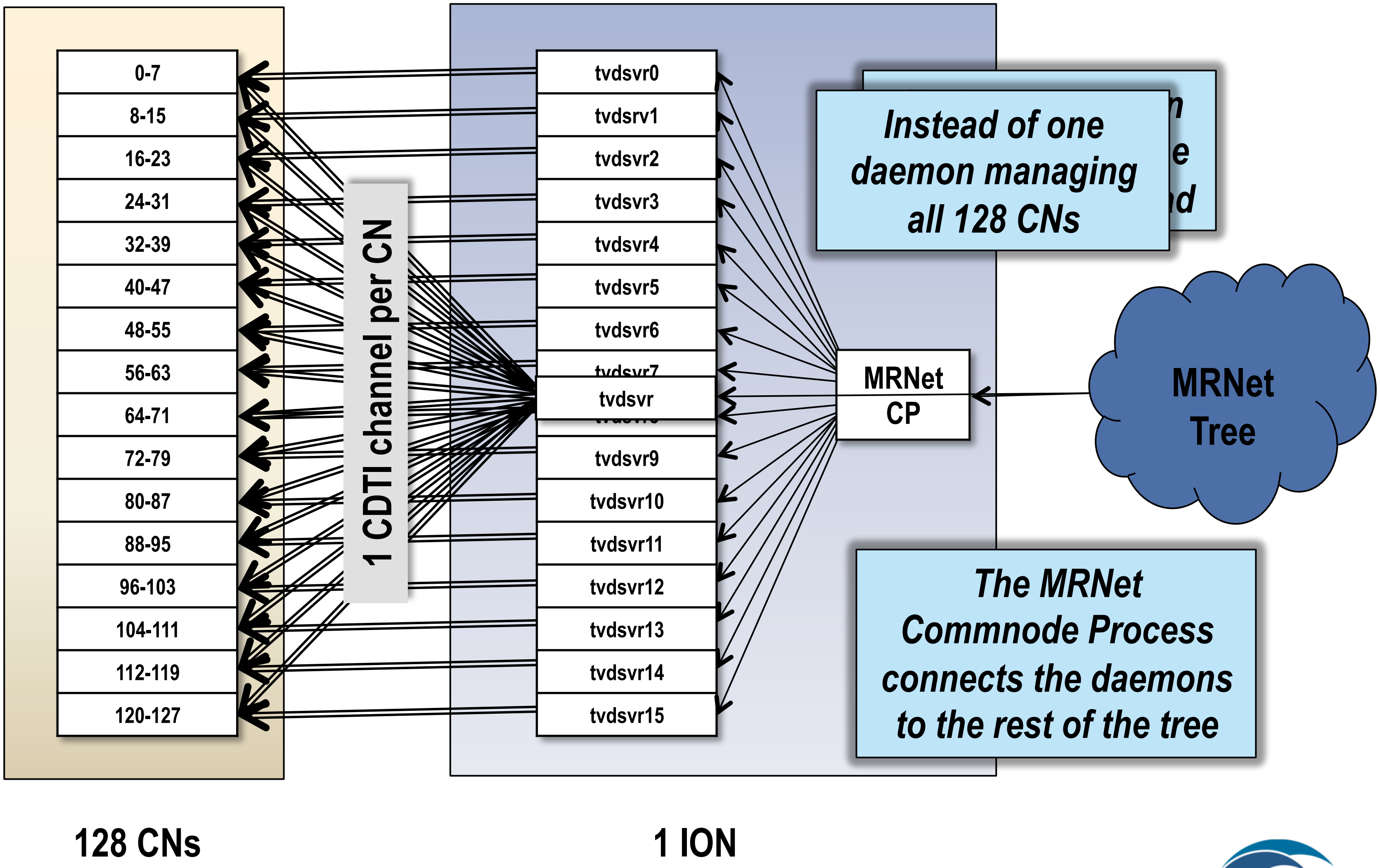
- Process and threads counts per IO node are still high!

- What to do about that?

- “Divide and conquer”
  - Place a small number of daemons on the ION
  - We do have 17 cores we can use

- Unlike CIOD on BG/L&P, CDTI on BG/Q can operate in parallel
  - There’s one CDTI debug channel per compute node

# Solution: TotalView/MRNet Trees on the IO Nodes



*Instead of one daemon managing all 128 CNs*

*The MRNet Commnode Process connects the daemons to the rest of the tree*

# TotalView on BG/Q Support

- BG/Q TotalView is as functional as BG/P TotalView
  - MPI, OpenMP, pthreads, hybrid MPI+threads
  - C, C++, Fortran, assembler; IBM and GNU compilers
  - Basics: source code, variables, breakpoints, watchpoints, stacks, single stepping, read/write memory/registers, conditional breakpoints, static/dynamic executables, etc.
  - Memory debugging, message queues, binary core files, etc.
- PLUS, advanced BG/Q TotalView features
  - QPX (floating point) instruction set and register model
  - Fast compiled conditional breakpoints and watchpoints
  - Asynchronous thread control
- Working with IBM on debug interfaces for TM/SE regions
  - TM == transactional memory; SE == speculative execution

# Advanced BG/Q TotalView Features

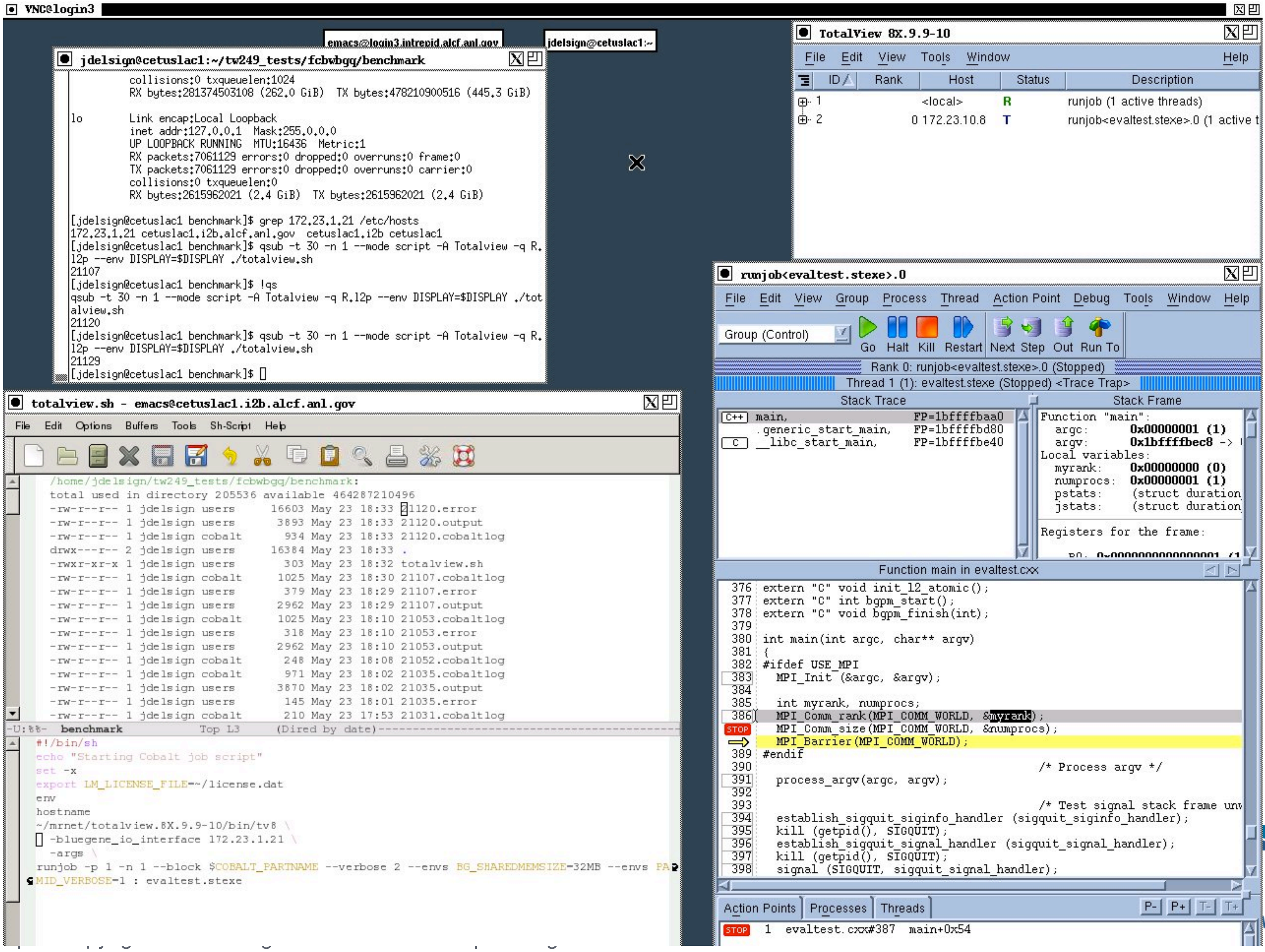
- **Asynchronous thread control**
  - A TotalView feature on Linux and other platforms, ported to BG/Q
  - Allows you to individually control the execution of threads
  - Run and halt individual threads
  - Single-step a group of threads in lockstep
  - Hold and release the execution of individual threads
  - Create stop-thread and thread barrier breakpoints
- **Fast compiled conditional breakpoints and watchpoints**
  - A TotalView feature on AIX and other platforms, ported to BG/Q
  - Conditional breakpoints and watchpoints execute in as little as 7 μsecs
  - Conditional expressions are compiled and dynamically patched into the process
  - Evaluation is performed in parallel by the triggering thread

# TotalView Availability

- TotalView on Blue Gene/Q Today
  - LLNL has it up and running on rzuseq, and is using it to debug applications.
  - IBM is using it internally for debugging and testing.
  - It's installed on IBM's Blue Gene On Demand Center Q32 (if anyone has access to that system).

- TotalView On Blue Gene at Argonne
  - 1024 Tokens (BG/P)
  - Research license is available with 65,536 tokens

# TotalView on VEAS!



jdelsign@cetuslac1:~/tw249_tests/fcbwbgq/benchmark

```
          collisions:0 txqueuelen:1024
          RX bytes:281374503108 (262.0 GiB)  TX bytes:478210900516 (445.3 GiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:7061129 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7061129 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2615962021 (2.4 GiB)  TX bytes:2615962021 (2.4 GiB)

[jdelsign@cetuslac1 benchmark]$ grep 172.23.1.21 /etc/hosts
172.23.1.21 cetuslac1.i2b.alcf.anl.gov  cetuslac1.i2b cetuslac1
[jdelsign@cetuslac1 benchmark]$ qsub -t 30 -n 1 --mode script -A Totalview -q R.
12p --env DISPLAY=$DISPLAY ./totalview.sh
21107
[jdelsign@cetuslac1 benchmark]$ !qs
qsub -t 30 -n 1 --mode script -A Totalview -q R.12p --env DISPLAY=$DISPLAY ./tot
alview.sh
21120
[jdelsign@cetuslac1 benchmark]$ qsub -t 30 -n 1 --mode script -A Totalview -q R.
12p --env DISPLAY=$DISPLAY ./totalview.sh
21129
[jdelsign@cetuslac1 benchmark]$ 
```

totalview.sh - emacs@cetuslac1.i2b.alcf.anl.gov

```
/home/jdelsign/tw249_tests/fcbwbgq/benchmark:
total used in directory 205536 available 464287210496
-rw-r--r-- 1 jdelsign users     16603 May 23 18:33 21120.error
-rw-r--r-- 1 jdelsign users      3893 May 23 18:33 21120.output
-rw-r--r-- 1 jdelsign cobalt      934 May 23 18:33 21120.cobaltlog
drwx---r-- 2 jdelsign users     16384 May 23 18:33 .
-rwxr-xr-x 1 jdelsign users       303 May 23 18:32 totalview.sh
-rw-r--r-- 1 jdelsign cobalt     1025 May 23 18:30 21107.cobaltlog
-rw-r--r-- 1 jdelsign users       379 May 23 18:29 21107.error
-rw-r--r-- 1 jdelsign users      2962 May 23 18:29 21107.output
-rw-r--r-- 1 jdelsign cobalt     1025 May 23 18:10 21053.cobaltlog
-rw-r--r-- 1 jdelsign users       318 May 23 18:10 21053.error
-rw-r--r-- 1 jdelsign users      2962 May 23 18:10 21053.output
-rw-r--r-- 1 jdelsign cobalt      248 May 23 18:08 21052.cobaltlog
-rw-r--r-- 1 jdelsign cobalt      971 May 23 18:02 21035.cobaltlog
-rw-r--r-- 1 jdelsign users      3870 May 23 18:02 21035.output
-rw-r--r-- 1 jdelsign users       145 May 23 18:01 21035.error
-rw-r--r-- 1 jdelsign cobalt      210 May 23 17:53 21031.cobaltlog
-U:%%-  benchmark          Top L3      (Dired by date)-----------------------------
```

```
#!/bin/sh
echo "Starting Cobalt job script"
set -x
export LM_LICENSE_FILE=~/license.dat
env
hostname
~/mrnet/totalview.8X.9.9-10/bin/tv8 \
 -bluegene_io_interface 172.23.1.21 \
  -args \
runjob -p 1 -n 1 --block $COBALT_PARTNAME --verbose 2 --envs BG_SHAREDMEMSIZE=32MB --envs PA
MID_VERBOSE=1 : evaltest.stexe
```

TotalView 8X.9.9-10

```
ID   Rank  Host          Status  Description
⊞ 1        <local>       R       runjob (1 active threads)
⊞ 2        0 172.23.10.8 T       runjob<evaltest.stexe>.0 (1 active t
```

runjob<evaltest.stexe>.0

File Edit View Group Process Thread Action Point Debug Tools Window Help
Group (Control)
Go
Halt
Kill
Restart
Next
Step
Out
Run To

Rank 0: runjob<evaltest.stexe>.0 (Stopped)

Thread 1 (1): evaltest.stexe (Stopped) <Trace Trap>

Stack Trace

```
C++  main,                    FP=1bffffbaa0
     .generic_start_main,     FP=1bffffbd80
C    __libc_start_main,       FP=1bffffbe40
```

Stack Frame

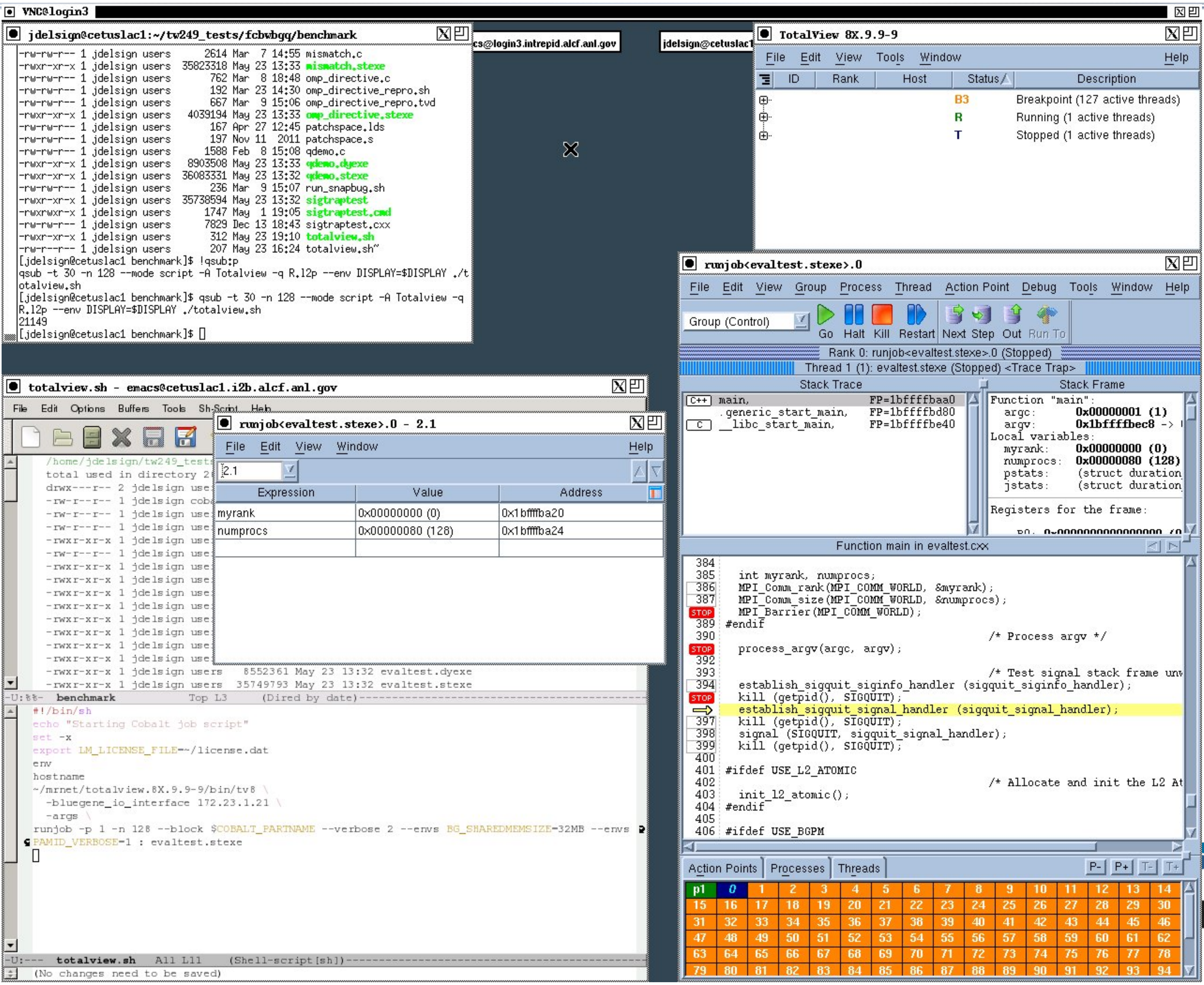
```
Function "main":
  argc:       0x00000001 (1)
  argv:       0x1bffffbec8 ->
Local variables:
  myrank:     0x00000000 (0)
  numprocs:   0x00000001 (1)
  pstats:     (struct duration
  jstats:     (struct duration

Registers for the frame:
```

Function main in evaltest.cxx

```
extern "C" void init_l2_atomic();
extern "C" int bgpm_start();
extern "C" void bgpm_finish(int);

int main(int argc, char** argv)
{
#ifdef USE_MPI
  MPI_Init (&argc, &argv);

  int myrank, numprocs;
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Barrier(MPI_COMM_WORLD);
#endif
                                        /* Process argv */
  process_argv(argc, argv);

                                        /* Test signal stack frame unw
  establish_sigquit_siginfo_handler (sigquit_siginfo_handler);
  kill (getpid(), SIGQUIT);
  establish_sigquit_signal_handler (sigquit_signal_handler);
  kill (getpid(), SIGQUIT);
  signal (SIGQUIT, sigquit_signal_handler);
```

```
STOP  1  evaltest.cxx#387  main+0x54
```

# TotalView on VEAS!



jdelsign@cetuslac1:~/tw249_tests/fcbwbgq/benchmark

```
-rw-rw-r-- 1 jdelsign users     2614 Mar  7 14:55 mismatch.c
-rwxr-xr-x 1 jdelsign users 35823318 May 23 13:33 mismatch.stexe
-rw-rw-r-- 1 jdelsign users      762 Mar  8 18:48 omp_directive.c
-rw-rw-r-- 1 jdelsign users      192 Mar 23 14:30 omp_directive_repro.sh
-rw-rw-r-- 1 jdelsign users      667 Mar  9 15:06 omp_directive_repro.tvd
-rwxr-xr-x 1 jdelsign users  4039194 May 23 13:33 omp_directive.stexe
-rw-rw-r-- 1 jdelsign users      167 Apr 27 12:45 patchspace.lds
-rw-rw-r-- 1 jdelsign users      197 Nov 11  2011 patchspace.s
-rw-rw-r-- 1 jdelsign users     1588 Feb  8 15:08 qdemo.c
-rwxr-xr-x 1 jdelsign users  8903508 May 23 13:33 qdemo.dyexe
-rwxr-xr-x 1 jdelsign users 36083331 May 23 13:32 qdemo.stexe
-rw-rw-r-- 1 jdelsign users      236 Mar  9 15:07 run_snapbug.sh
-rwxr-xr-x 1 jdelsign users 35738594 May 23 13:32 sigtraptest
-rwxrwxr-x 1 jdelsign users     1747 May  1 19:05 sigtraptest.cmd
-rw-rw-r-- 1 jdelsign users     7829 Dec 13 18:43 sigtraptest.cxx
-rwxr-xr-x 1 jdelsign users      312 May 23 19:10 totalview.sh
-rw-r--r-- 1 jdelsign users      207 May 23 16:24 totalview.sh~
[jdelsign@cetuslac1 benchmark]$ !qsub:p
qsub -t 30 -n 128 --mode script -A Totalview -q R.12p --env DISPLAY=$DISPLAY ./t
otalview.sh
[jdelsign@cetuslac1 benchmark]$ qsub -t 30 -n 128 --mode script -A Totalview -q
R.12p --env DISPLAY=$DISPLAY ./totalview.sh
21149
[jdelsign@cetuslac1 benchmark]$ 
```

totalview.sh - emacs@cetuslac1.i2b.alcf.anl.gov

```
  /home/jdelsign/tw249_tests
  total used in directory 2
  drwx---r-- 2 jdelsign use
  -rw-r--r-- 1 jdelsign cob
  -rw-r--r-- 1 jdelsign use
  -rw-r--r-- 1 jdelsign use
  -rwxr-xr-x 1 jdelsign use
  -rw-r--r-- 1 jdelsign use
  -rwxr-xr-x 1 jdelsign use
  -rwxr-xr-x 1 jdelsign use
  -rwxr-xr-x 1 jdelsign use
  -rwxr-xr-x 1 jdelsign use
  -rwxr-xr-x 1 jdelsign use
  -rwxr-xr-x 1 jdelsign use
  -rwxr-xr-x 1 jdelsign use
  -rwxr-xr-x 1 jdelsign use
  -rwxr-xr-x 1 jdelsign users   8552361 May 23 13:32 evaltest.dyexe
  -rwxr-xr-x 1 jdelsign users  35749793 May 23 13:32 evaltest.stexe
-U:%%-  benchmark         Top L3     (Dired by date)------------------------------
```

```
#!/bin/sh
echo "Starting Cobalt job script"
set -x
export LM_LICENSE_FILE=~/license.dat
env
hostname
~/mrnet/totalview.8X.9.9-9/bin/tv8 \
  -bluegene_io_interface 172.23.1.21 \
  -args \
runjob -p 1 -n 128 --block $COBALT_PARTNAME --verbose 2 --envs BG_SHAREDMEMSIZE=32MB --envs 
PAMID_VERBOSE=1 : evaltest.stexe

-U:---  totalview.sh    All L11     (Shell-script[sh])----------------------------
(No changes need to be saved)
```

runjob<evaltest.stexe>.0 - 2.1

| Expression | Value | Address |
|---|---|---|
| myrank | 0x00000000 (0) | 0x1bffffba20 |
| numprocs | 0x00000080 (128) | 0x1bffffba24 |
| | | |

TotalView 8X.9.9-9

| ID | Rank | Host | Status | Description |
|---|---|---|---|---|
| ⊞ | | | B3 | Breakpoint (127 active threads) |
| ⊞ | | | R | Running (1 active threads) |
| ⊞ | | | T | Stopped (1 active threads) |

runjob<evaltest.stexe>.0

Rank 0: runjob<evaltest.stexe>.0 (Stopped)

Thread 1 (1): evaltest.stexe (Stopped) <Trace Trap>

Stack Trace

```
C++  main,                  FP=1bffffbaa0
     .generic_start_main,   FP=1bffffbd80
C    __libc_start_main,     FP=1bffffbe40
```

Stack Frame

```
Function "main":
  argc:       0x00000001 (1)
  argv:       0x1bffffbec8 -> 
Local variables:
  myrank:     0x00000000 (0)
  numprocs:   0x00000080 (128)
  pstats:     (struct duration
  jstats:     (struct duration

Registers for the frame:

   R0:  0x0000000000000000 (0
```

Function main in evaltest.cxx

```

    int myrank, numprocs;
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
STOP    MPI_Barrier(MPI_COMM_WORLD);
  #endif
                                           /* Process argv */
STOP    process_argv(argc, argv);

                                           /* Test signal stack frame unw
    establish_sigquit_siginfo_handler (sigquit_siginfo_handler);
STOP    kill (getpid(), SIGQUIT);
⇨    establish_sigquit_signal_handler (sigquit_signal_handler);
    kill (getpid(), SIGQUIT);
    signal (SIGQUIT, sigquit_signal_handler);
    kill (getpid(), SIGQUIT);

  #ifdef USE_L2_ATOMIC
                                           /* Allocate and init the L2 At
    init_l2_atomic();
  #endif

  #ifdef USE_BGPM
```

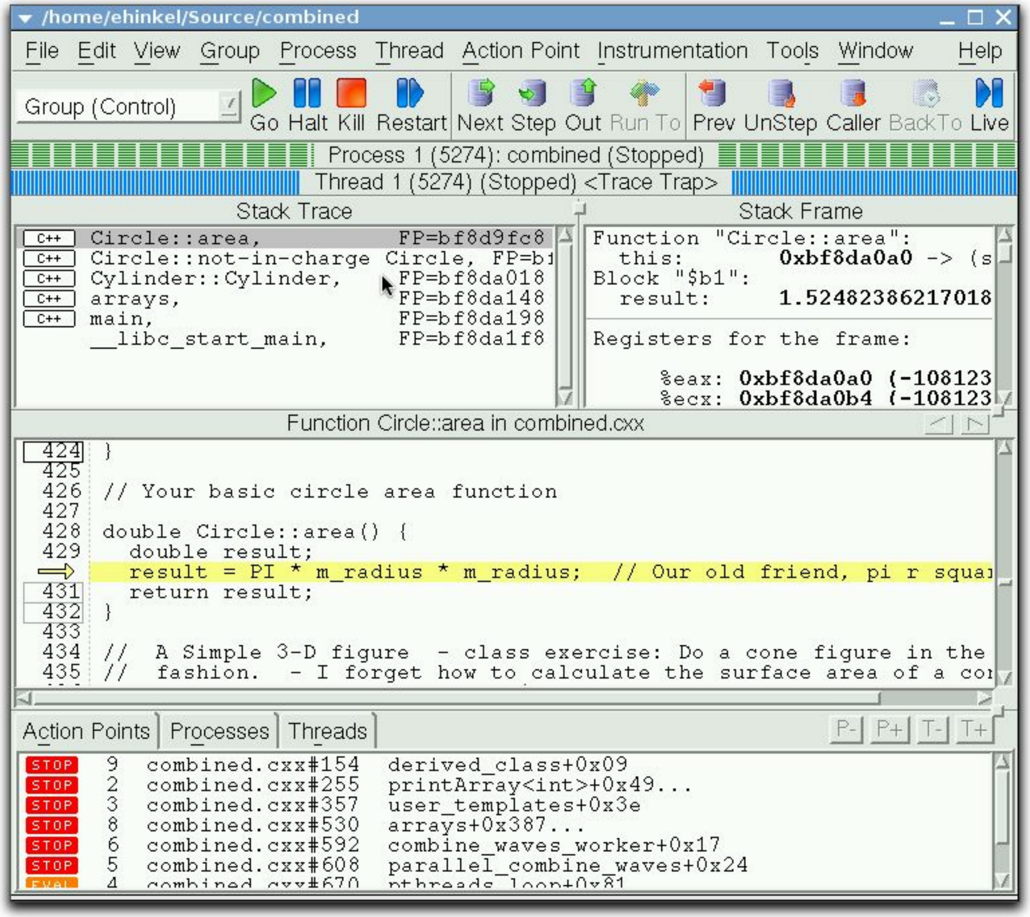
| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| p1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 |
| 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 |
| 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 |
| 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 |

# Techniques for Debugging Challenges

# 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
  - Multi-process Debugging
- **Integrated Memory Debugging**
- **Reverse Debugging available**
- **Supports Multiple Usage Models**
  - Powerful and Easy GUI – Highly Graphical
  - CLI for Scripting
  - Long Distance Remote Debugging
  - Unattended Batch Debugging



Process 1 (5274): combined (Stopped)

Thread 1 (5274) (Stopped) <Trace Trap>

```
C++  Circle::area,                FP=bf8d9fc8
C++  Circle::not-in-charge Circle, FP=bf
C++  Cylinder::Cylinder,          FP=bf8da018
C++  arrays,                      FP=bf8da148
C++  main,                        FP=bf8da198
     __libc_start_main,           FP=bf8da1f8
```

```
Function "Circle::area":
  this:          0xbf8da0a0 -> (s
Block "$b1":
  result:        1.52482386217018

Registers for the frame:

      %eax:  0xbf8da0a0 (-108123
      %ecx:  0xbf8da0b4 (-108123
```

Function Circle::area in combined.cxx

```
}

// Your basic circle area function

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

//  A Simple 3-D figure  - class exercise: Do a cone figure in the
//  fashion.  - I forget how to calculate the surface area of a co
```

```
STOP  9  combined.cxx#154  derived_class+0x09
STOP  2  combined.cxx#255  printArray<int>+0x49...
STOP  3  combined.cxx#357  user_templates+0x3e
STOP  8  combined.cxx#530  arrays+0x387...
STOP  6  combined.cxx#592  combine_waves_worker+0x17
STOP  5  combined.cxx#608  parallel_combine_waves+0x24
EVAL  4  combined.cxx#670  pthreads_loop+0x81
```

# Debugging Complex Codes

- Mechanize
- Minimize
- Visualize
- … and Don’t Forget the Memory

# Mechanize

## Extended Automation Capabilities

# Automated Debugging

## TVscript

- Non-Interactive Batch Debugging –
  - Work in the “main” batch queue
  - Don’t have to baby-sit job waiting on it to run
  - Use scripting to perform checks that would be tedious to do by hand
  - Verification through automated processes (nightly build and test)

## TTF and C++View

- Automatic Transformation of Data –
  - Simplify interactive (and scripted) debugging
  - Perform validation/sanity checking of large datasets
  - Comparative debugging
  - Allows you to focus on troubleshooting your program

# Non-Interactive Batch Debugging with TVscript

- Run multiple debugging sessions without the need for recompiling, unlike with printf
- TVscript syntax:

  tvscript [ options ] [ filename ] [ -a program_args ]
- More complex actions-to-events are possible, utilizing TCL within a CLI file
- TVscript lets you define what events to act on, and what actions to take

## TVscript uses a simple, Event/Action interface

### Typical Events

- Action_point
- Any_memory_event
- Guard_corruption error

### Typical Actions

- Display_backtrace [-level *level-num*]
- List_leaks
- Save_memory
- Print [-slice {*slice_exp*] {*variable* | *exp*}

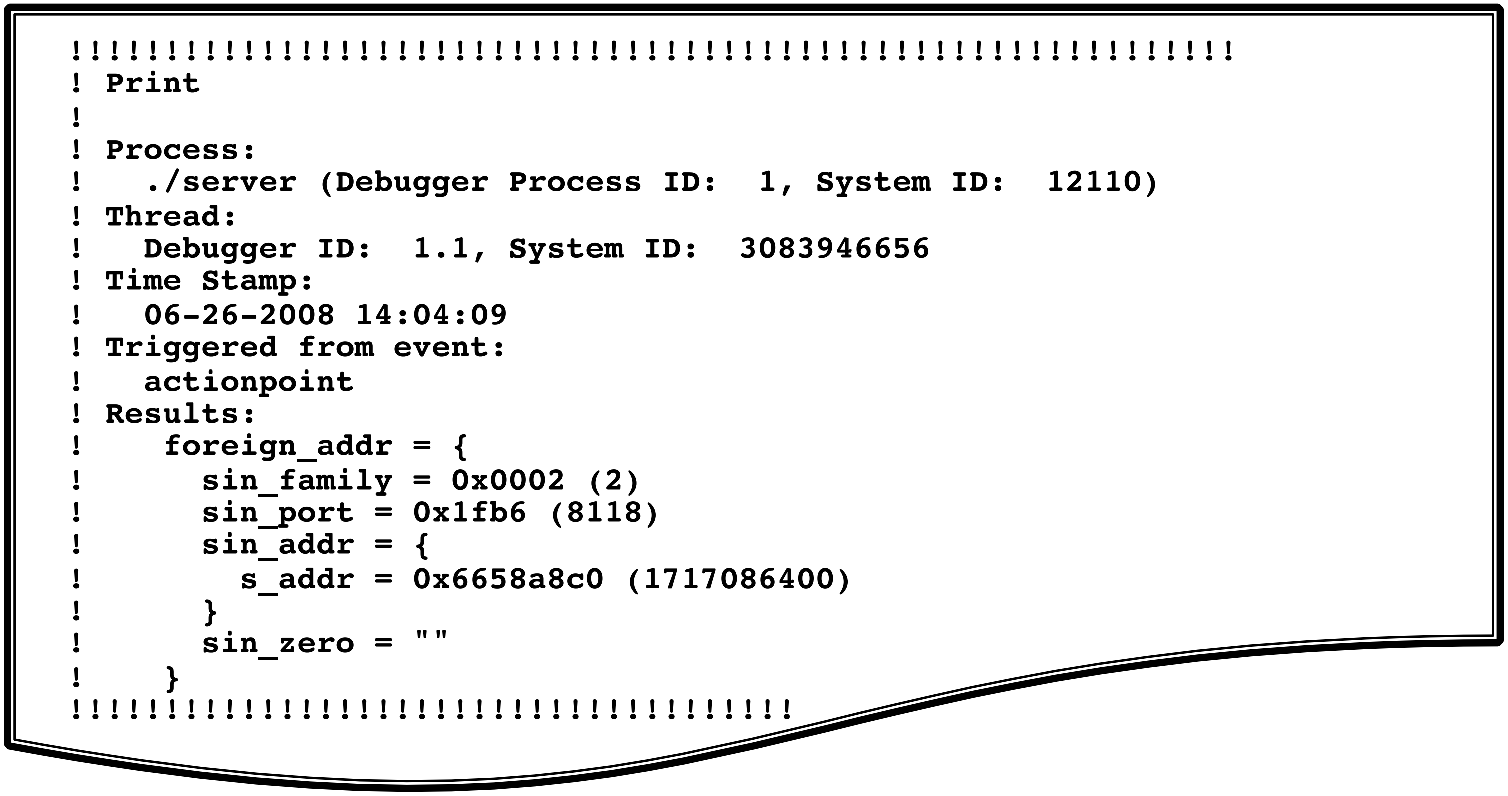
# Unattended Debugging with Tvscript

**Example**

The following tells tvscript to report the contents of the *foreign_addr* structure each time the program gets to line 85

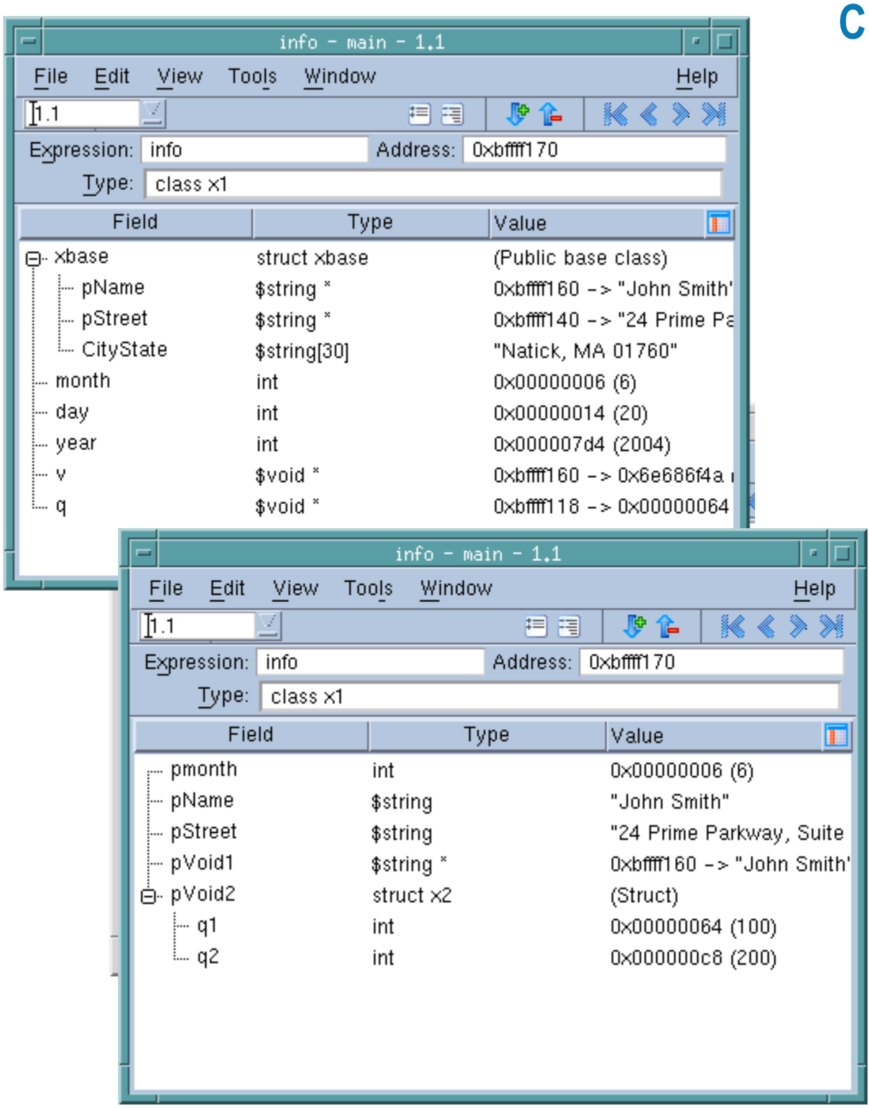
**-create_actionpoint "#85=>print foreign_addr"**

Typical output sample with tvscript:



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

# Creating Type Transformations



Expression: info Address: 0xbffff170

Type: class x1

| Field | Type | Value |
|---|---|---|
| ⊟ xbase | struct xbase | (Public base class) |
| pName | $string * | 0xbffff160 -> "John Smith' |
| pStreet | $string * | 0xbffff140 -> "24 Prime Pa |
| CityState | $string[30] | "Natick, MA 01760" |
| month | int | 0x00000006 (6) |
| day | int | 0x00000014 (20) |
| year | int | 0x000007d4 (2004) |
| v | $void * | 0xbffff160 -> 0x6e686f4a |
| q | $void * | 0xbffff118 -> 0x00000064 |

Expression: info Address: 0xbffff170

Type: class x1

| Field | Type | Value |
|---|---|---|
| pmonth | int | 0x00000006 (6) |
| pName | $string | "John Smith" |
| pStreet | $string | "24 Prime Parkway, Suite |
| pVoid1 | $string * | 0xbffff160 -> "John Smith' |
| ⊟ pVoid2 | struct x2 | (Struct) |
| q1 | int | 0x00000064 (100) |
| q2 | int | 0x000000c8 (200) |

## Customize your own Transformations

**In $HOME/.tvdrc:**

```
::TV::TTF::RTF::build_struct_transform {
     name     {^class x1$}
     members {
          { pmonth    { month } }
          { pName      { xbase upcast { * pName    } } }
          { pStreet   { xbase upcast { * pStreet } } }
          { pVoid1    { "$string *"    cast v       } }
          { pVoid2    { * { "class x2 *"    cast q } } }
     }
}
```

**Meta Language:**

**{member}**

**{* expr}**

**{expr . Expr}**

**{expr -> expr}**

**{datatype case expr}**

**{baseclass upcast expr}**

# C++View

- C++View is an easy way to customize TotalView's display of object data.
- How does it work?
  - User writes short display functions within their program
  - TotalView uses these functions to simplify the display of data when the user explores their data within that program
  - C++View transforms are easy to define
  - Great for collaborative codes (transforms can be distributed with the program)
- Benefit: Easier for scientists and developers to work with complex applications

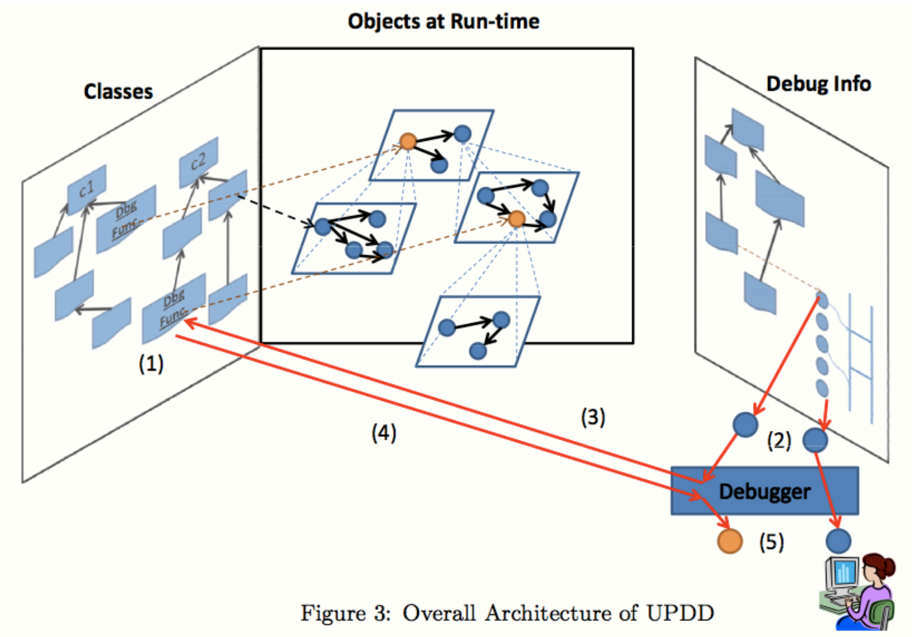
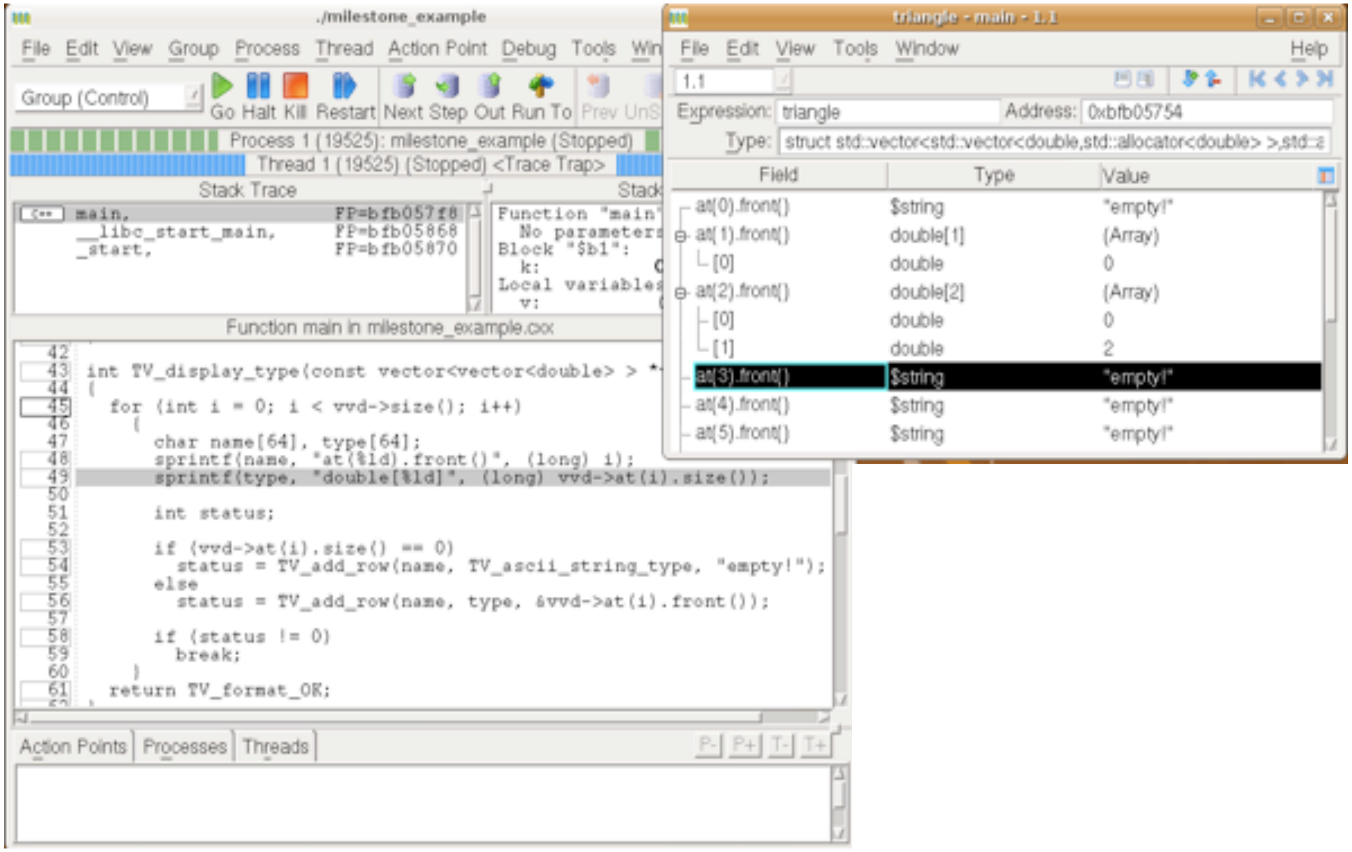


Figure 3: Overall Architecture of UPDD

Developers can now write display and analysis functions for their C++ classes that are invoked whenever an object is inspected interactively in the debugger.

# C++View

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


./milestone_example
File Edit View Group Process Thread Action Point Debug Tools Win
Group (Control)
Go Halt Kill Restart Next Step Out Run To Prev UnS
Process 1 (19525): milestone_example (Stopped)
Thread 1 (19525) (Stopped) <Trace Trap>
Stack Trace
Stack
Function main in milestone_example.cxx
Action Points Processes Threads
P- P+ T- T+
triangle - main - 1.1
File Edit View Tools Window Help
1.1

```
main,                  FP=bfb057f8
__libc_start_main,     FP=bfb05868
_start,                FP=bfb05870
```

```
Function "main"
  No parameters
Block "$b1":
  k:
Local variables
  v:
```

```
int TV_display_type(const vector<vector<double> > *
{
  for (int i = 0; i < vvd->size(); i++)
    {
      char name[64], type[64];
      sprintf(name, "at(%ld).front()", (long) i);
      sprintf(type, "double[%ld]", (long) vvd->at(i).size());

      int status;

      if (vvd->at(i).size() == 0)
        status = TV_add_row(name, TV_ascii_string_type, "empty!");
      else
        status = TV_add_row(name, type, &vvd->at(i).front());

      if (status != 0)
        break;
    }
  return TV_format_OK;
```

Expression: triangle Address: 0xbfb05754

Type: struct std::vector<std::vector<double,std::allocator<double> >,std::a

| Field | Type | Value |
|---|---|---|
| at(0).front() | $string | "empty!" |
| at(1).front() | double[1] | (Array) |
| [0] | double | 0 |
| at(2).front() | double[2] | (Array) |
| [0] | double | 0 |
| [1] | double | 2 |
| at(3).front() | $string | "empty!" |
| at(4).front() | $string | "empty!" |
| at(5).front() | $string | "empty!" |

# Minimize

## Reduce the Scope of Effort

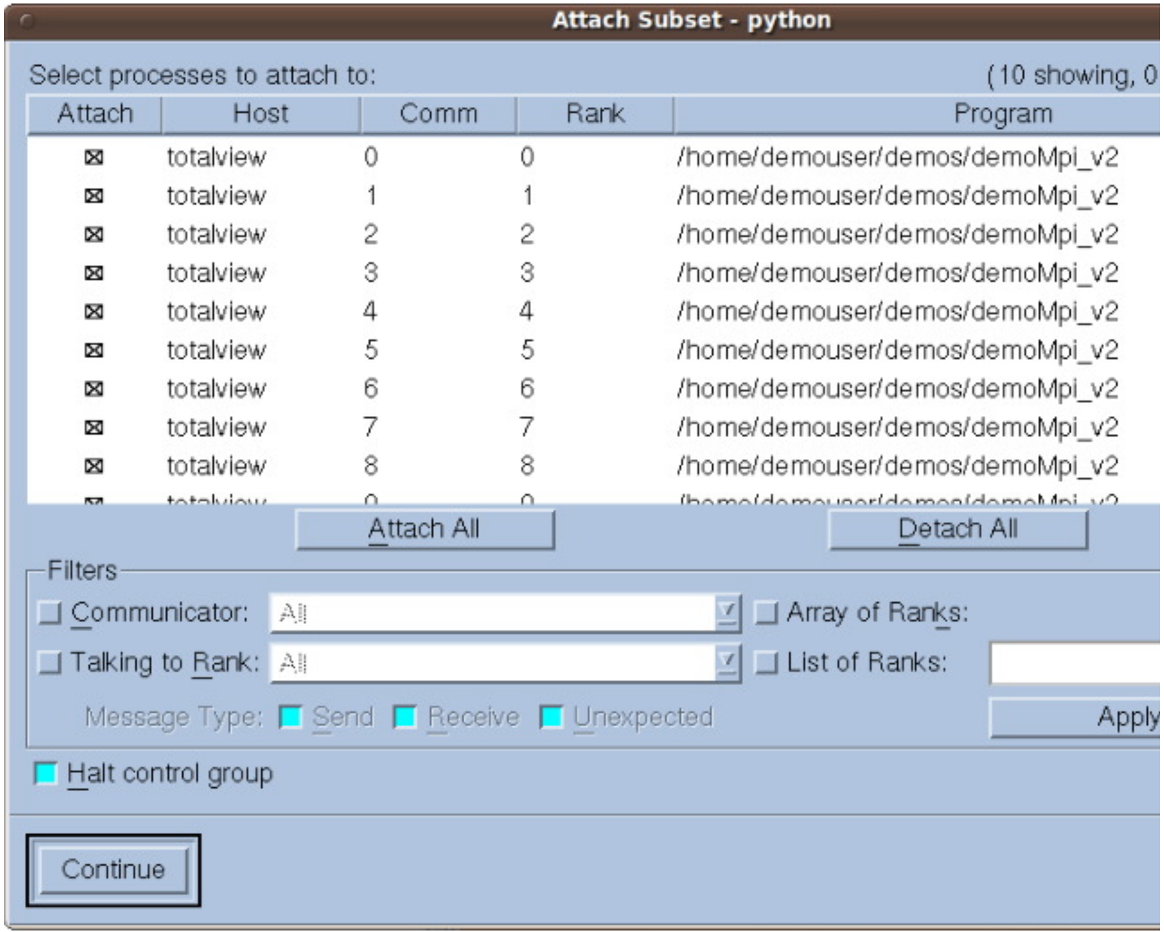
# Subset Debugging

## With TotalView

# Subset Attach

## You need not 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
- It also reduces license size requirements



Attach Subset - python

Select processes to attach to: (10 showing, 0

| Attach | Host | Comm | Rank | Program |
|---|---|---|---|---|
| ☒ | totalview | 0 | 0 | /home/demouser/demos/demoMpi_v2 |
| ☒ | totalview | 1 | 1 | /home/demouser/demos/demoMpi_v2 |
| ☒ | totalview | 2 | 2 | /home/demouser/demos/demoMpi_v2 |
| ☒ | totalview | 3 | 3 | /home/demouser/demos/demoMpi_v2 |
| ☒ | totalview | 4 | 4 | /home/demouser/demos/demoMpi_v2 |
| ☒ | totalview | 5 | 5 | /home/demouser/demos/demoMpi_v2 |
| ☒ | totalview | 6 | 6 | /home/demouser/demos/demoMpi_v2 |
| ☒ | totalview | 7 | 7 | /home/demouser/demos/demoMpi_v2 |
| ☒ | totalview | 8 | 8 | /home/demouser/demos/demoMpi_v2 |
| ☒ | totalview | 9 | 9 | /home/demouser/demos/demoMpi_v2 |

Filters

☐ Communicator: All ☐ Array of Ranks:

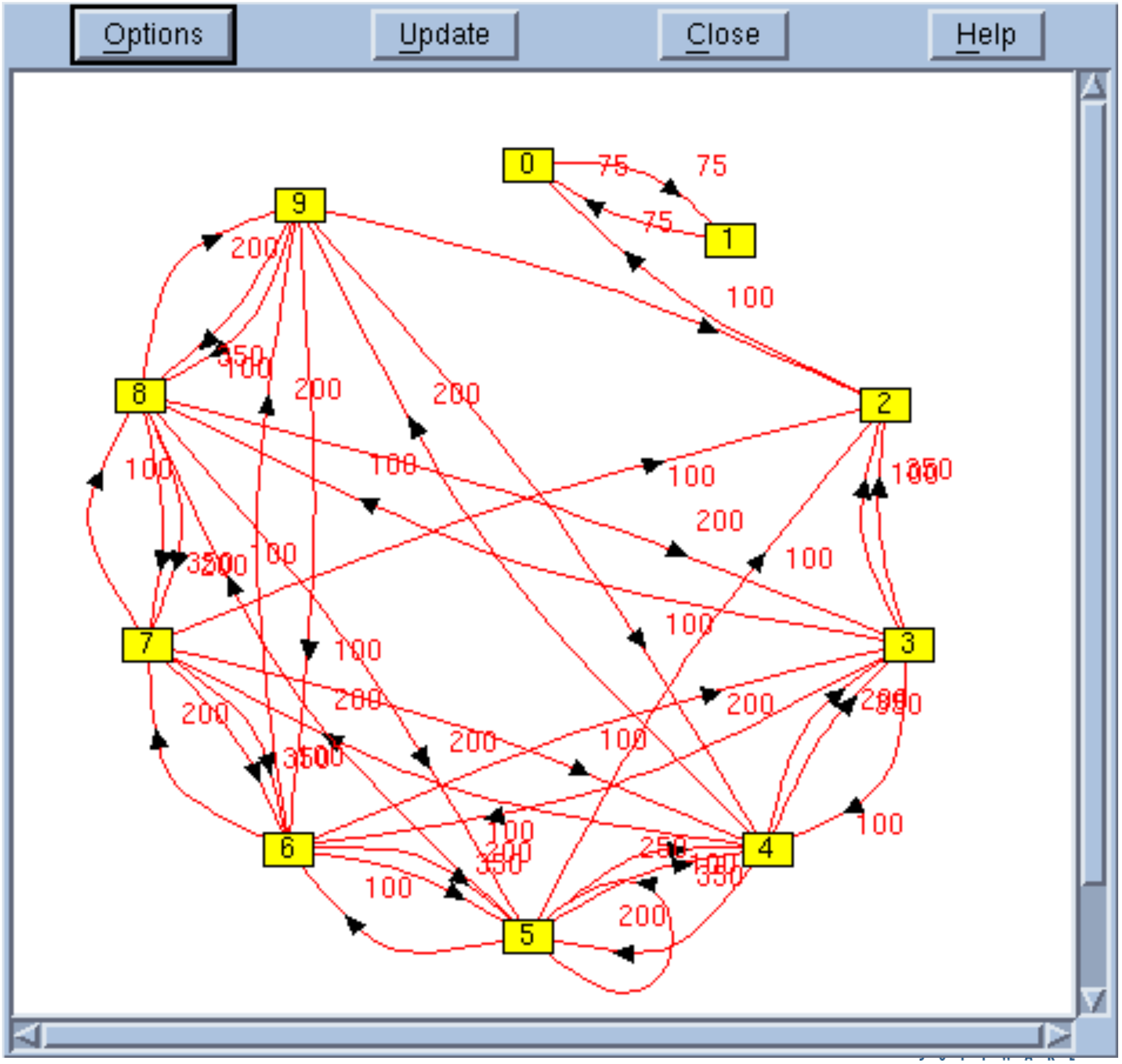
☐ Talking to Rank: All ☐ List of Ranks:

Message Type: ☑ Send ☑ Receive ☑ Unexpected

☑ Halt control group

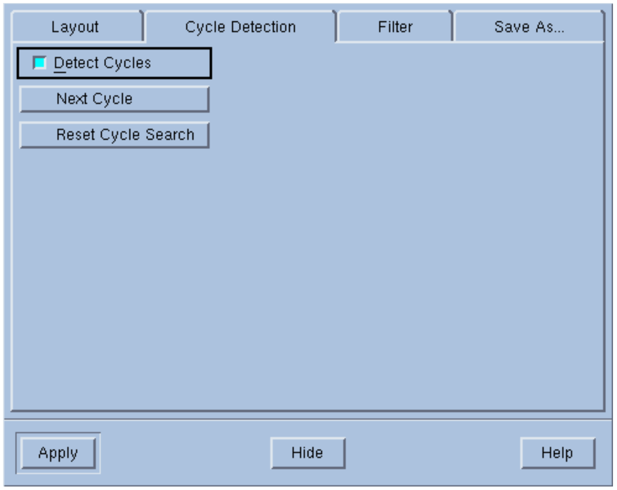
# Message Queue Graph

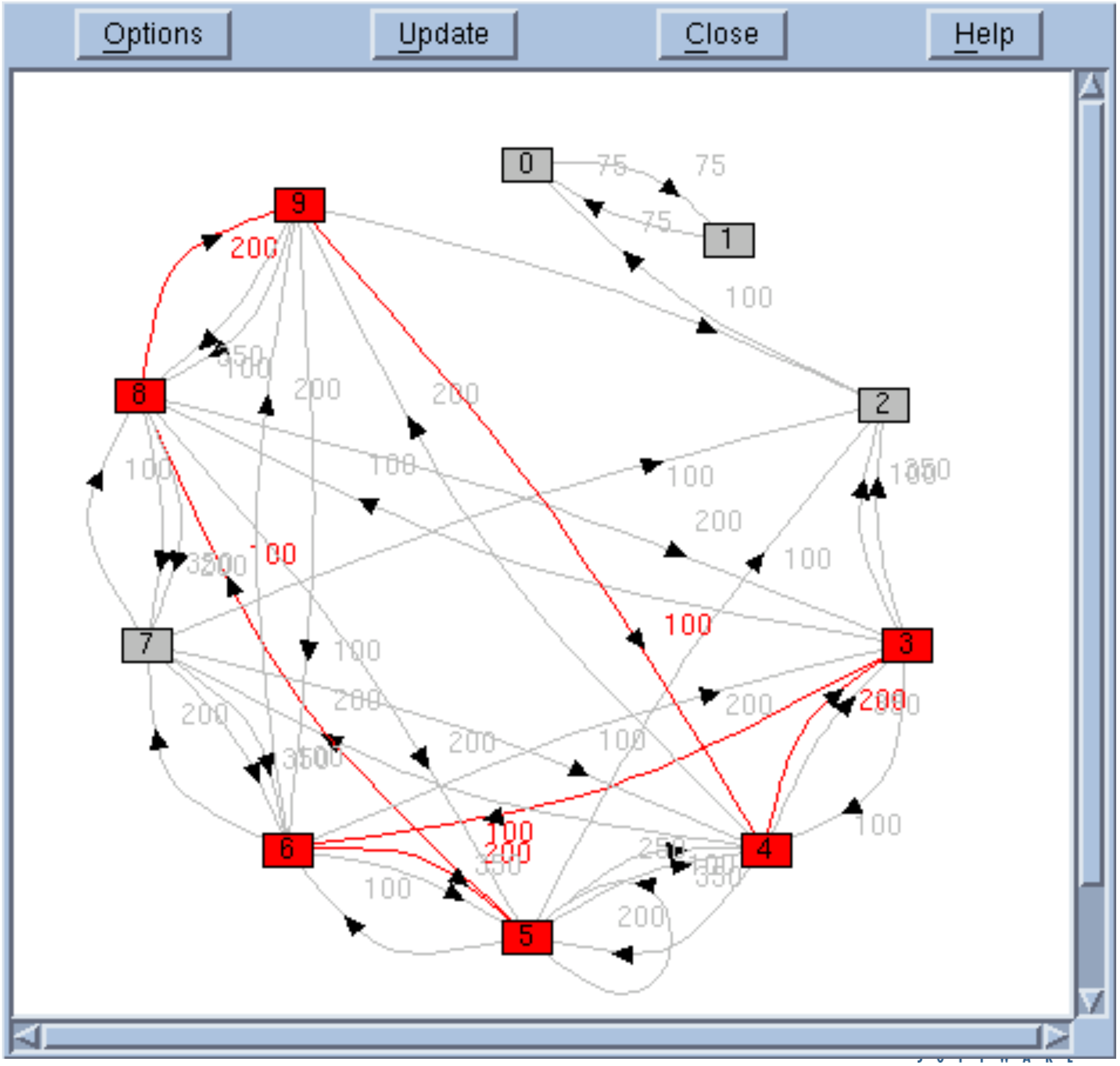
- Hangs & Deadlocks
- Pending Messages
  - Receives
  - Sends
  - Unexpected
- Inspect
  - Individual entries
- Patterns



# Message Queue Debugging

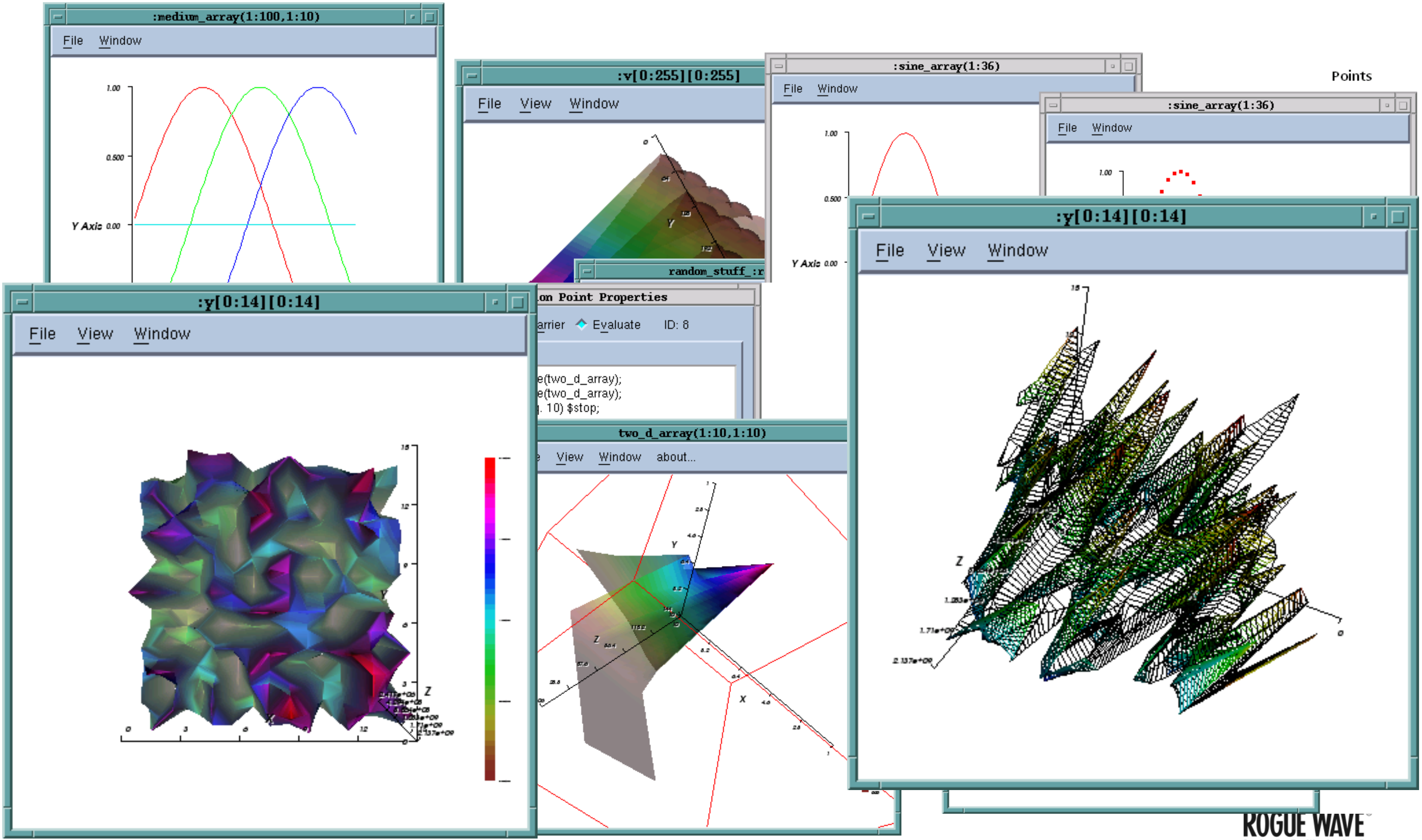
- Filtering
  - Tags
  - MPI Communicators
- Cycle detection
  - Find deadlocks





# Visualize

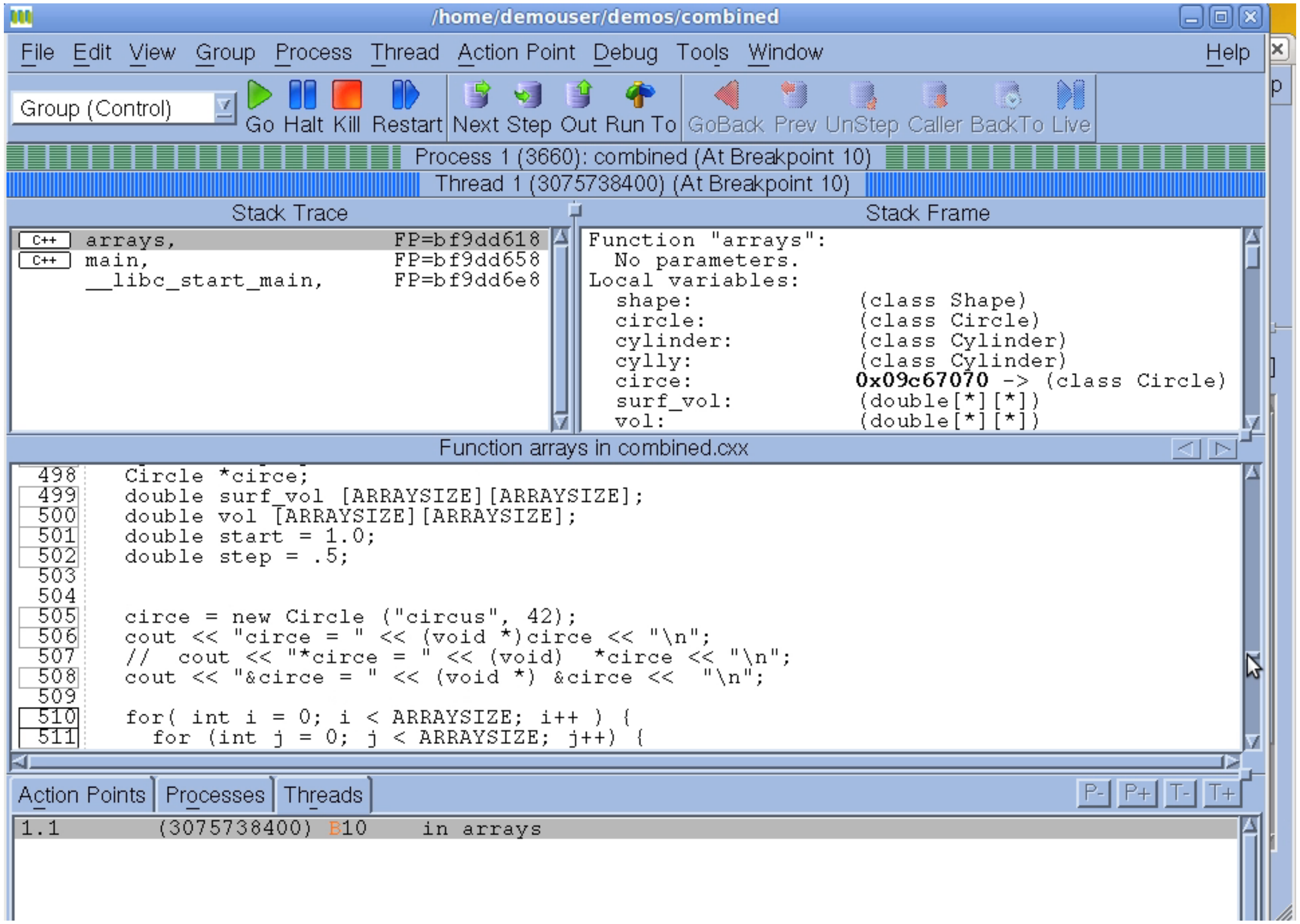
# Visualization



```
e(two_d_array);
e(two_d_array);
. 10) $stop;
```

# Visualization

**Get the big picture – Observe anomalies – Utilize Pattern recognition – Save time!**



Process 1 (3660): combined (At Breakpoint 10)

Thread 1 (3075738400) (At Breakpoint 10)

```
C++  arrays,              FP=bf9dd618
C++  main,                FP=bf9dd658
     __libc_start_main,   FP=bf9dd6e8
```

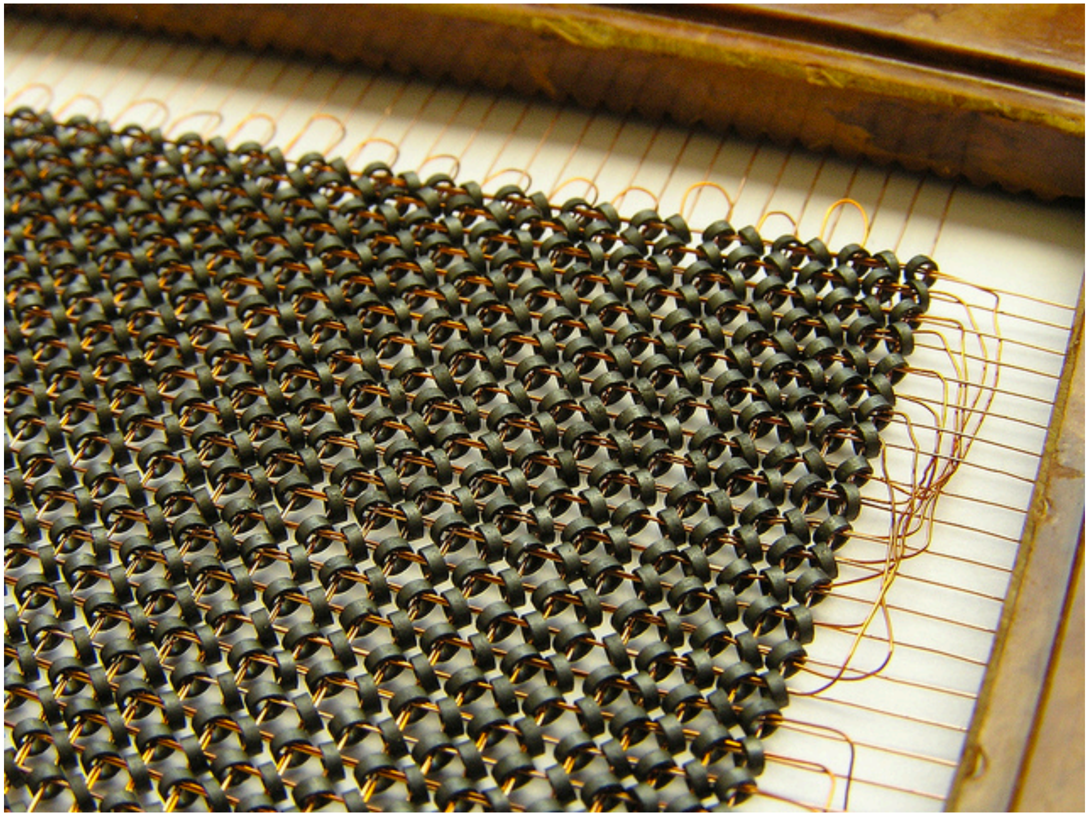
```
Function "arrays":
  No parameters.
Local variables:
  shape:                 (class Shape)
  circle:                (class Circle)
  cylinder:              (class Cylinder)
  cylly:                 (class Cylinder)
  circe:                0x09c67070 -> (class Circle)
  surf_vol:              (double[*][*])
  vol:                   (double[*][*])
```

Function arrays in combined.cxx

```
    Circle *circe;
    double surf_vol [ARRAYSIZE][ARRAYSIZE];
    double vol [ARRAYSIZE][ARRAYSIZE];
    double start = 1.0;
    double step = .5;


    circe = new Circle ("circus", 42);
    cout << "circe = " << (void *)circe << "\n";
    //  cout << "*circe = " << (void)  *circe << "\n";
    cout << "&circe = " << (void *) &circe <<  "\n";

    for( int i = 0; i < ARRAYSIZE; i++ ) {
      for (int j = 0; j < ARRAYSIZE; j++) {
```

```
1.1        (3075738400) B10     in arrays
```

# … And Don’t Forget the Memory!



# MemoryScape

## Memory bugs often go undetected until the worst possible time

- Symptoms often surface long after the actual damage is done
- Some only surface after hours or even days of operation
- In many cases, the programs affected are “innocent bystanders”

## MemoryScape: Fully Integrated in TotalView

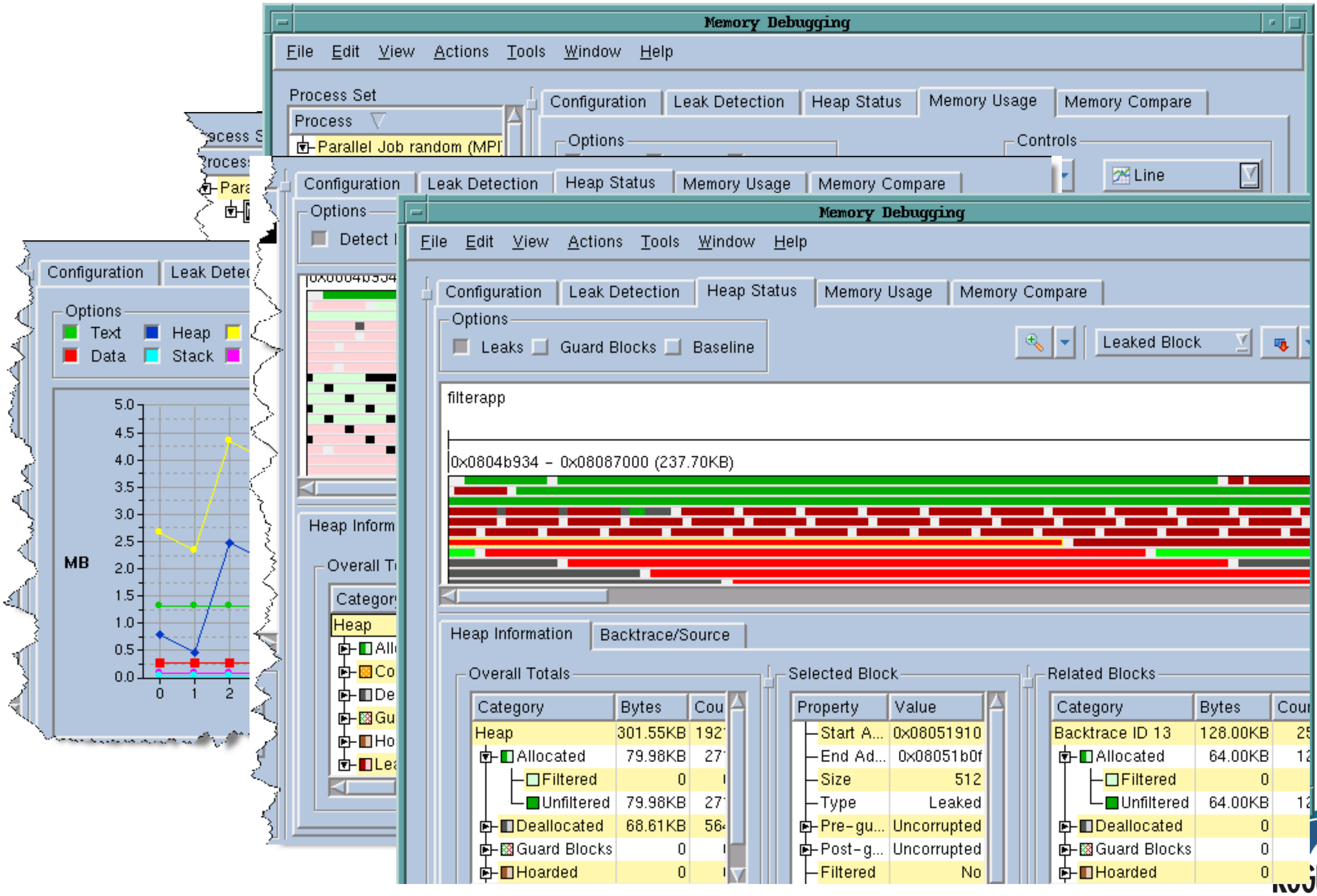
### No Source Code or Binary Instrumentation

- Use it with your existing builds
- Programs run nearly full speed
- Low performance overhead
- Low memory overhead • Efficient memory usage

### MemoryScape Feature Highlights

- Automatic allocation problem detection
- Heap Graphical View
- Leak detection
- Block painting
- Dangling pointer detection
- Deallocation/reallocation notification
- Memory Corruption Detection - Guard Blocks
- Memory Hoarding
- Memory Comparisons between processes
- Collaboration features

# MemoryScape


Memory Debugging
File Edit View Actions Tools Window Help
Process Set
Process
Parallel Job random (MPI
Configuration Leak Detection Heap Status Memory Usage Memory Compare
Options
Controls
Line
Detect
Options
Text Heap
Data Stack
MB
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
Heap Inform
Overall T
Categor
Heap
All
Co
De
Gu
Ho
Lea
Memory Debugging
File Edit View Actions Tools Window Help
Configuration Leak Detection Heap Status Memory Usage Memory Compare
Options
Leaks Guard Blocks Baseline
Leaked Block
Heap Information Backtrace/Source
Overall Totals
Selected Block
Related Blocks

filterapp

0x0804b934 – 0x08087000 (237.70KB)

Overall Totals

| Category | Bytes | Cou |
|---|---|---|
| Heap | 301.55KB | 192 |
| Allocated | 79.98KB | 27 |
| Filtered | 0 | |
| Unfiltered | 79.98KB | 27 |
| Deallocated | 68.61KB | 56 |
| Guard Blocks | 0 | |
| Hoarded | 0 | |

Selected Block

| Property | Value |
|---|---|
| Start A... | 0x08051910 |
| End Ad... | 0x08051b0f |
| Size | 512 |
| Type | Leaked |
| Pre-gu... | Uncorrupted |
| Post-g... | Uncorrupted |
| Filtered | No |

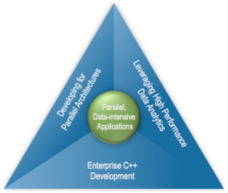
Related Blocks

| Category | Bytes | Cour |
|---|---|---|
| Backtrace ID 13 | 128.00KB | 25 |
| Allocated | 64.00KB | 12 |
| Filtered | 0 | |
| Unfiltered | 64.00KB | 12 |
| Deallocated | 0 | |
| Guard Blocks | 0 | |
| Hoarded | 0 | |

# What's Coming

- <u>Increased Scalability</u>
  - Leveraging TotalView's Architecture
  - Efficient Use of Cluster Resources
    - Extremely light weight debug agents; Minimal memory footprint
    - More space on the compute nodes for user application code
  - Tree-Based Overlay Network
    - Broadcast of Operations; Aggregation of Events and Data
- <u>Advanced User Interface</u>
    - New GUI Framework
    - Changes focused on extreme scale debugging
- <u>CUDA 4.1 now; 4.2 and 5.0  this year</u>
- Replay Enhancements
  - Record on Demand (in Beta)
  - Replay Debug from Core File
- OpenACC Support
- <u>Intel MIC Support</u>
  - Come see a demo at ISC '12

# Developing for Parallel Architectures


Developing for Parallel Architectures
Leveraging High Performance Data Analytics
Parallel, Data-intensive Applications
Enterprise C++ Development

## TotalView®

- Code debugging
  - Highly scalable interactive GUI debugger
    - Easy to use -- without sacrificing detail that users need to debug
    - Used from workstations to the largest supercomputers
  - Powerful features for debugging multi-threaded, multi-process, and MPI parallel programs
  - Compatible with wide variety of compilers across several platforms and operating systems
- Memory Debugging
  - Parallel memory analysis and error detection
    - Intuitive for both intensive and infrequent users
  - Easily integrated into the validation process
- Reverse Debugging
  - Parallel record and deterministic replay within TotalView
    - Users can run their program “backwards” to find bugs
  - Allows straightforward resolution of otherwise stochastic bugs
- GPU CUDA Debugging
  - Full Hybrid Architecture Support
  - Asynchronous Warp Control
  - Multi-Device and MPI Support

Thanks!

ROGUE WAVE®
SOFTWARE

Developing parallel, data-intensive applications is hard.

**We make it easier.**

**www.roguewave.com**