THE SUPERCOMPUTER COMPANY

Cray Debugger Support Tools

Luiz DeRose
Programming Environments Director
Cray Inc.

HP2C

CSCS
Swiss National Supercomputing Centre

Debuggers on Cray Systems

- Systems with hundreds of thousands of threads of execution need a new debugging paradigm
  - Innovative techniques for productivity and scalability
    - ➢ Scalable Solutions based on MRNet from University of Wisconsin
      STAT - Stack Trace Analysis Tool
      - » Scalable generation of a single, merged, stack backtrace tree
        - ☞ running at 216K back-end processes

      ATP - Abnormal Termination Processing
      - » Scalable analysis of a sick application, delivering a STAT tree and a minimal, comprehensive, core file set.
    - ➢ Fast Track Debugging
      - o Debugging optimized applications
      - o Added to Allinea's DDT 2.6 (June 2010)
    - ➢ Comparative debugging
      - o A **data-centric paradigm** instead of the traditional control-centric paradigm
      - o Collaboration with Monash University and University of Wisconsin for scalability
  - Support for traditional debugging mechanism
    - ➢ TotalView, DDT, and gdb

MRNet - Multicast Reduction Network

- Tree based software overlay network

- Provides efficient multicast and reduction communications for parallel and distributed tools

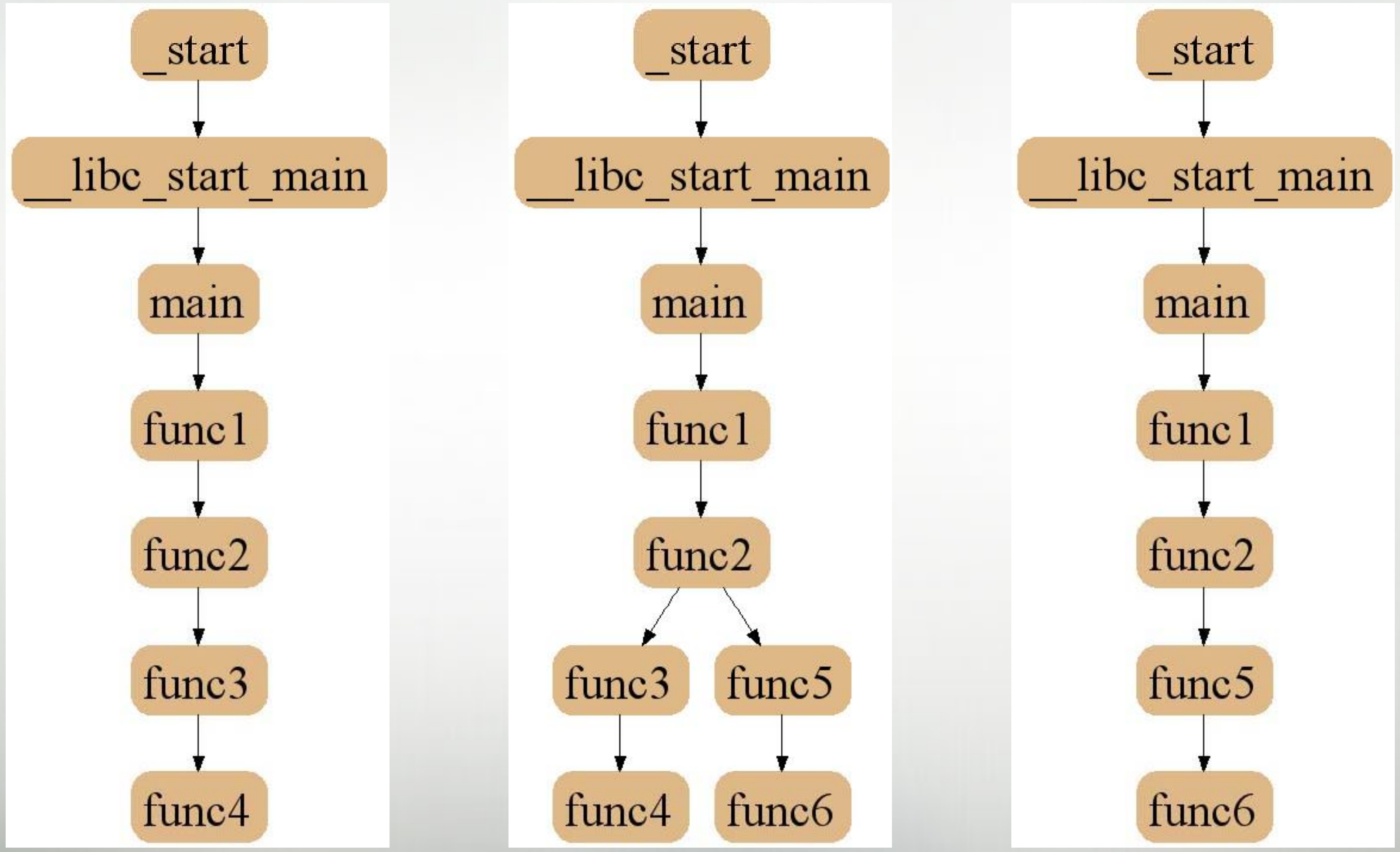
- Uses a tree of processes between the tool's front-end and back-ends to improve group communication performance
  - Internal processes are used to distribute important tool activities
    - Reduce data analysis time
    - Keep tool front-end loads manageable

Stack Trace Analysis Tool (STAT)

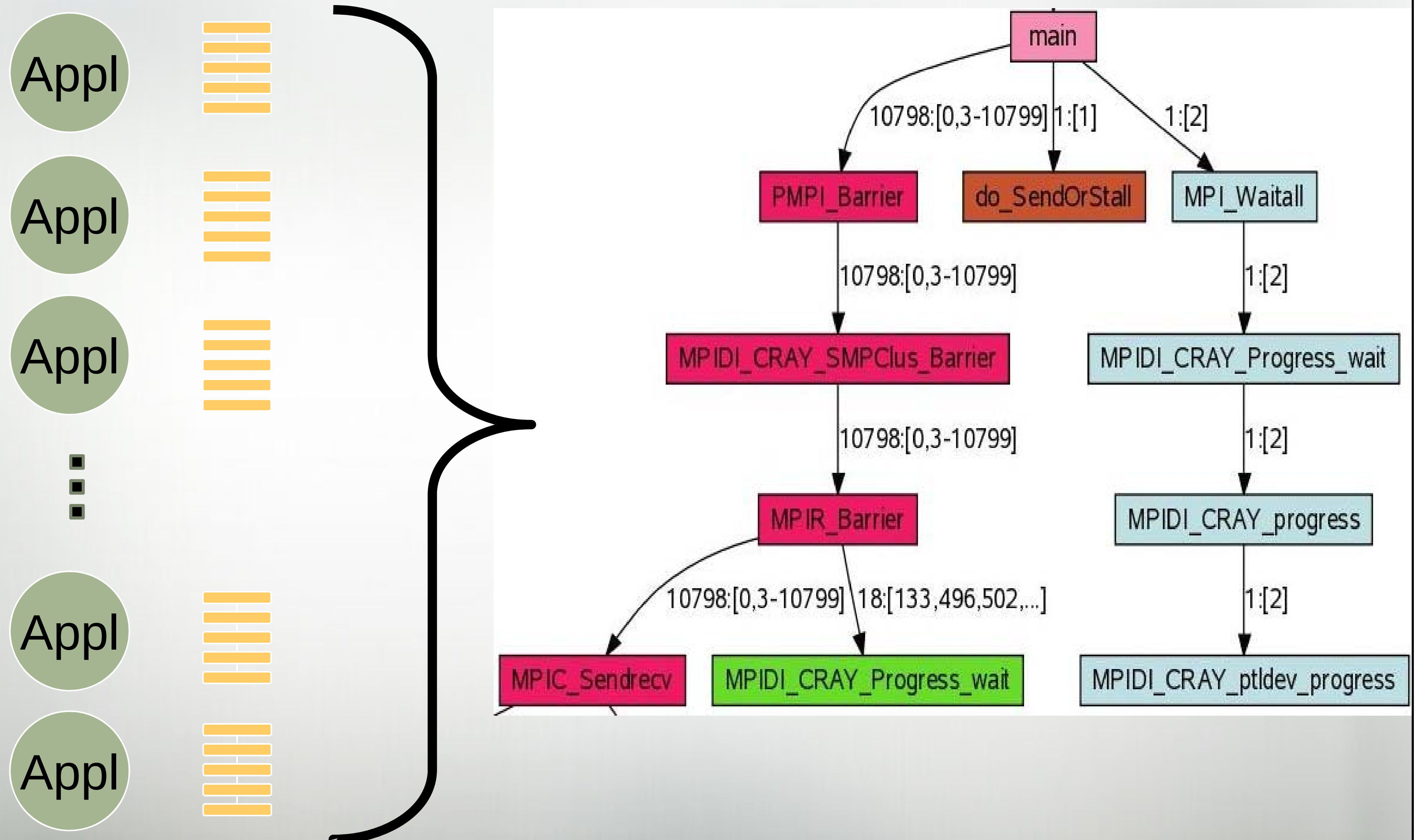
- Stack trace sampling and analysis for large scale applications
  - Sample application stack traces
  - Scalable generation of a single, merged, stack backtrace tree
    - A comprehensible view of the entire application
    - Discover equivalent process behavior
      - Group similar processes
      - Reduce number of tasks to debug
    - 128K processes analyzed in 2.7 seconds, using MRNet

- Merge/analyze traces:
  - Facilitate scalable analysis/data presentation
  - Multiple traces over space or time
  - Create call graph prefix tree
    - Compressed representation
    - Scalable visualization
    - Scalable analysis

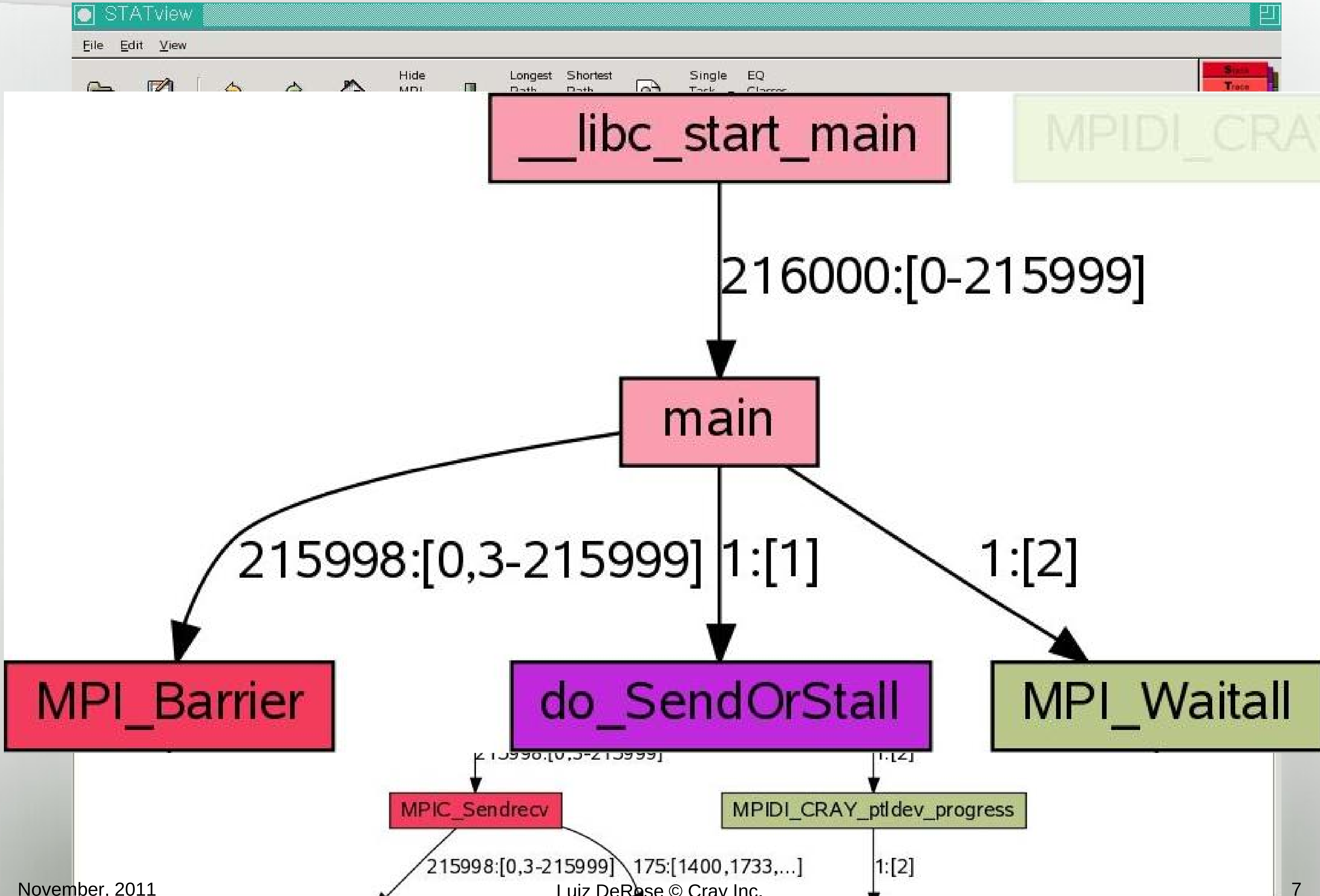
Stack Trace Merge Example



2D-Trace/Space Analysis



Merged Stack for Cray XT



STAT 1.1.3

- module load stat
  - Not loaded by default on Todi
- **man STAT**
- STAT <pid_of_aprun>
  - Creates STAT_results/<app_name>/<merged_bt_file>
- Scaling limited by number file descriptors
- STAT 1.2.0 planned for November 2011

ATP: The Problem Being Solved

- When a large scale parallel application dies, one, many, or all processes might trap!
  - It is next to impossible to examine all the core files and backtraces
    - No one wants that many stack backtraces
    - No one wants that many core files
      - They are too slow and too big
        - Sufficient storage for all core files is a problem
    - They are too much to comprehend
  - A single core file or stack backtrace is usually not enough to debug either!
    - A single backtrace produced might not be from the process that first failed

- Requirements:
  - Minimum jitter
  - Scalability
  - Robustness
  - Small footprint
  - Limited core file dumping

- **ATP 1.3 released in August 2011**

ATP Description

- System of light weight back-end monitor processes on compute nodes
  - Coupled together with **MRNet**
  - Automatically launched by aprun in parallel with application launch
    - Enabled/disabled via ATP_ENABLED environment variable
- Leap into action on any application process trapping
  - stderr **backtrace** of first process to trap
    - dumps core file set (if limit/ulimit allows)
  - Uses **StackwalkerAPI** to collect individual stack backtraces, even for optimized code
- STAT like analysis **provides merged stack backtrace tree**
  - Leaf nodes of tree define a modest set of processes to core dump
    - or, a set of processes to attach to with a debugger

Abnormal Termination Processing

- ATP produces a single merged stack trace
  - or a reduced set of core files

- The benefits:
  - Minimal impact on application run
    - Can be used with production runs
  - Automated, transparent collection of data
  - Ability to hold failing application for close inspection
    - This is site dependent
  - Easy to navigate the merged stack trace
  - Manageable set of core files
  - Reduced amount of data saved
    - Especially true in the core file situation

ATP: How It Works

- ATP is launched via an ALPS enhancement which includes the fork/exec of a login side ATP front-end daemon
  - The ATP front-end uses MRNet and the ALPS tool helper library to launch ATP back-end servers on all compute nodes associated with the application
- ATP signal handler runs within an application to catch fatal errors
  - It handles the following signals:
    - SIGQUIT, SIGILL, SIGTRAP, SIGABRT, SIGFPE, SIGBUS, SIGSEGV, SIGSYS, SIGXCPU, SIGXFSZ
    - Setting the environment variables MPICH_ABORT_ON_ERROR and SHMEM_ABORT_ON_ERROR will cause a signal to be thrown and captured for MPI and SHMEM fatal errors
- ATP daemon running on the compute node captures signals, starts termination processing
  - Rest of the application processes are notified
  - Generates a stacktrace
  - Creates a single merged stack trace file
- The stack trace file is viewed with the STATview tool

Fast Track Debugging: The Problem Being Solved

- How to debug parallel optimized codes

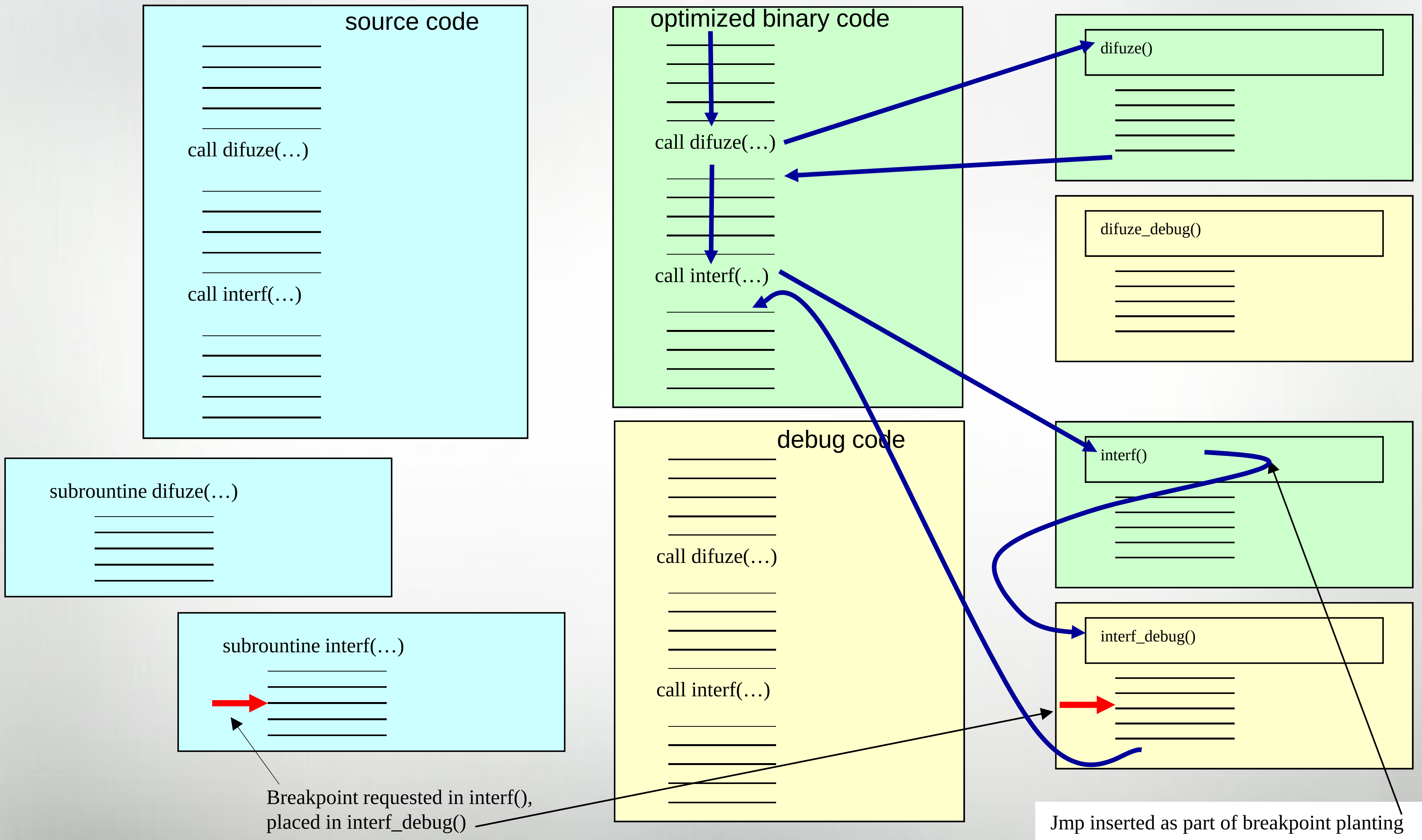
- Debug flags eliminate optimizations
  - Today's machines really need optimizations
  - Slows down execution
  - Problem might disappear

- Fast Track Debugging addresses this problem

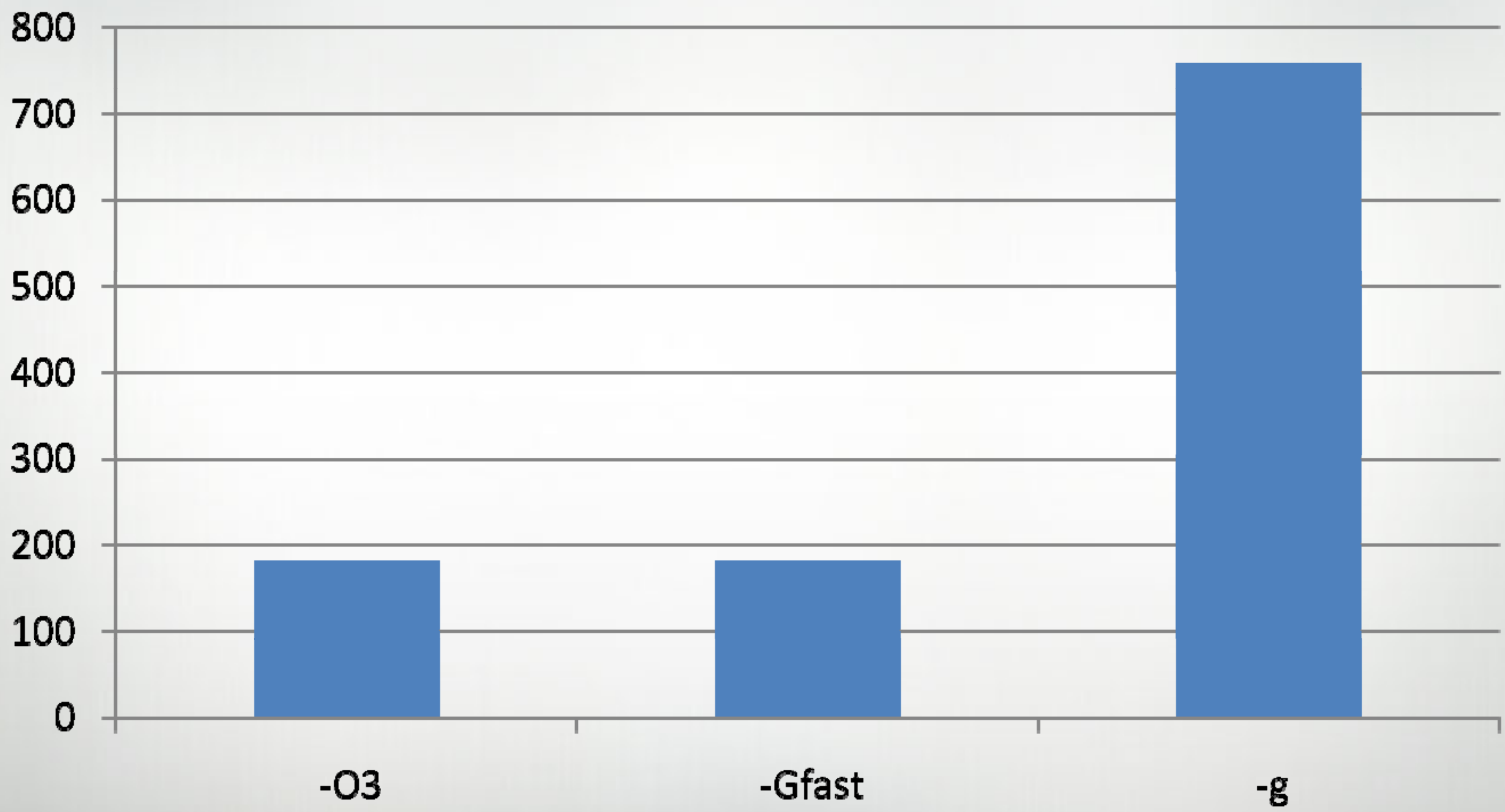
How to do "Fast Track Debugging"?

- Compile such that both debug and non-debug (optimized) versions of each routine are created
  - Debug and non-debug versions of each subroutine appear in the executable
- Linkage such that optimized versions are used by default
- User sets breakpoints or other debug constructs
  - Debugger overrides default linkage when setting breakpoints and stepping into functions
  - Routines automatically presented using the debug version of the routine
  - Rest of program executes using optimized versions of the routines

A Closer Look at How FTD Works



Tera TF Execution Time



Fast Track Debugger Status

- Support available in the Cray Compilation Environment (CCE)

- Prototype in gdb
  - Exercised through lgdb

- Added to Allinea's DDT 2.6 (June 2010)

- Issues / Cost:
  - Compiles are slower
  - Executable uses more disk space
  - Libraries probably don't have a debug version
  - Inlining turned off
    - 1.7% average slow down of all SPEC2007MPI tests
    - Range of slight speedup to 19.5% slow down
  - Uses more memory
    - 4% larger at start up
    - 0.0001% larger after computation