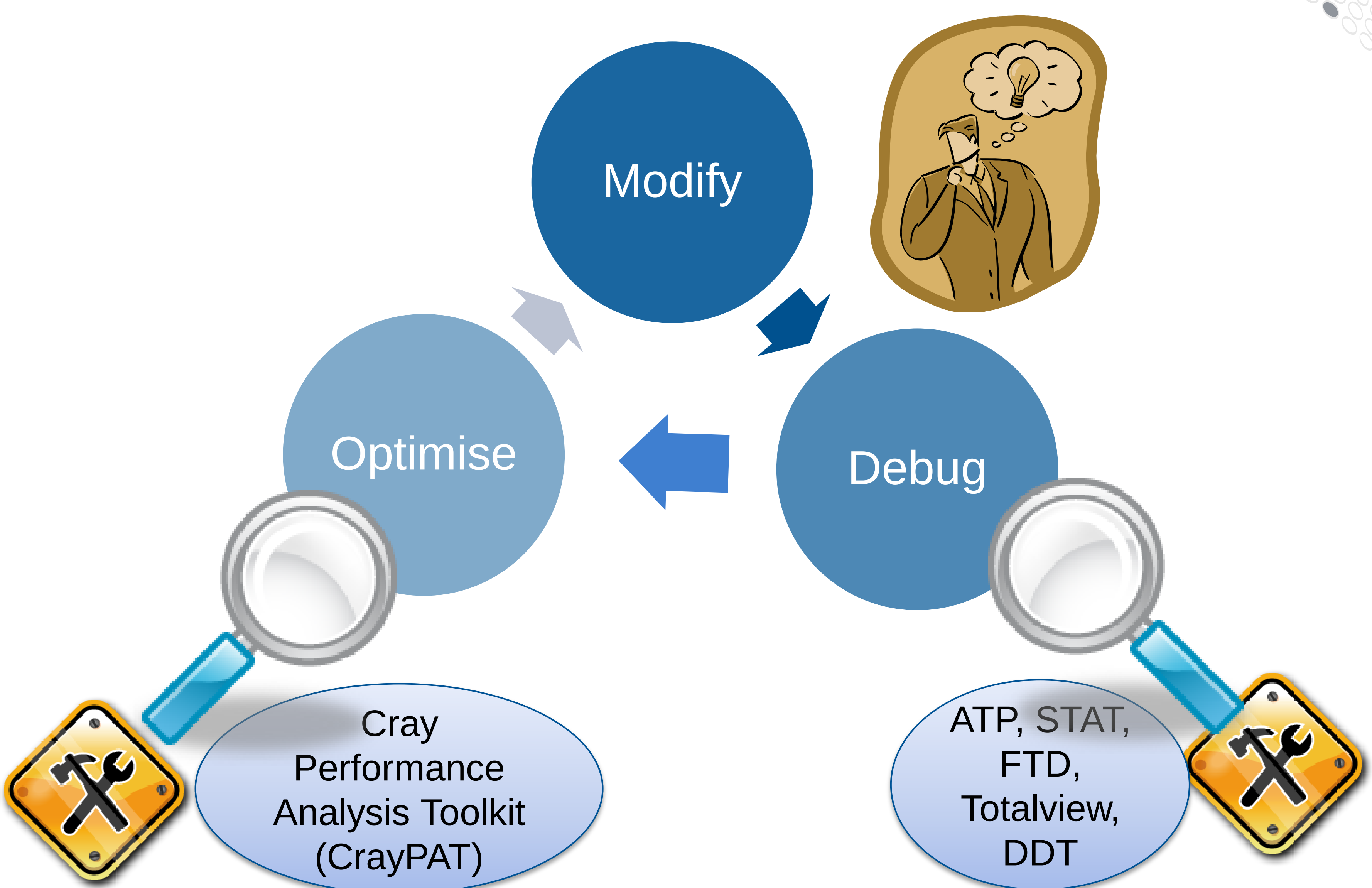
Short Introduction to Tools on the Cray XC systems

Assisting the port/debug/optimize cycle

The Porting/Optimisation Cycle



Introduction

- Cray develops several tools for XC and CS computers
  - There is lot of effort going into the development
- Several of the tools are ‘stand-alone’ solutions, being developed for a specific problem
  - STAT, ATP
  - IOBUF (includes serial IO monitoring)
  - MPIIO profiling
- Other tools will work together in order to be more efficient or to create new solution for a problem
  - CCE providing ‘hooks’ for profiling on loop level
  - Reveal using CCE listing information and CrayPat Profiling

Which tools does Cray develop

- It doesn't make sense to develop tools where a good tool already exists on the market
  DDT and Totalview are good examples

- Cray's tools are either
  - Something new, like Reveal
  - Concentrate on a solution to a specific issue, like STAT
  - Are part of the development process, like MPIIO Stats
  - Comes out of benchmarking, like IOBUF

- Cray also collaborate with different sites in developing the tools

CCE : Cray Compiler Environment

- The compiler is in general not considered a ‘tool’, but in fact it is the most important piece of user software
  - Compiles and Link the user application
  - Feedback about the application
    - Code errors
    - How optimization was done/or not done (lst file)
  - Providing ‘hooks’ into different levels of the application, to which other tools can attach
    - Functions
    - Loops
- This makes CCE the ‘centerpiece’ in Cray’s Tools Strategies
  - CCE can adapt rather quickly to user/tool needs
  - All Cray tools will work with other Compilers, but there might be some limitations
    The goal is not to force a user to use CCE, but to provide extensions where it makes sense

Overview : Tools infrastructure (selection)

	Light weight At most relinking. Get a first picture of a performance or problems during execution.	**In-depth** Recompile/Relink. Provides detailed information at user routine level.
Debugging Get your code up and running correctly.	• ATP • STAT	• lgdb with ccdb • Fast track • DDT • Totalview • Intel Inspector
Profiling Locate performance bottlenecks.	• CrayPAT-lite • IOBUF • MPIIO Stats	• CrayPAT • Apprentice2 • Reveal • Intel Vtune

Abnormal Termination Processing (ATP)

- For when things break unexpectedly…
- (Collecting back-trace information)

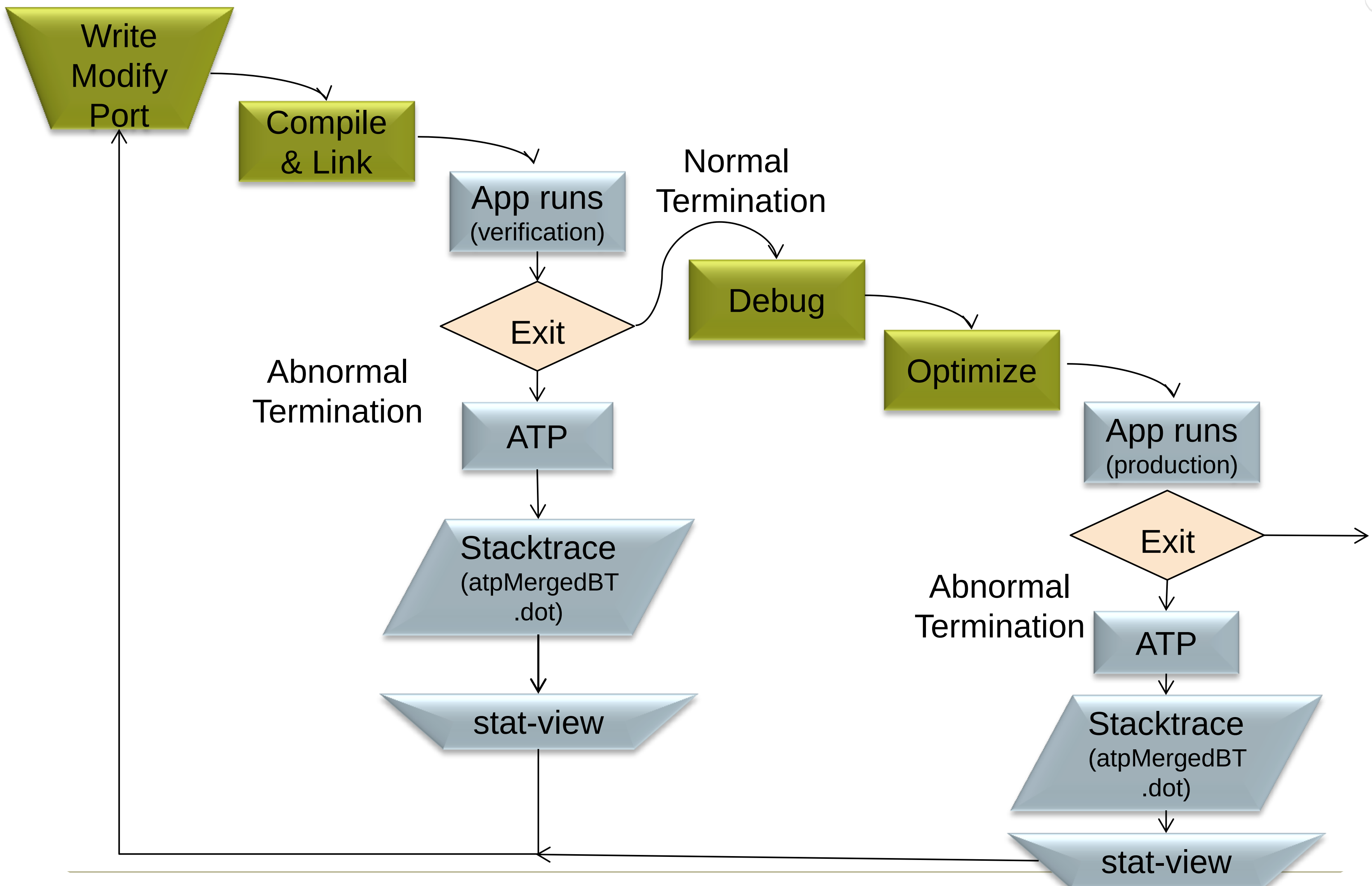
Debugging in production and scale

- **Even with the most rigorous testing, bugs may occur during development or production runs.**
  - It can be very difficult to recreate a crash without additional information
  - Even worse, for production codes need to be efficient so usually have debugging disabled
- **The failing application may have been using tens of or hundreds of thousands of processes**
  - If a crash occurs one, many, or all of the processes might issue a signal.
  - We don’t want the core files from every crashed process, they’re slow and too big!
  - We don’t want a backtrace from every processes, they’re difficult to comprehend and analyze.

ATP Description

- **Abnormal Termination Processing is a lightweight monitoring framework that detects crashes and provides more analysis**
  - Designed to be so light weight it can be used all the time with almost no impact on performance.
  - Almost completely transparent to the user
    - Requires `atp` module loaded during compilation (usually included by default)
    - Output controlled by the `ATP_ENABLED` environment variable (set by system).
  - Tested at scale (tens of thousands of processors)
- **ATP rationalizes parallel debug information into three easier to user forms:**
  1. A single stack trace of the first failing process to stderr
  2. A visualization of every processes stack trace when it crashed
  3. A selection of representative core files for analysis

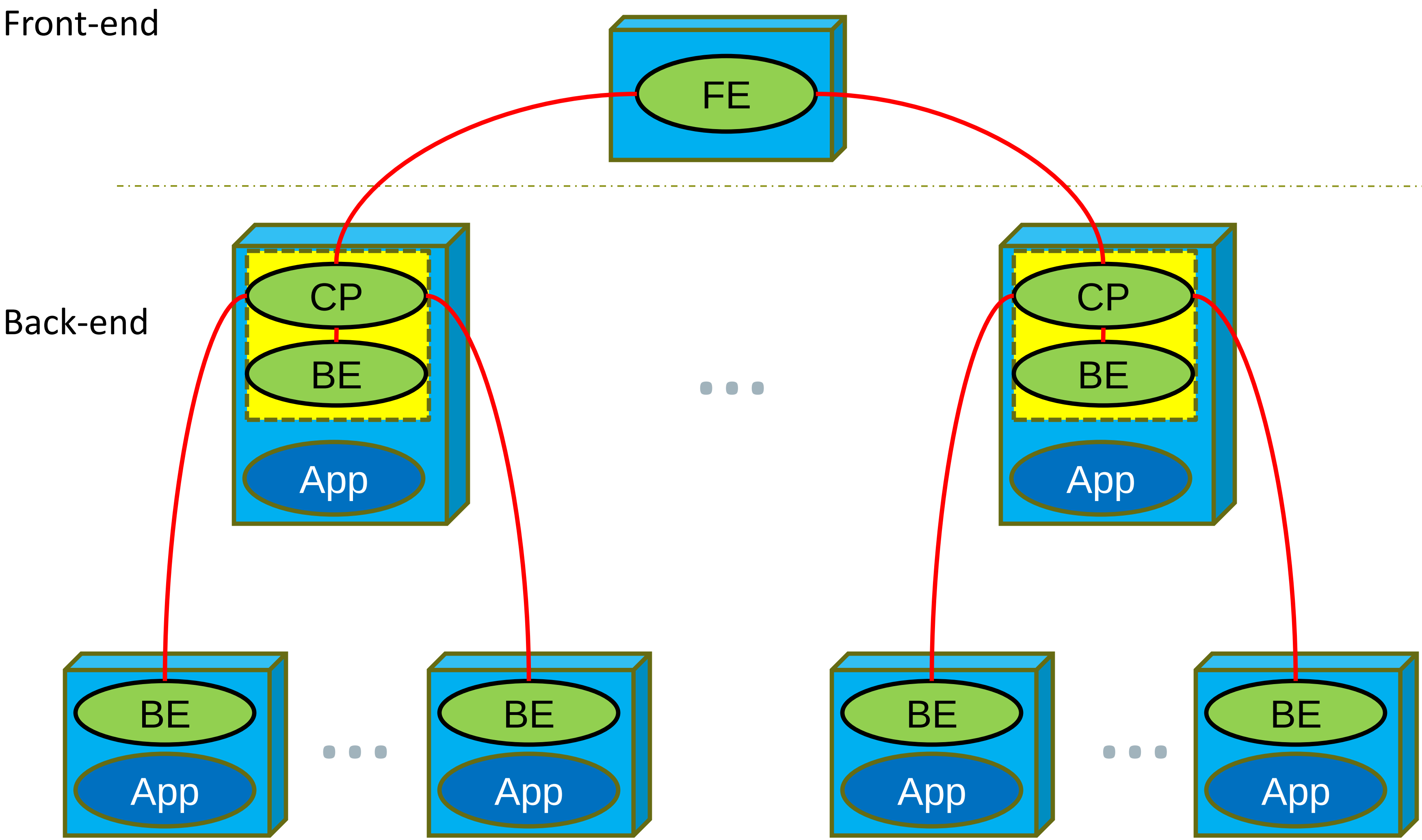
ATP – Abnormal Termination Processing



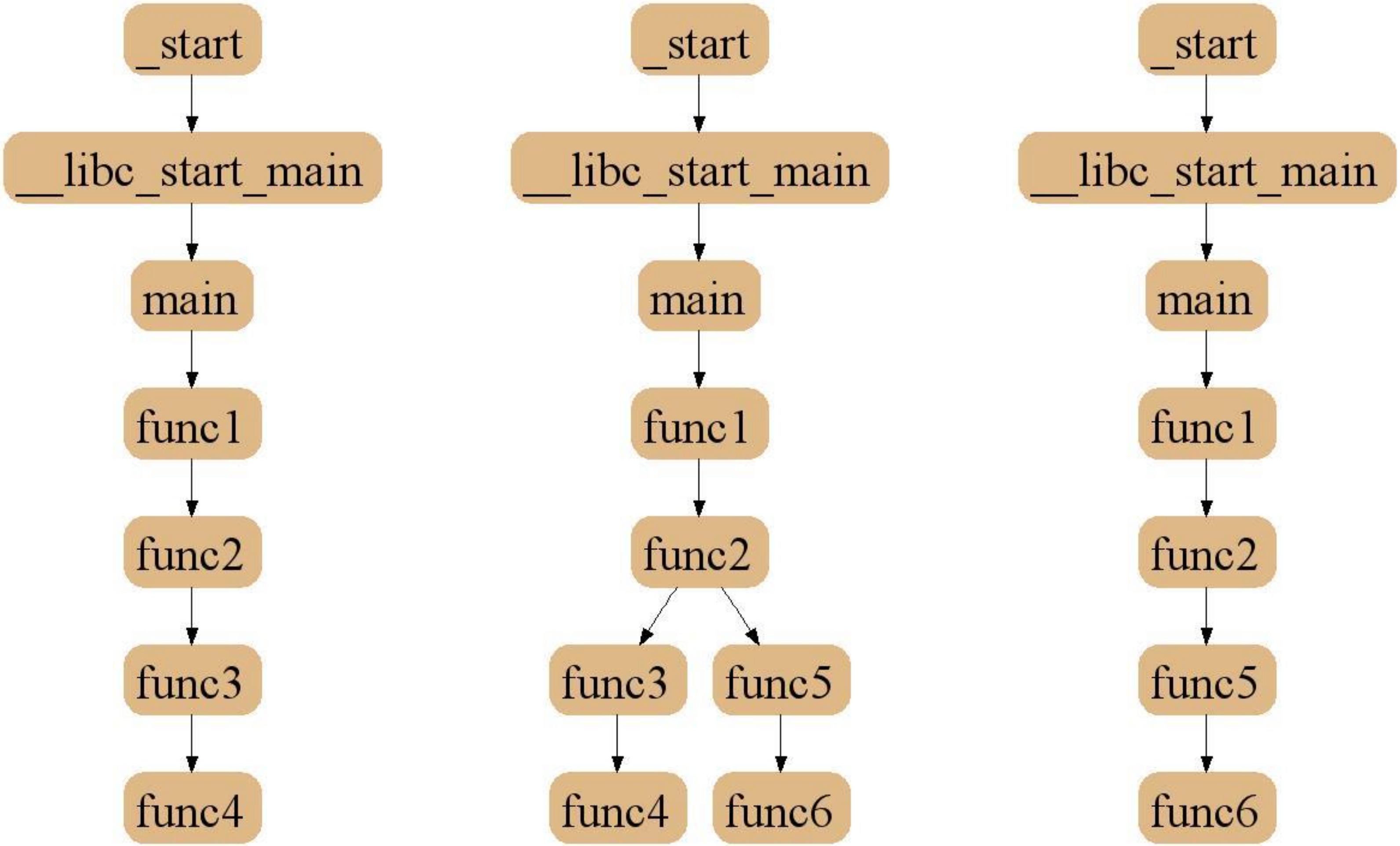
ATP Components

- **Application process signal handler**
  - triggers analysis
  - controls its own core_pattern
- **Back-end monitor**
  - collects backtraces via StackwalkerAPI
  - forces core dumps as directed
- **Front-end controller**
  - coordinates analysis via MRNet
  - selects process set that is to dump core
- **Once initial set up complete, all components comatose**

ATP Communications Tree



Stack Trace Merge Example



Usage

Compilation – environment must have module loaded

```
module load atp
```

Execution (scripts must explicitly set these if not included by default)

```
export ATP_ENABLED=1
ulimit -c unlimited
```

ATP respects `ulimits` on corefiles. So to see corefiles the `ulimit` must change.
On crash ATP will produce a selection of relevant cores files with unique, informative names.

More information (while atp module loaded)

```
man atp
```

Viewing the results - stderr

```
Application 867282 is crashing. ATP analysis proceeding

Stack walkback for Rank 16 starting:
  [empty]@0xffffffffffffffff
  funcA@crash.c:8
Stack walkback for Rank 16 done
Process died with signal 11: 'Segmentation fault'
Forcing core dumps of ranks 16, 0
View application merged backtrace tree with: statview atpMergedBT.dot
You may need to: module load stat

_pmiu_daemon(SIGCHLD): [NID 00752] [c3-0c2s1
 PE RANK 0 exit signal Segmentation fault
[NID 00752] 2013-02-12 19:08:18 Apid 867282:
ion
_pmiu_daemon(SIGCHLD): [NID 00753] [c3-0c2s12n1] [Tue Feb 12 19:08:18 2013]
 PE RANK 16 exit signal Segmentation fault
Application 867282 exit codes: 139
Application 867282 resources: utime ~2s, stime ~2s
slurm-10340.out lines 1-16/16 (END)
```

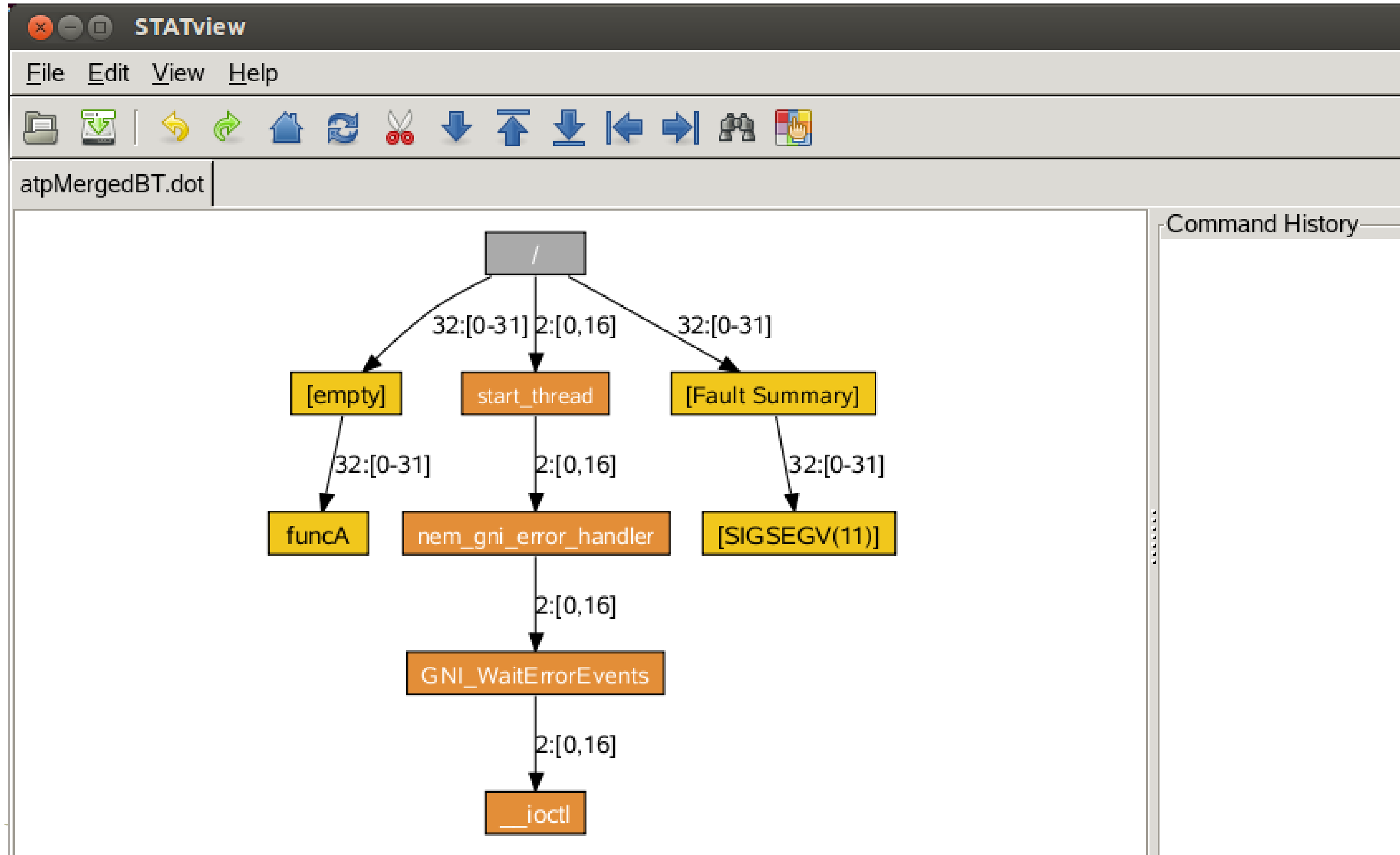
Trace back of crashing process

Core files being generated

Example output in stderr.

Viewing the results – merged backtrace

```
module load stat
stat-view atpMergedBT.dot
```

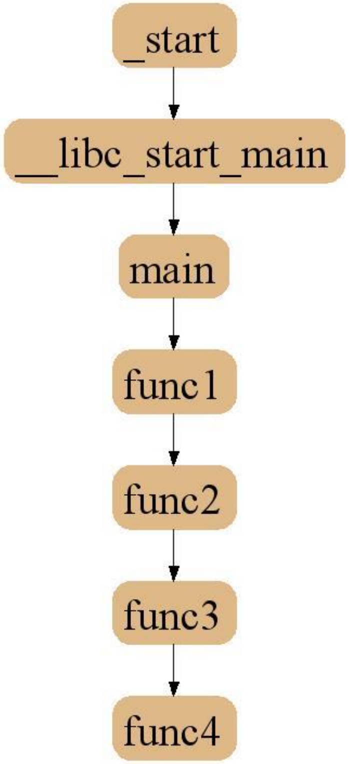


Stack Trace Analysis Tool (STAT)

- For when nothing appears to be happening…

STAT

- Stack Trace Analysis Tool (STAT) is a cross-platform tool from the University of Wisconsin-Madison.
- ATP is based on the same technology as STAT. Both gather and merge stack traces from a running application's parallel processes.
- It is very useful when application seems to be stuck/hung
- Full information including use cases is available at http://www.paradyn.org/STAT/STAT.html
- Scales to many thousands of concurrent process, only limited by number file descriptors



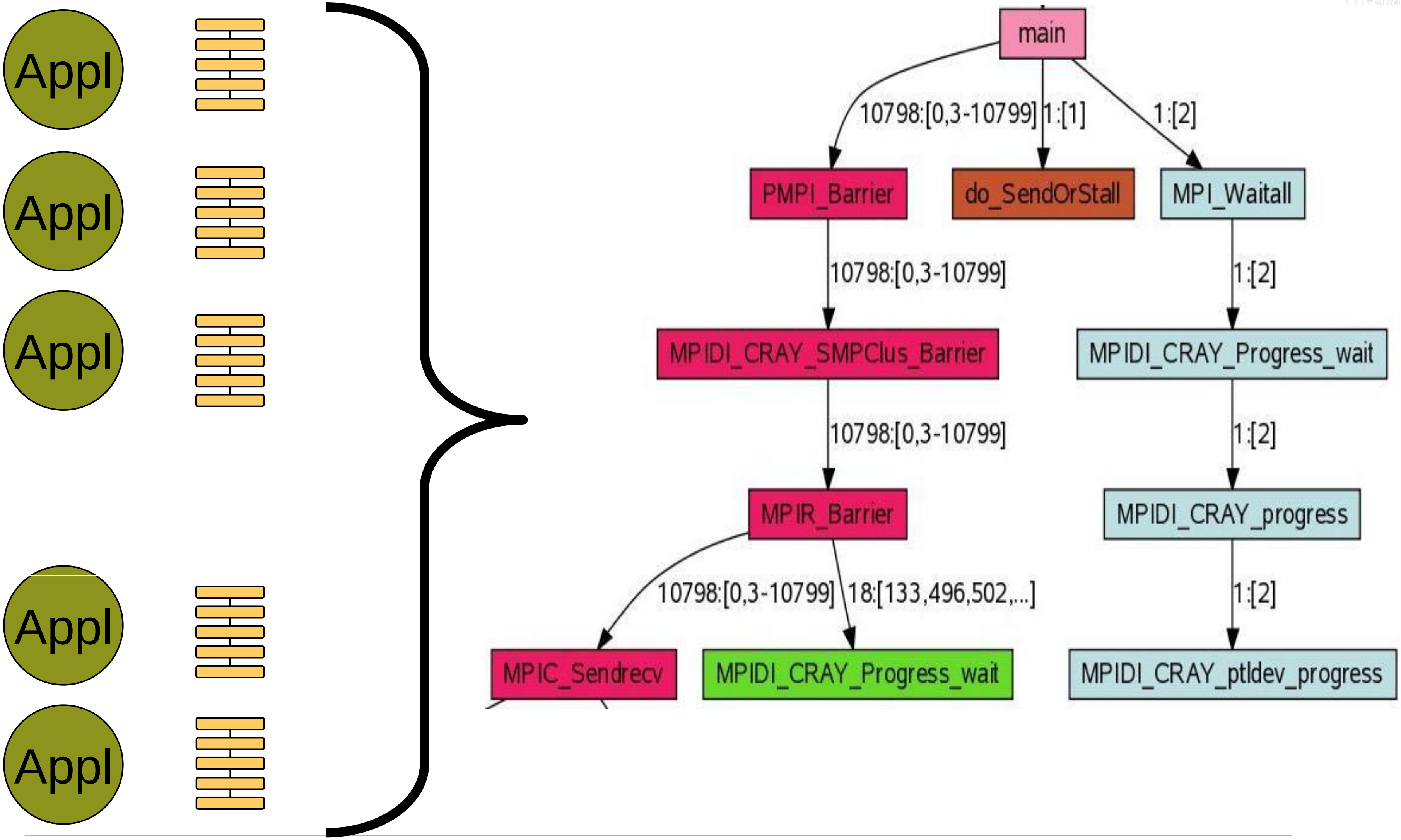
Stack Trace Analysis Tool (STAT)

- **Stack trace sampling and analysis for large scale applications**
  - Reduce number of tasks to debug
  - Discover equivalent process behavior
- **Extreme scaling**
  - Jaguar – 216K processes
  - BG/L – 208K processes

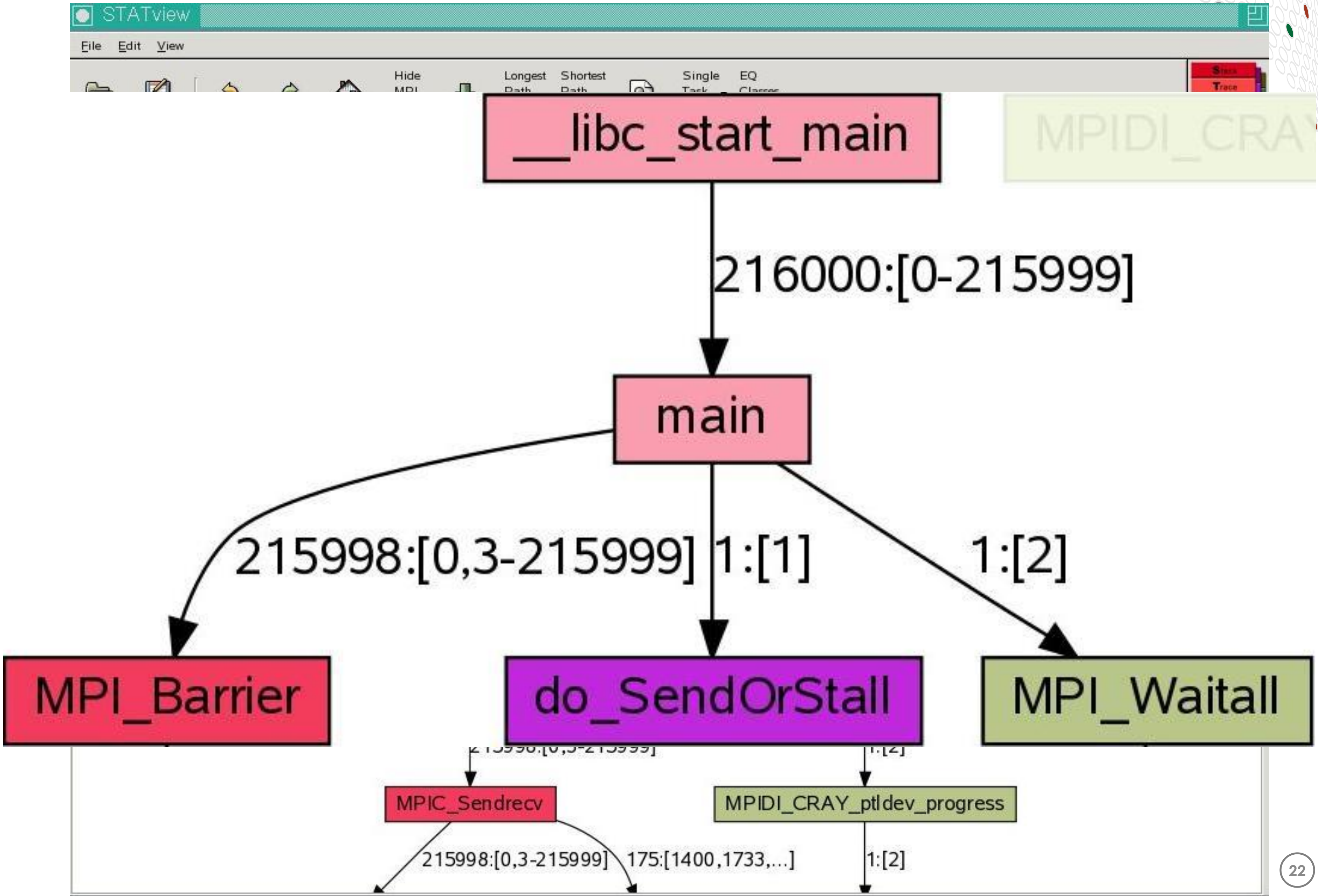
Merging Stack Traces

- **Multiple traces over space or time**
- **Create call graph prefix tree**
  - Compressed representation
  - Scalable visualization
  - Scalable analysis

2D-Trace/Space Analysis



Merged Stack for Cray XT



Using STAT

Start an interactive job…

```
module load stat

<launch job script> &

# Wait until application hangs:

stat-cl <pid of aprun>

# Kill job

stat-view STAT_results/<exe>/<exe>.0000.dot
```

LGDB

- **Diving in through the command line…**

lgdb - Command line debugging

- **LGDB is a line mode parallel debugger for Cray systems**
  - Available through `cray-lgdb` module
  - Binaries should be compiled with debugging enabled, e.g. –g. (Or Fast-Track Debugging see later).
  - The recent 2.0 update has introduced new features. All previous syntax is deprecated
- **It has many of the features of the standard GDB debugger, but includes extensions for handling parallel processes.**

It can launch jobs, or attach to existing jobs

1. **To launch a new version of <exe>**
   1. Launch an interactive session
   2. Run `lgdb`
   3. Run `launch $pset{nprocs}` <exe>
2. **To attach to an existing job**
   1. find the `<apid>` using `apstat`.
   2. launch `lgdb`
   3. run `attach $<pset> <apid>` from the `lgdb` shell.

LGDB process groups

Debugging commands are issued in parallel to all processes in the "focus" group. By default this is `$<pset>`, all the processors in the application.

Output from commands is grouped into common sets, e.g. backtraces (`bt`) will be prepended with groups, e.g.

```
bt
all[0..15]: #0  0x00000000004009cf in main at /tdsnfs1/y02/y02/ted/xthi.c:55
```

Or

```
bt
all[0,2..31]: #0  0x0000000000400979 in main at /tdsnfs1/y02/y02/ted/xthi.c:47
all[1]: #0  0x0000000000400984 in main at /tdsnfs1/y02/y02/ted/xthi.c:48
```

LGDB process groups

New groups can be created

```
defset $<newgrp> $<pset>{rank1},$<pset>{rank37}
```

Changing focus can be changed with

```
focus $<newgrp>
```

Changing focus can be changed with

```
focus $&lt;newgrp&gt;
```

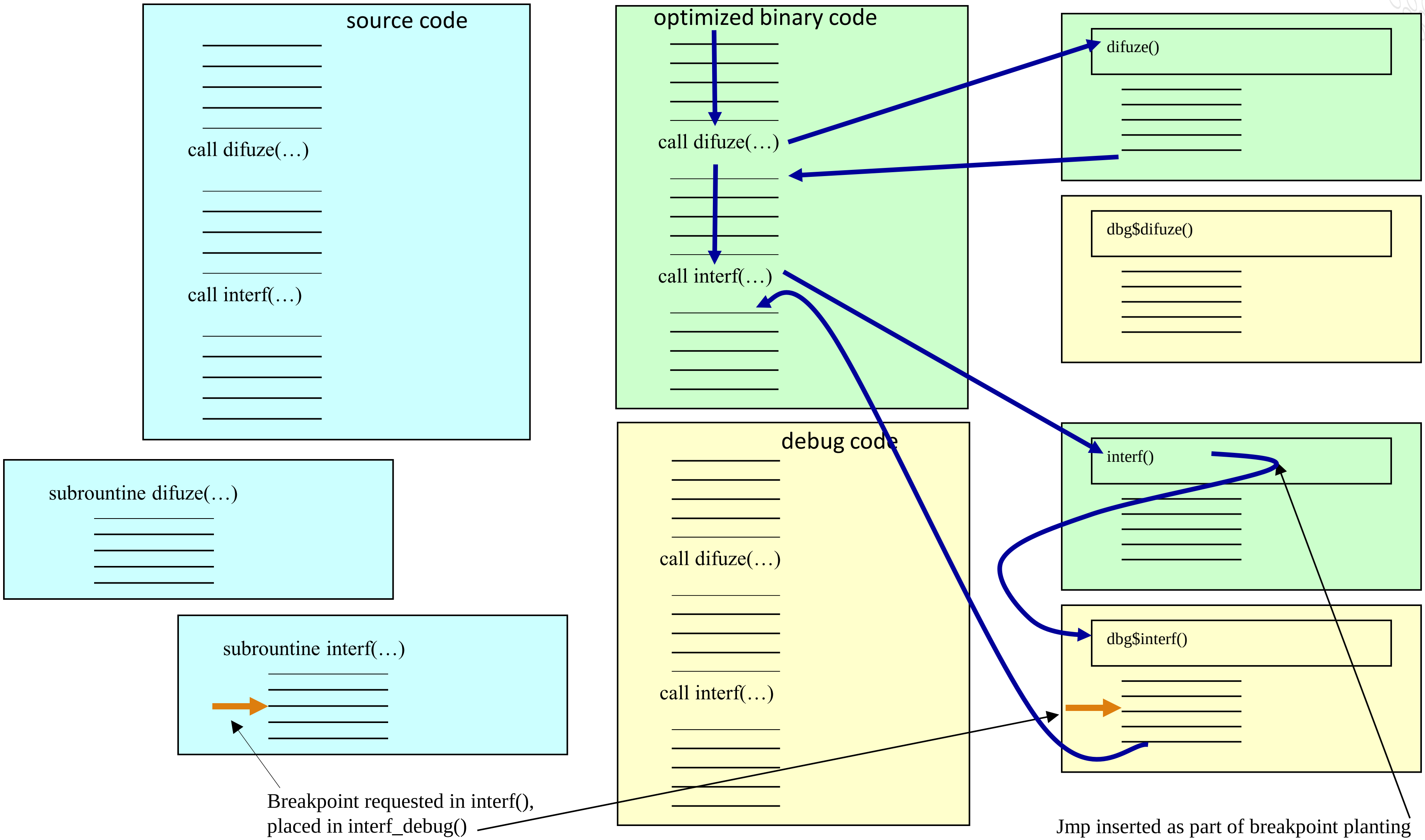
Fast Track Debugging

- For getting to the problem more quickly…

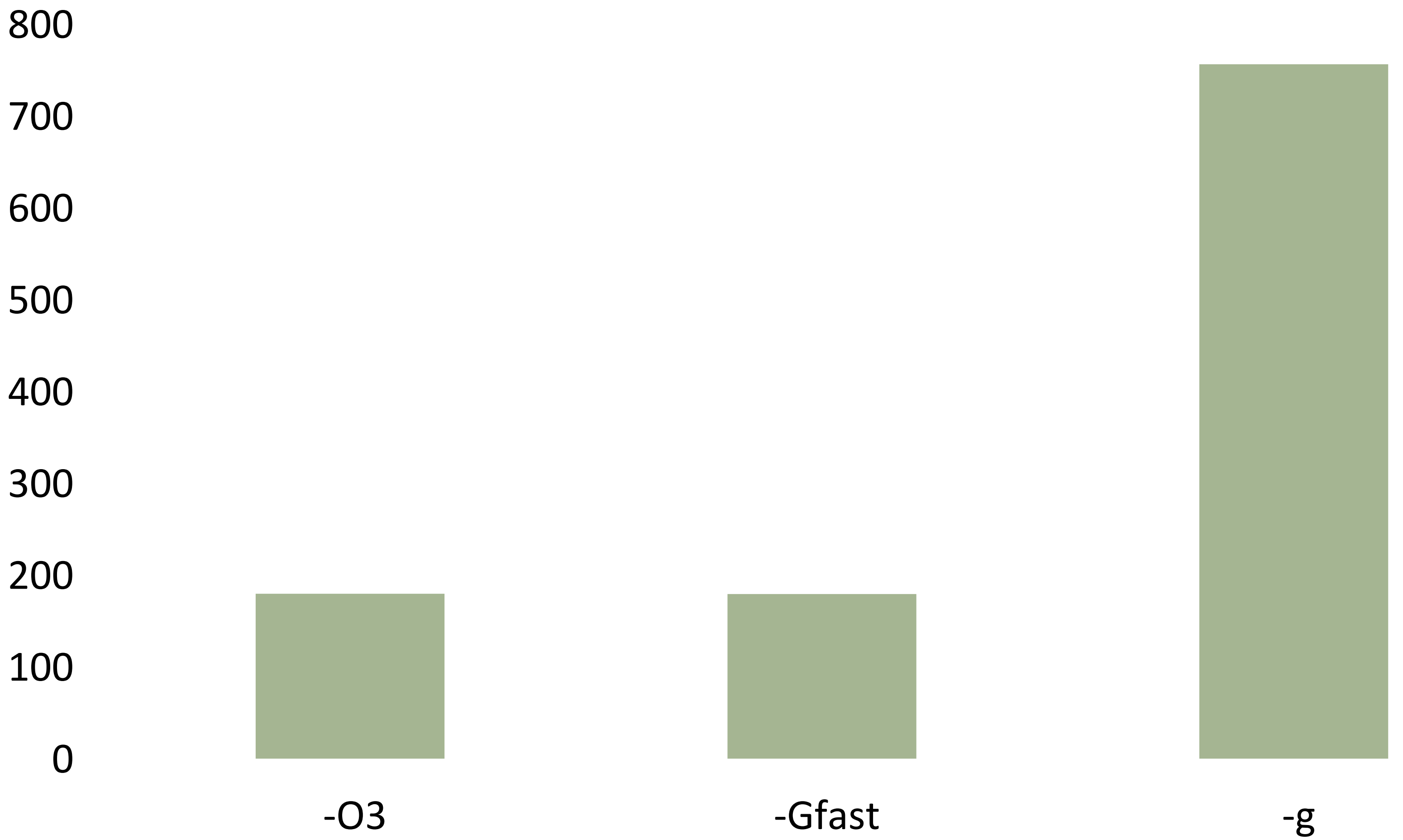
The Problem

- **Debug compilations eliminate optimizations**
  - Today's machines really need optimizations
  - Slows down execution
  - Problem might disappear
- **Compile such that both debug and non-debug (optimized) versions of each routine are created.**
- **Use –Gfast instead of –g with the Cray compiler.**
- **Linkage such that optimized versions are used by default**
- **Debugger overrides default linkage when setting breakpoints and stepping into functions**
- **Supported by DDT**

A Closer Look at How FTD Works



Tera TF Execution Time



–Gfast is 320% faster than –g

Cost of Fast Track Debugging

- **Compiles are slower**
- **Executable uses more disk space**
- **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

ccdb: Comparative debugger

- ccdb is a tool to allow comparison of two runs
- You can define expressions to be compared between runs

Usage:

- Launch both applications with lgdb
- Declare a decomposition scheme (for example 1d on 4 processes block distributed) to be used for comparisons
- Create comparisons by tying together variables at source locations using this scheme.
- Then run the programs – they will stop when the comparison fails

- See S-0042-22

Debugging Tools Recap

A range of tools are provided to help with debugging

- ATP
- STAT
- lgdb
- Ccdb

- use when appropriate