The pC++ System Programming Environment

Part II:

The Program and Performance Analysis Environment τ

Allen Malony Bernd Mohr

Dept. of Computer and Information Science
University of Oregon
Eugene, Oregon 97403

{malony,mohr}@cs.uoregon.edu

Motivation

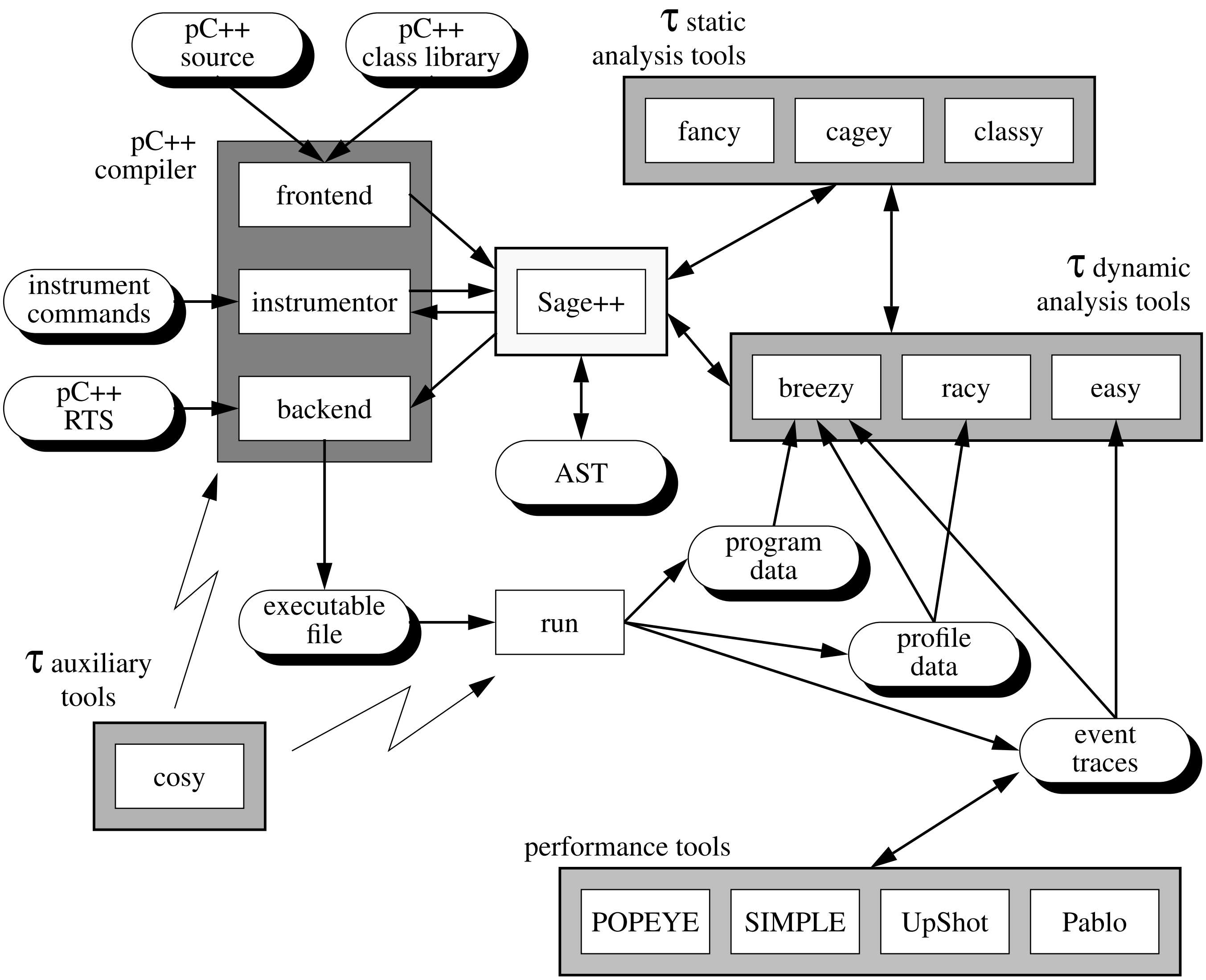
- High-level parallel language compilers (HPF, pC++, CC++, ...) generate code
  - node programs in conventional Fortran or C++
  - with runtime system library calls using message-passing (MPI, PVM, p4, RSR, ...) or shared memory operations

- **Problem**: program and performance analysis
  - ➠ currently used analysis tools (ParaGraph, XPVM, ...) show mainly low-level message-passing
  - ➠ don't show relation to source code
  - ➠ doesn't help the **user** in program optimization

- Necessary **requirements** for high-level parallel language programming environments
  - Providing a user (program-level) view
  - Support for high-level, parallel programming languages
  - Integration with compilers and runtime systems
  - Portability, extensibility, and retargetability
  - Usability

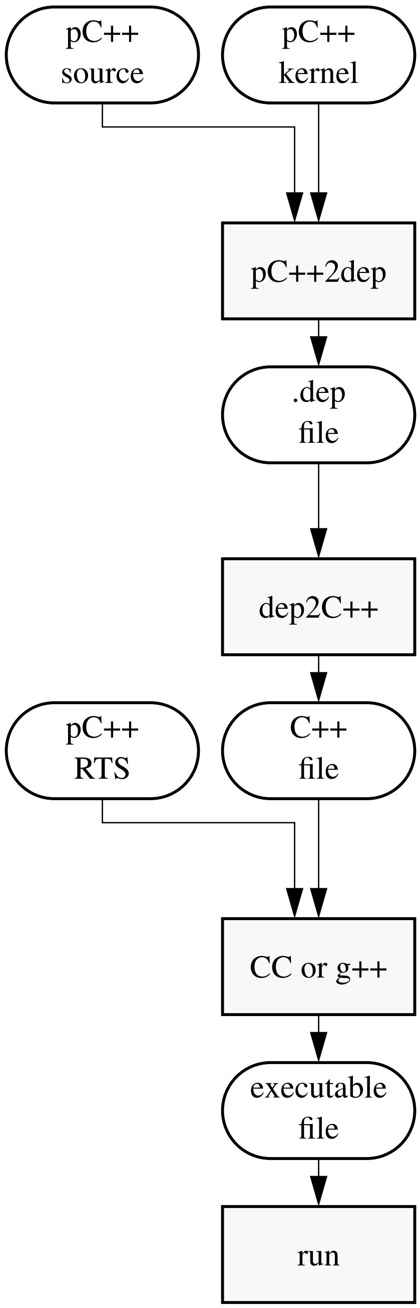
➠ **Our solution**: τ (TAU: ***T***uning and ***A***nalysis ***U***tilities)

The pC++ Programming Environment

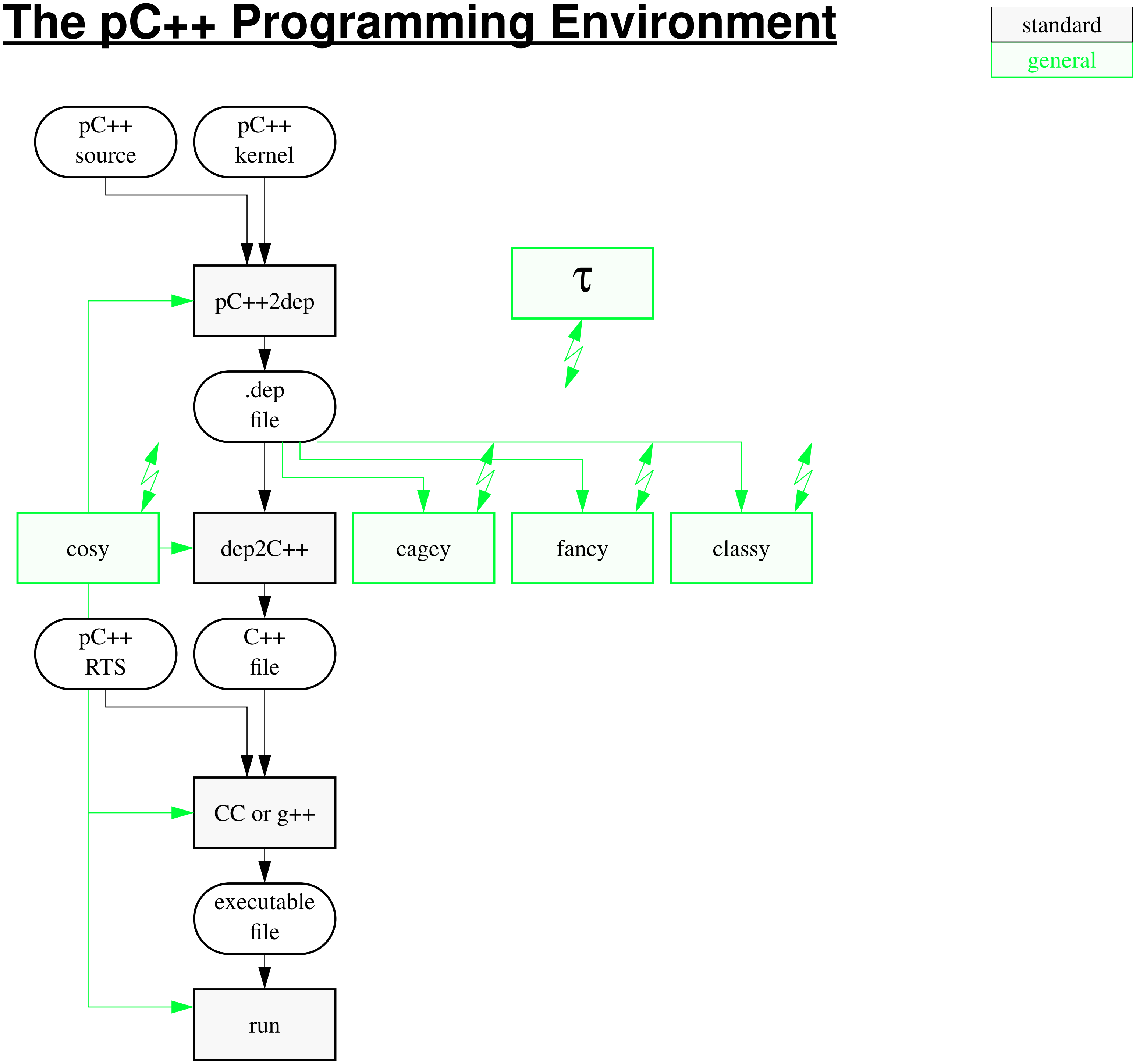


The pC++ Programming Environment

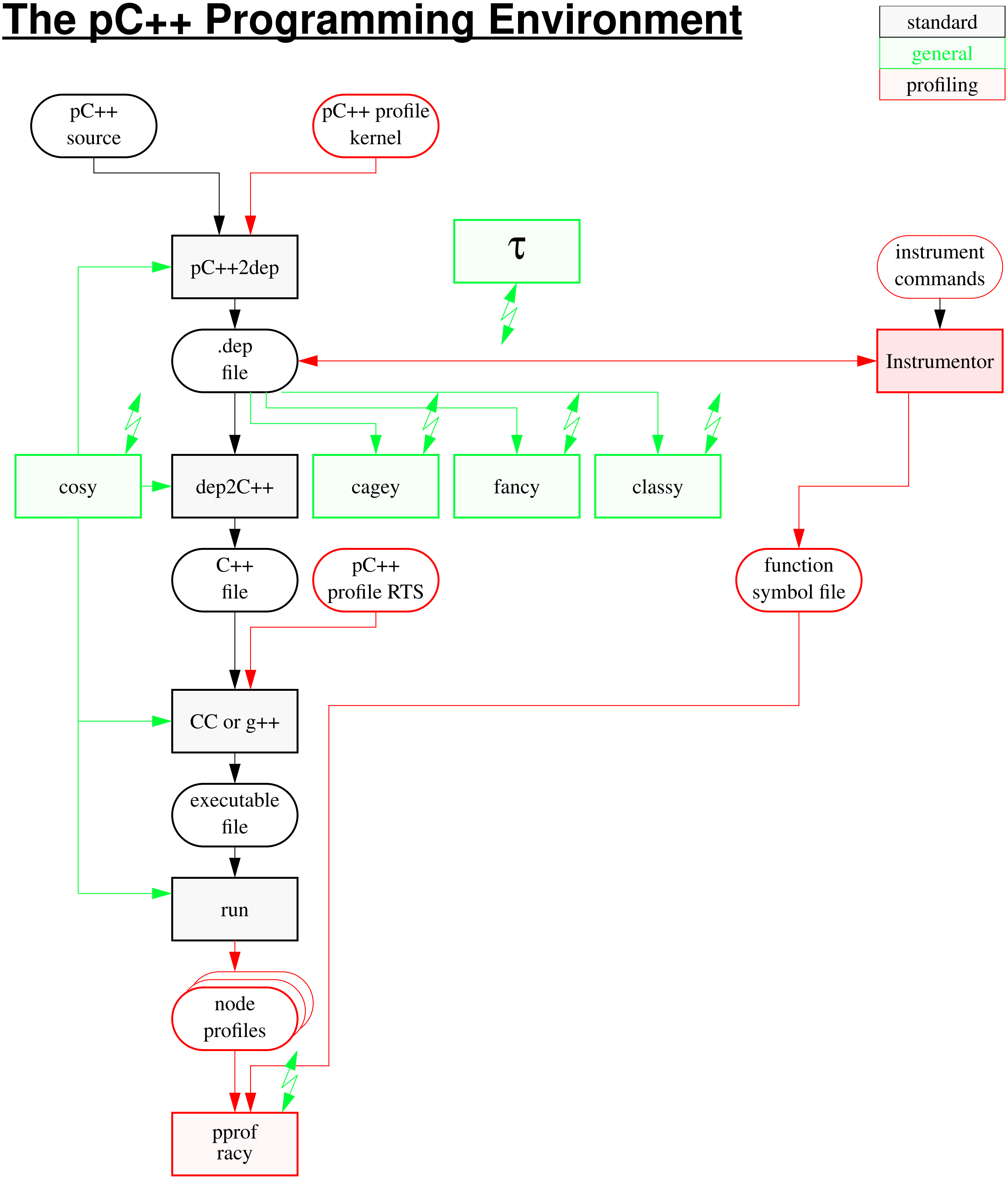
standard



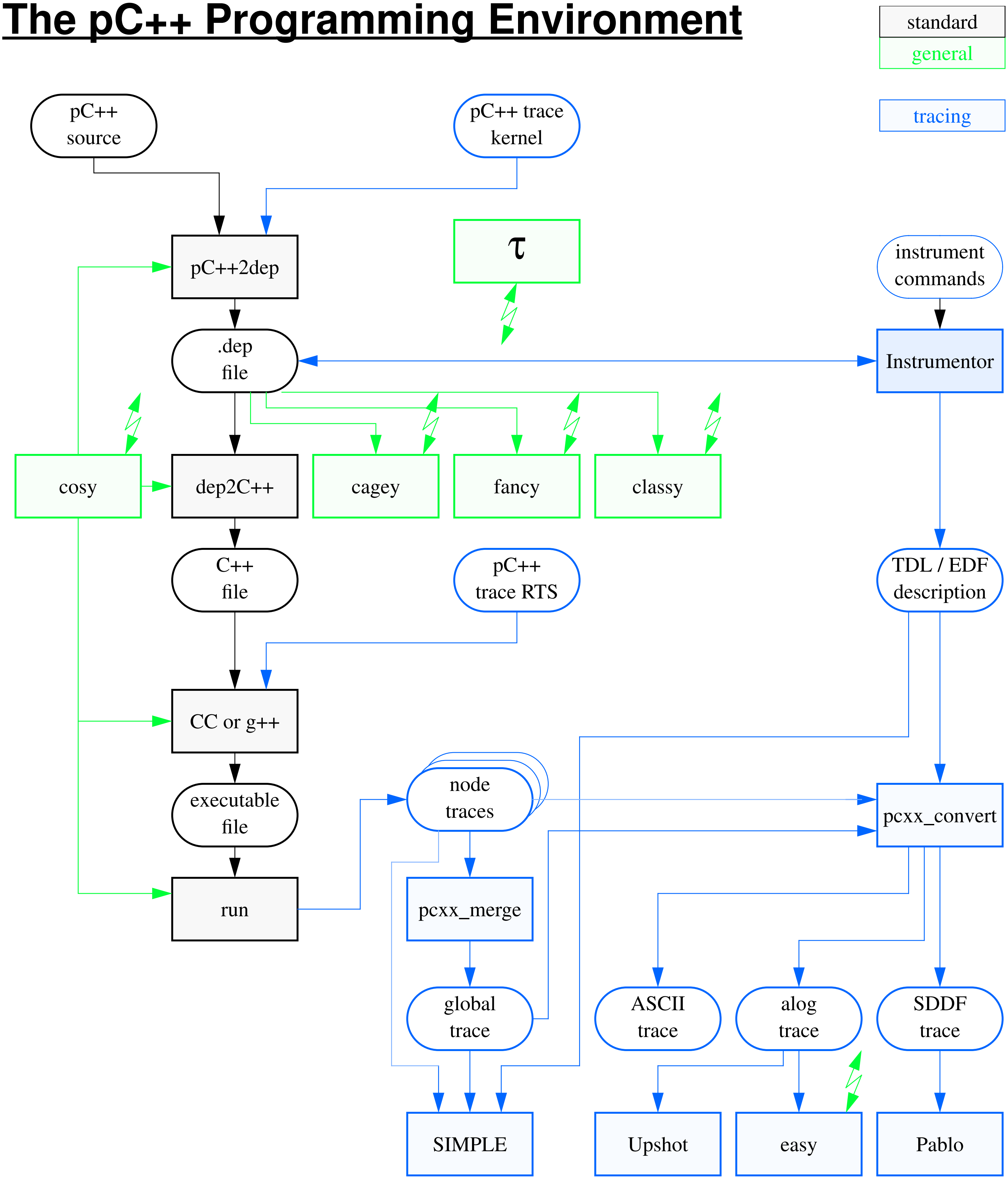
The pC++ Programming Environment



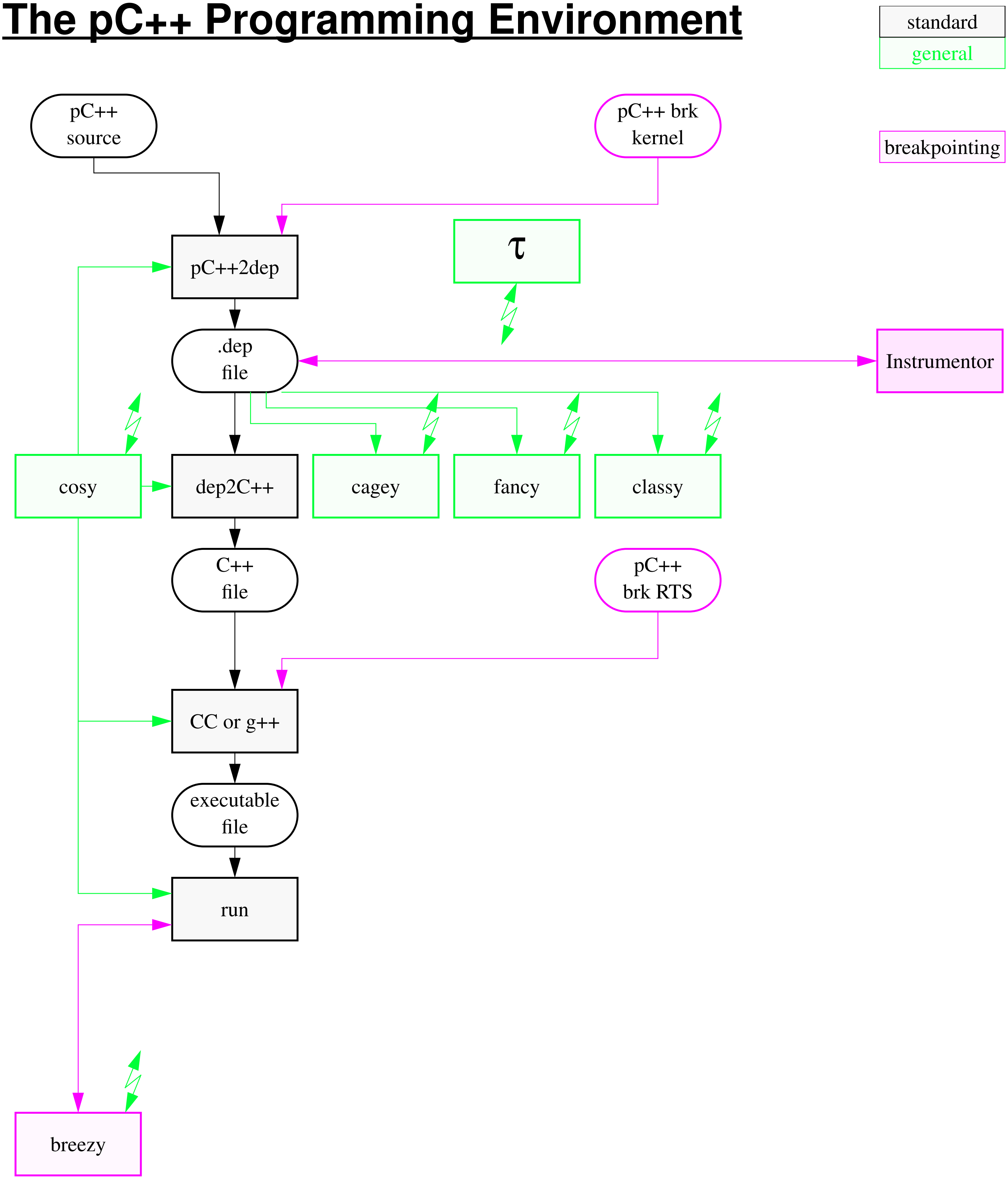
The pC++ Programming Environment



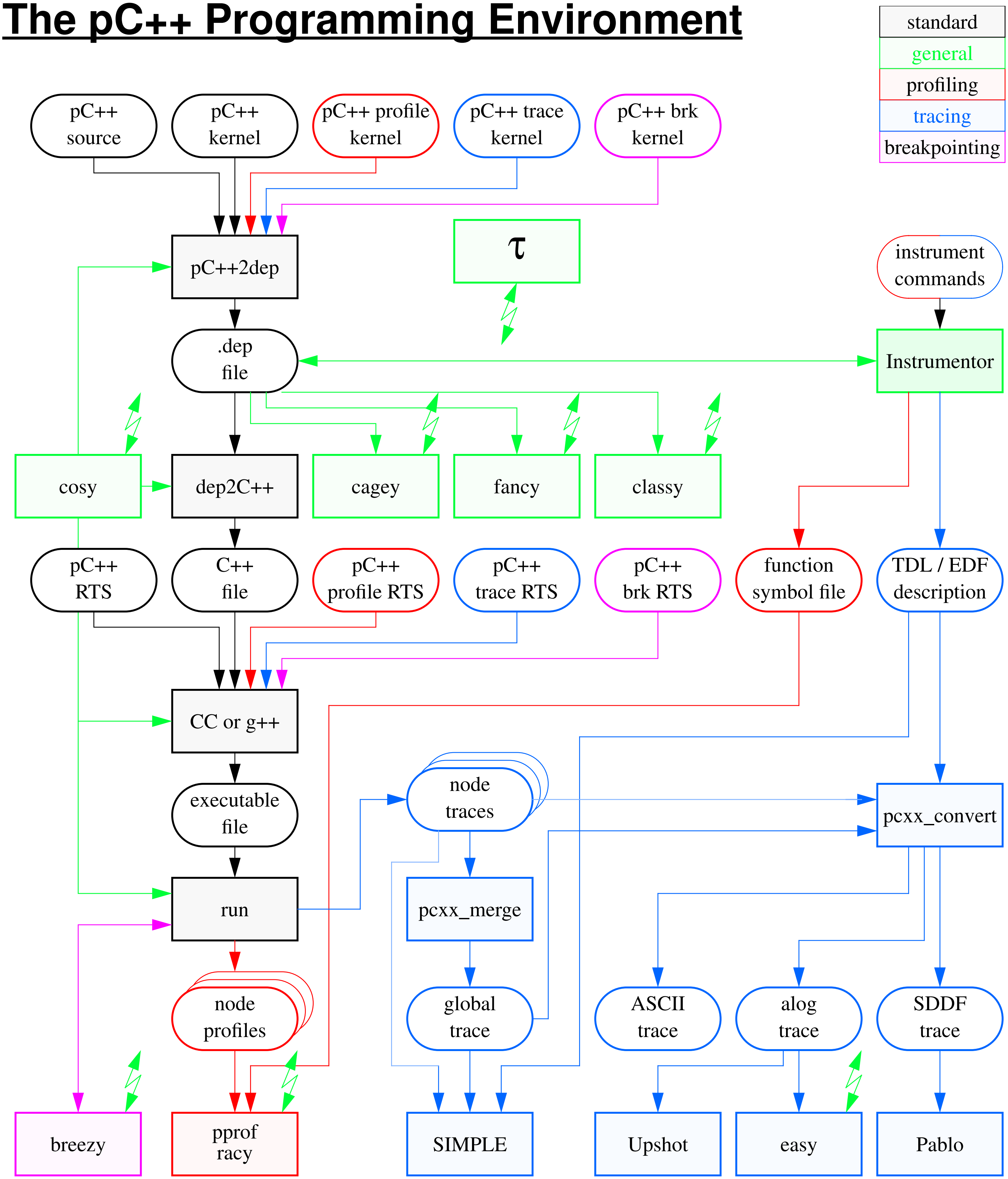
The pC++ Programming Environment



The pC++ Programming Environment

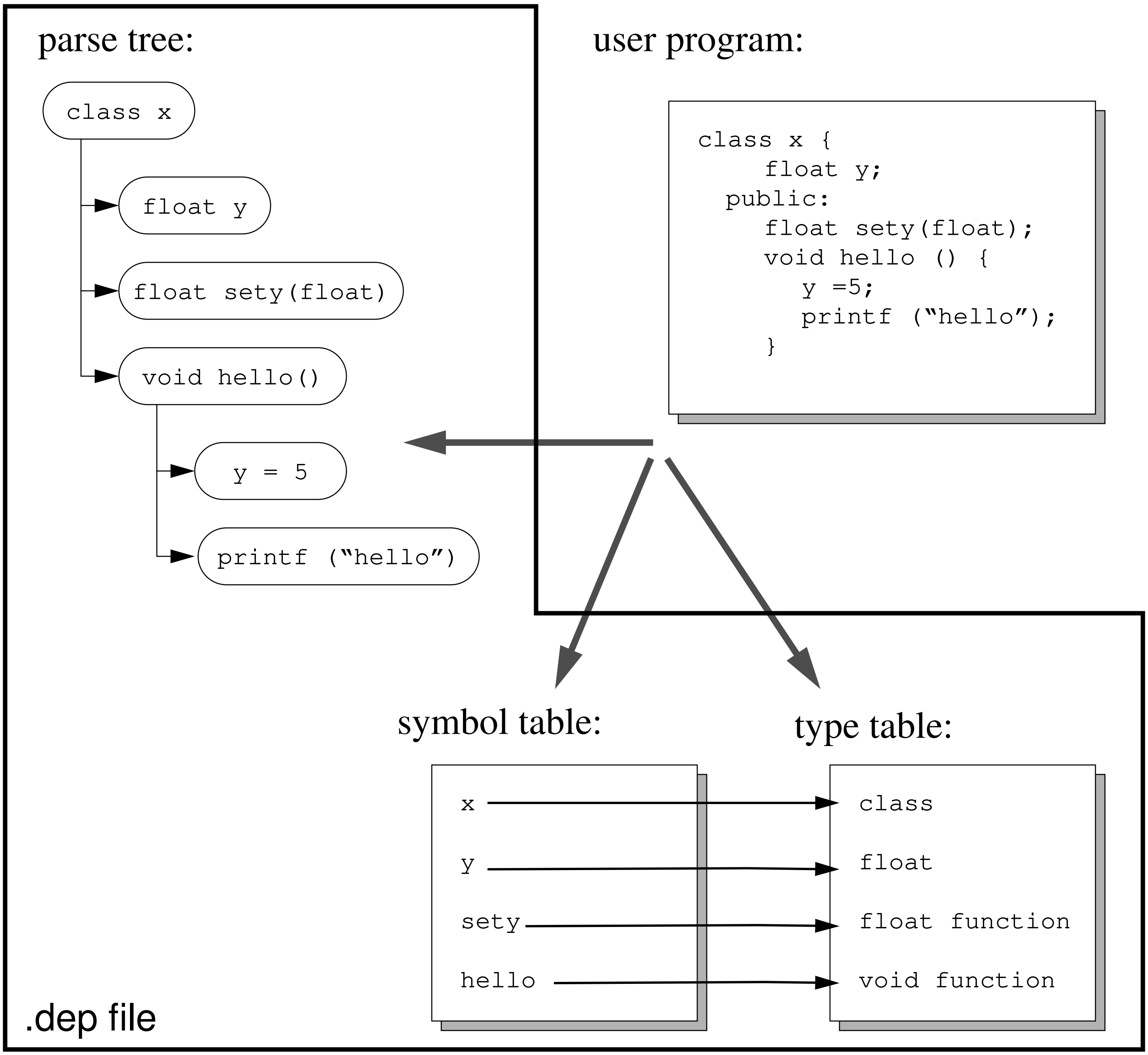


The pC++ Programming Environment



Sage++

- C++ class library for building program analysis and transformation tools
- contains parsers for Fortran 77 / 90, ANSI C, and C++
- organized as a class hierarchy for accessing and modifying the parse tree, symbol table, and type table



Internal Sage++ Applications

implemented:

- **pC++ compiler (dep2C++):**
  translates a .dep file into a C++ program with machine specific details
- **instrumentation tools (instr):**
  automatically inserting special code into the program to instrument for performance analysis and visualization purposes
- **algorithm animation system:**
  to watch the behavior of a pC++ program in “play back” mode
- **source text browser (fancy):**
  displays the pC++ source text and “highlights” functions, class methods or class definitions on request
- **call graph visualizer (cagey):**
  display the static call graph of a source program
- **class hierarchy browser (classy):**
  displays the class hierarchy graph and a table of class members on request

in progress:

- barrier breakpoint debugger (breezy)

planned:

- interface between the application source code and the performance expert systems Expose and Poirot

External Sage++ Applications

- Argonne National Laboratory:
  the automatic differenttiation project
- Argonne National Laboratory:
  The Fortran-M parallel programming system
- Los Alamos National Laboratory:
  The A++ Optimizer for array classes
- NASA Ames:
  The AIMS parallel performance analysis tools
- RIACS, NASA Ames:
  The RIACS parallel C++ experiments
- ~16 parallel research projects at Universities in the U.S, the U.K., France and Australia

τ

☆ Tools Are Us ☆

- ❑ Providing a user (program-level) view
  - ➠ τ graphical interface objects represent pC++ language level objects: collections, classes, methods, functions
- ❑ Support for high-level, parallel programming languages
  - ➠ τ designed and implemented in concert with pC++ system
  - ➠ τ translates low-level performance data ⇔ language level
- ❑ Integration with compilers and runtime systems
  - ➠ τ integrated with pC++ runtime system
  - ➠ τ uses Sage++ for access to ***P***rogram ***D***ata***B***ase (PDB)
- ❑ Portability, extensibility, and retargetability
  - ➠ τ implemented using C/C++ and Tcl/Tk for portability
  - ➠ τ implemented as **hypertools** for extensibilty
  - ➠ τ uses Sage++ for retargetability
- ❑ Usability
  - ➠ τ tool objects represent **hyperlinks**
  - ➠ τ supports **global features**
  - ➠ τ has on-line hypertext help

τ

☆ Tools Are Us ☆

- Currently available τ tools:
  - **cosy** (***CO***mpile manager ***S***tatus displa***Y***)
  - **fancy** (***F***ile ***AN***d ***C***lass displa***Y***)
  - **cagey** (***CA***ll ***G***raph ***E***xtended displa***Y***)
  - **classy** (***CLASS*** hierarch***Y*** browser)
  - **racy** (***R***outine and data ***AC***cess profile displa***Y***)

- Prototypes:
  - **easy** (***E***vent ***A***nd ***S***tate displa***Y***)
  - **breezy** (***BR***eakpoint ***E***xecutive ***E***nvironment for visuali***Z***ation and data displa***Y***)
  - **dandy** (***D***istributed ***A***rray ***N***avigator ***D***ispla***Y***)
  - **crafty** (***C***ont***R***ol flow ***A***nd ***F***unc***T***ion displa***Y***)
  - data distribution and performance visualization (POPEYE)

- τ can work with a local or remote pC++ language system
- τ currently restricted to C++/pC++ programs
- FORTRAN (Fortran 90, HPF) support planned

τ

☆ Tools Are Us ☆

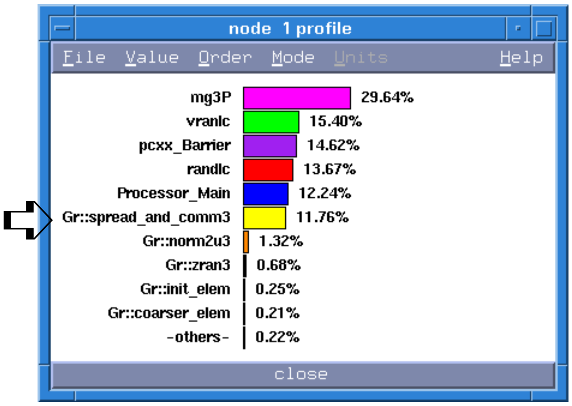
- **Hypertools**
  - τ tools are distinct tools, but they act in concert as if they were a single, monolithic application
  - implemented using **hyperlinks** and **global features**

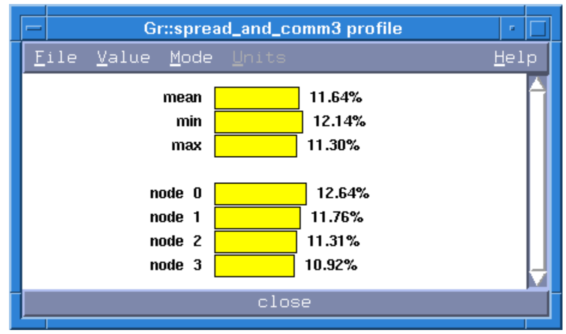
- **Hyperlinks**
  - τ graphical interface objects act like in hypermedia documents
  - selecting an object of interest brings up a more detailed or related view of the object

    e.g., selecting a class in the class hierarchy graph displays a table of class members

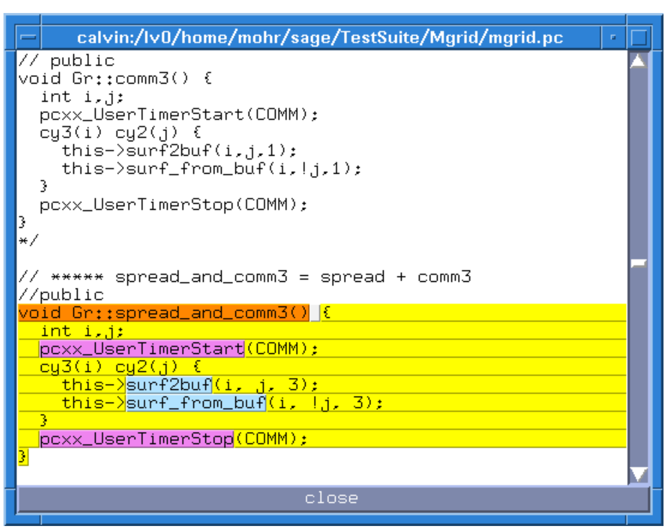
- **global features**
  - **synchronized hyperlinks**: execution of a global feature in one tool automatically updates views in the other tools
  - currently implemented:
    - `select-function`
    - `load-depfile`
    - `select-class`
    - `select-callsite`

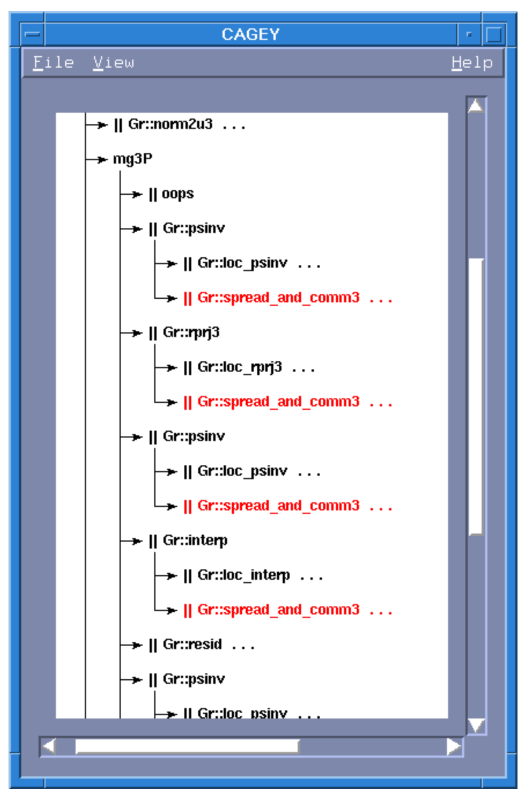
Implementation of global features





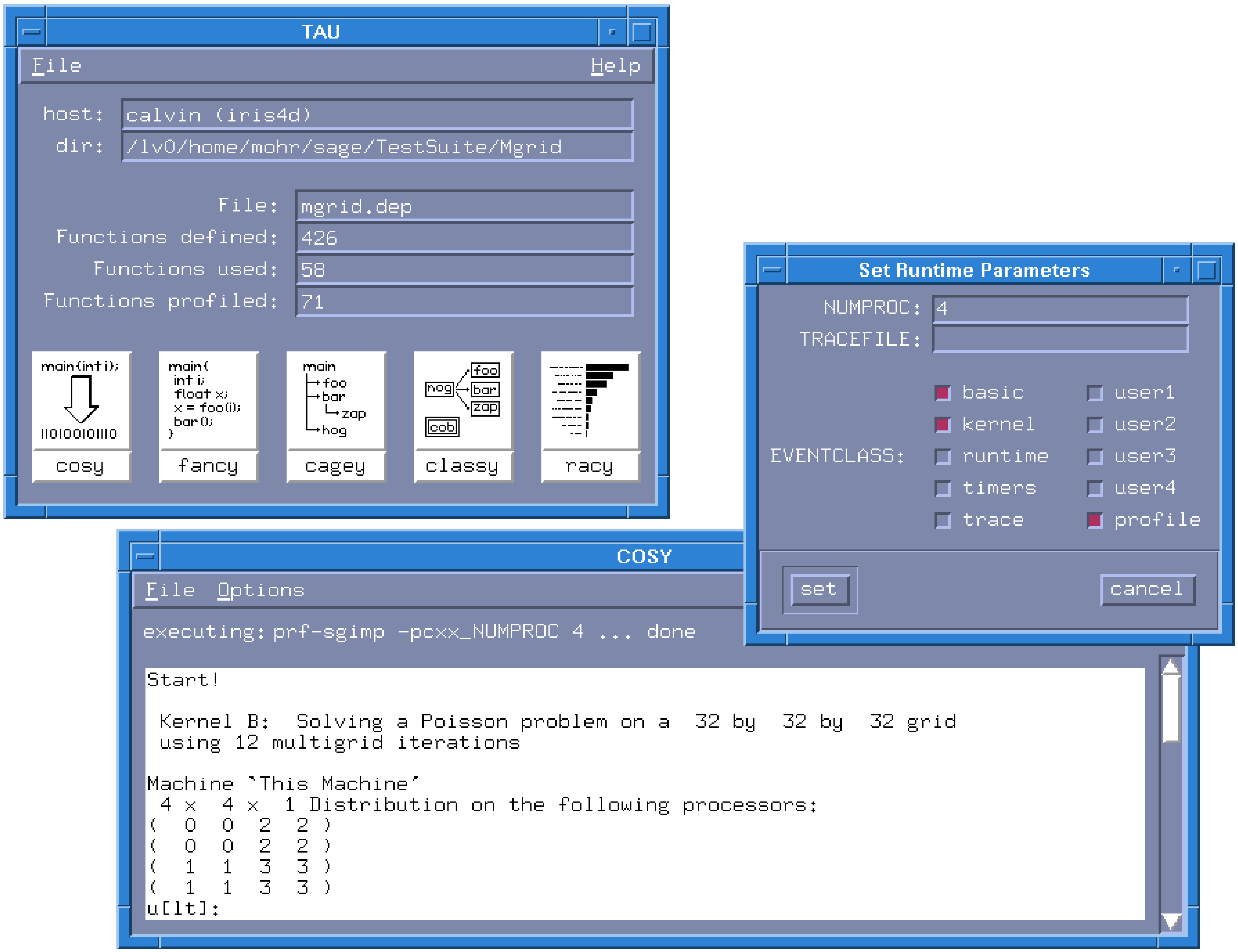

```
proc globalFeatureFoo {obj} {
   localFeatureFoo $obj
   foreach tool $FooToolList {
      if [ isRunning $tool ] {
         send $tool "localFeatureFoo $obj"
      }
   }
}
```





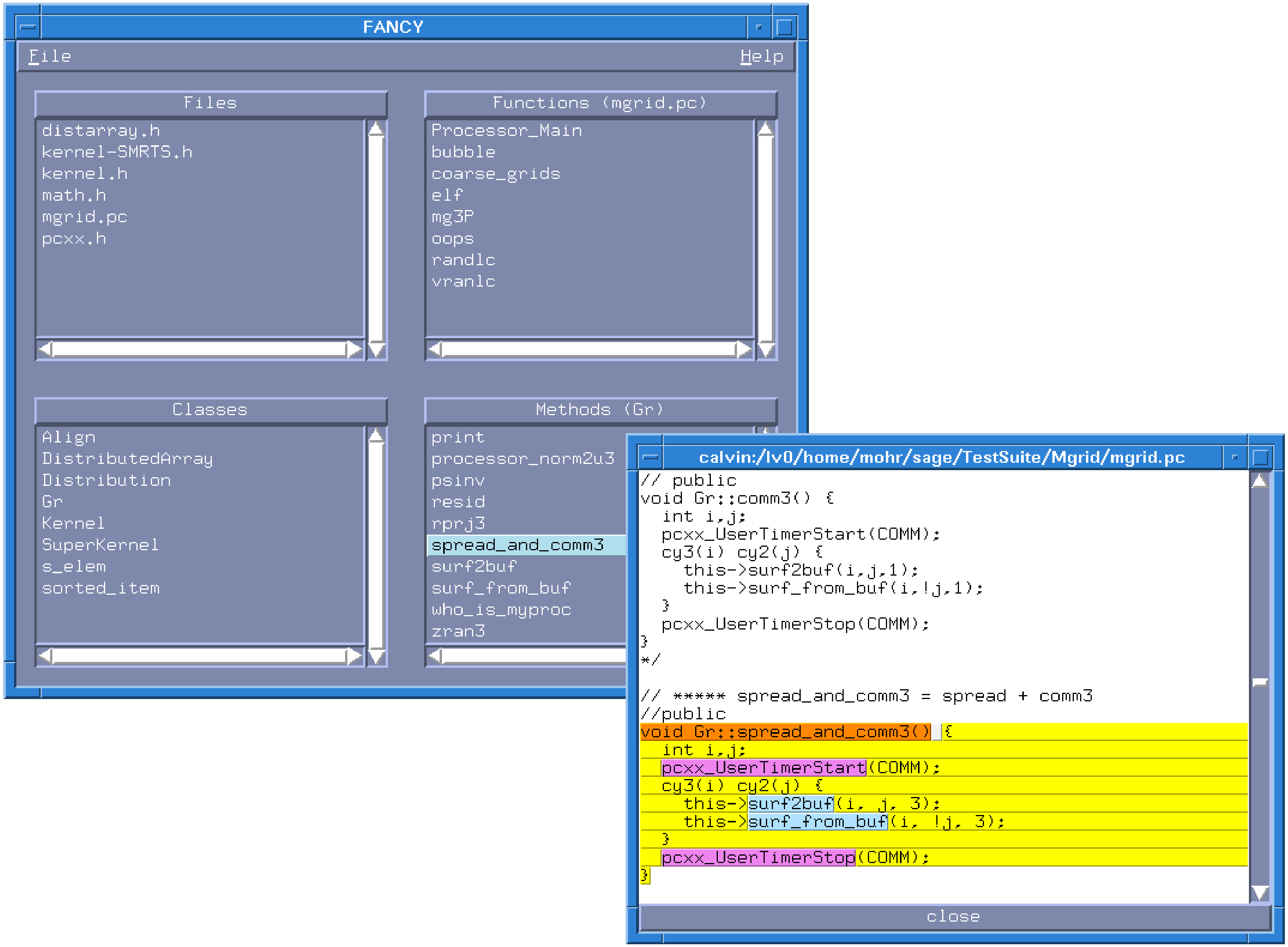
TAU / COSY
Main Control Panel / Compile and Execute Manager

τ

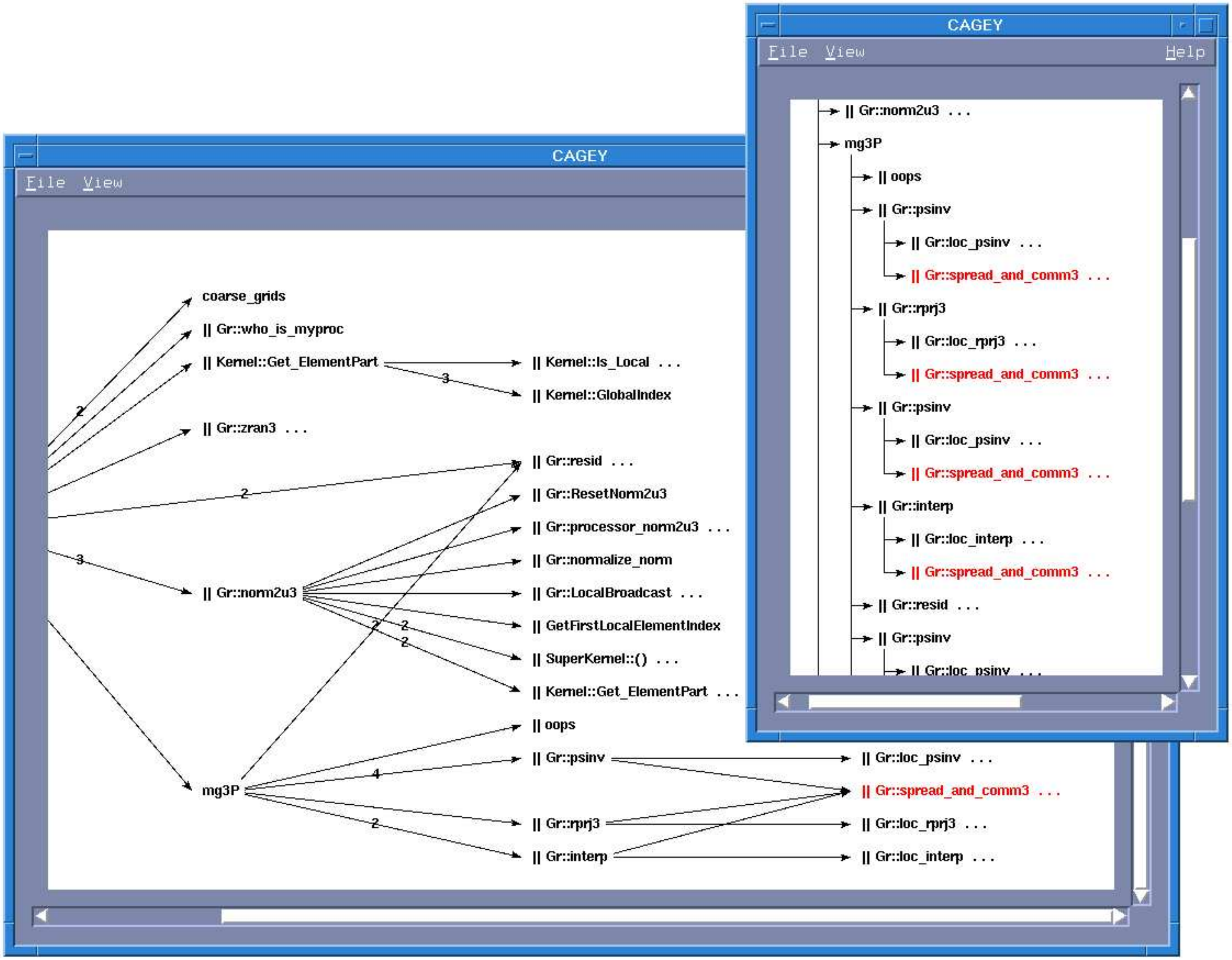


FANCY
Source Code Browser

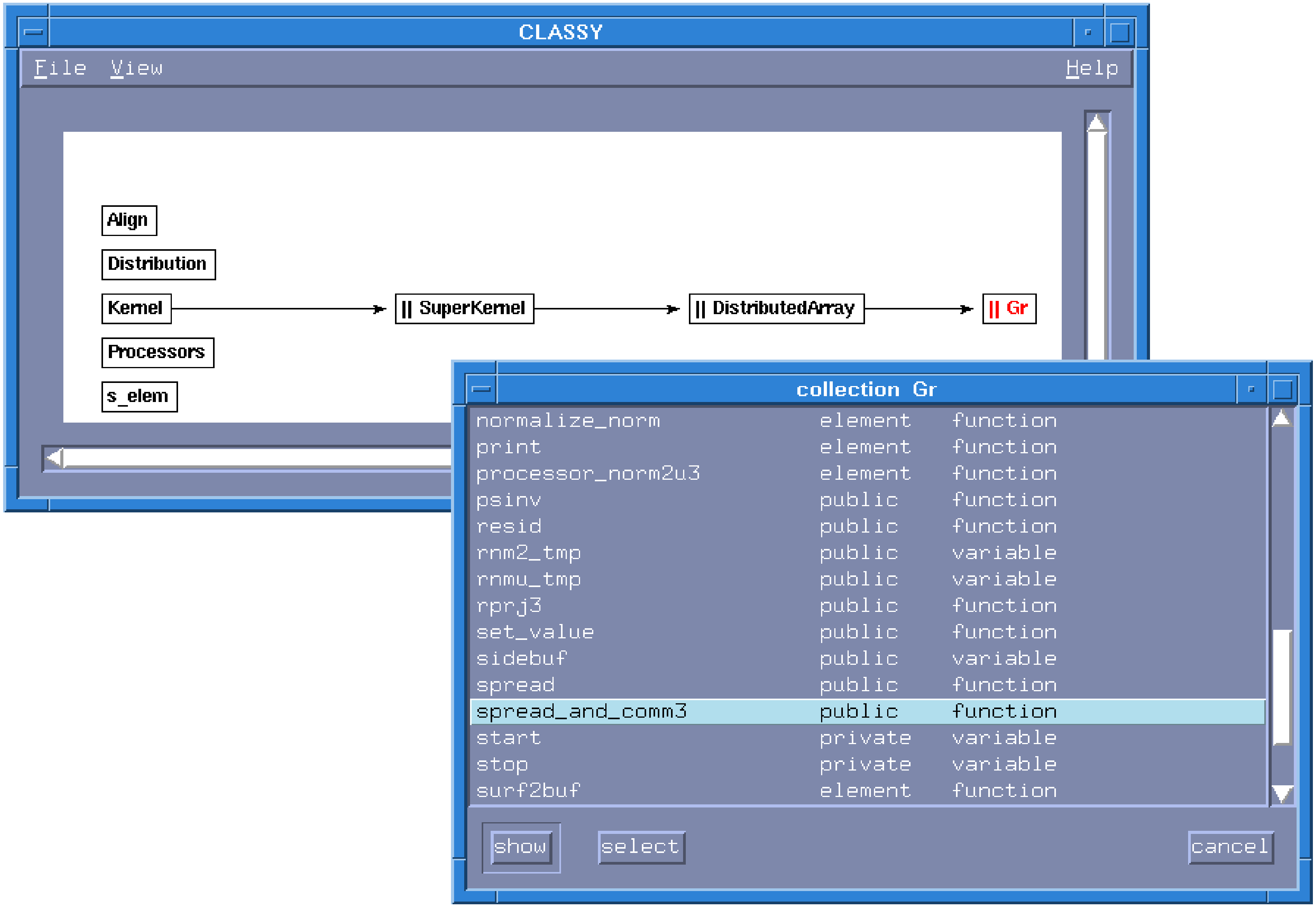
τ



CAGEY
Static Callgraph Browser



CLASSY
Class Hierarchy Browser



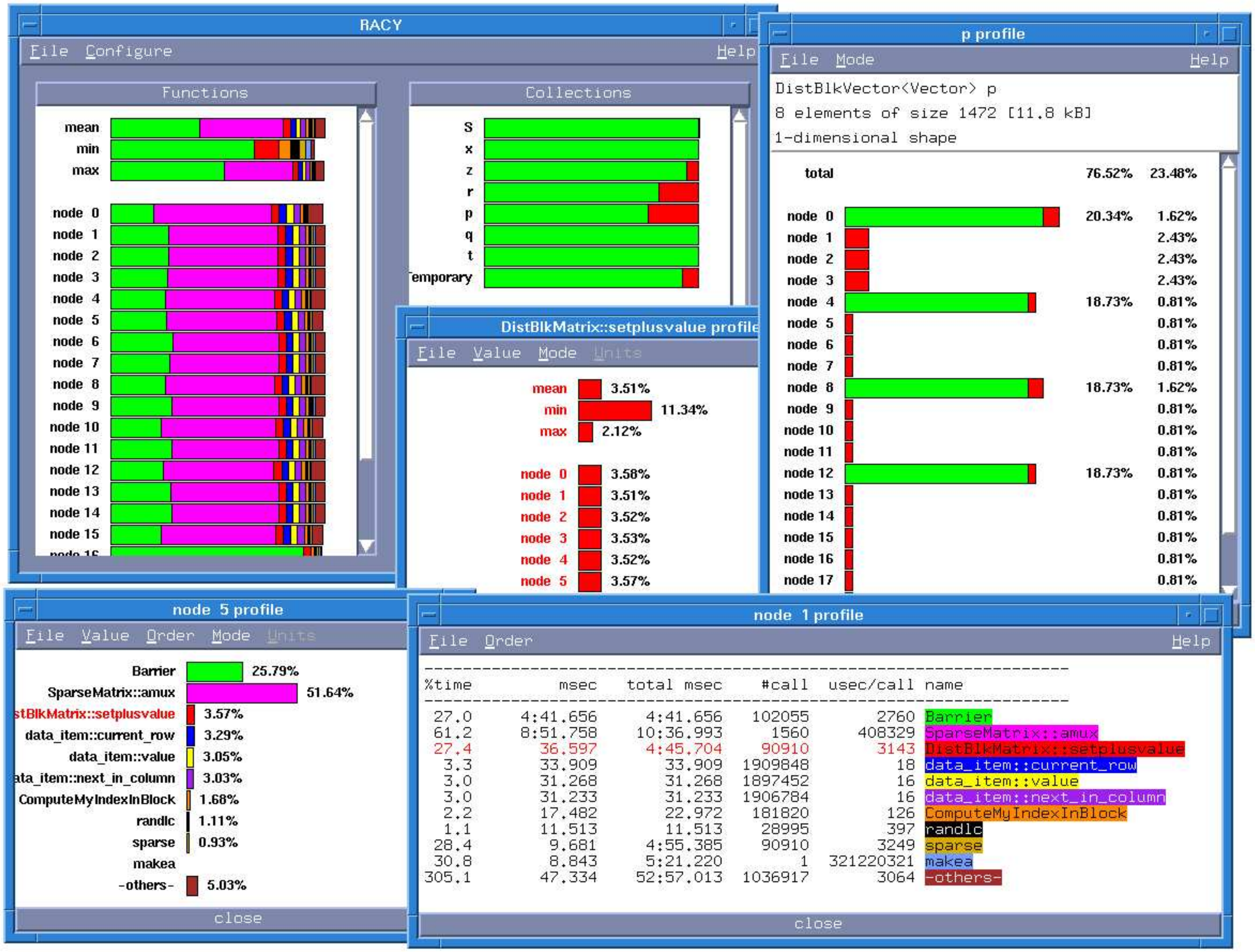
PPROF / RACY
a Parallel Program Profiler

τ

- uses **instr** tool to dynamically instrument
  - entry and exit of
  - functions, class member functions, constructors, and destructors
  - in the user application and the pC++ kernel
- uses “profile-instrumented” runtime system
- data gathered for each profiled function:
  - time spent in function including and excluding its children
  - number of calls
- data gathered for each pC++ collection:
  - number of local and remote accesses per node
- profile data is stored in one file per node in a **machine-independent** format
- **pprof**: prints ASCII profile report (like UNIX’s prof)
- **racy**: graphical profile data browser

RACY
a graphical Profile Data Browser

τ



❑ Example pprof Output

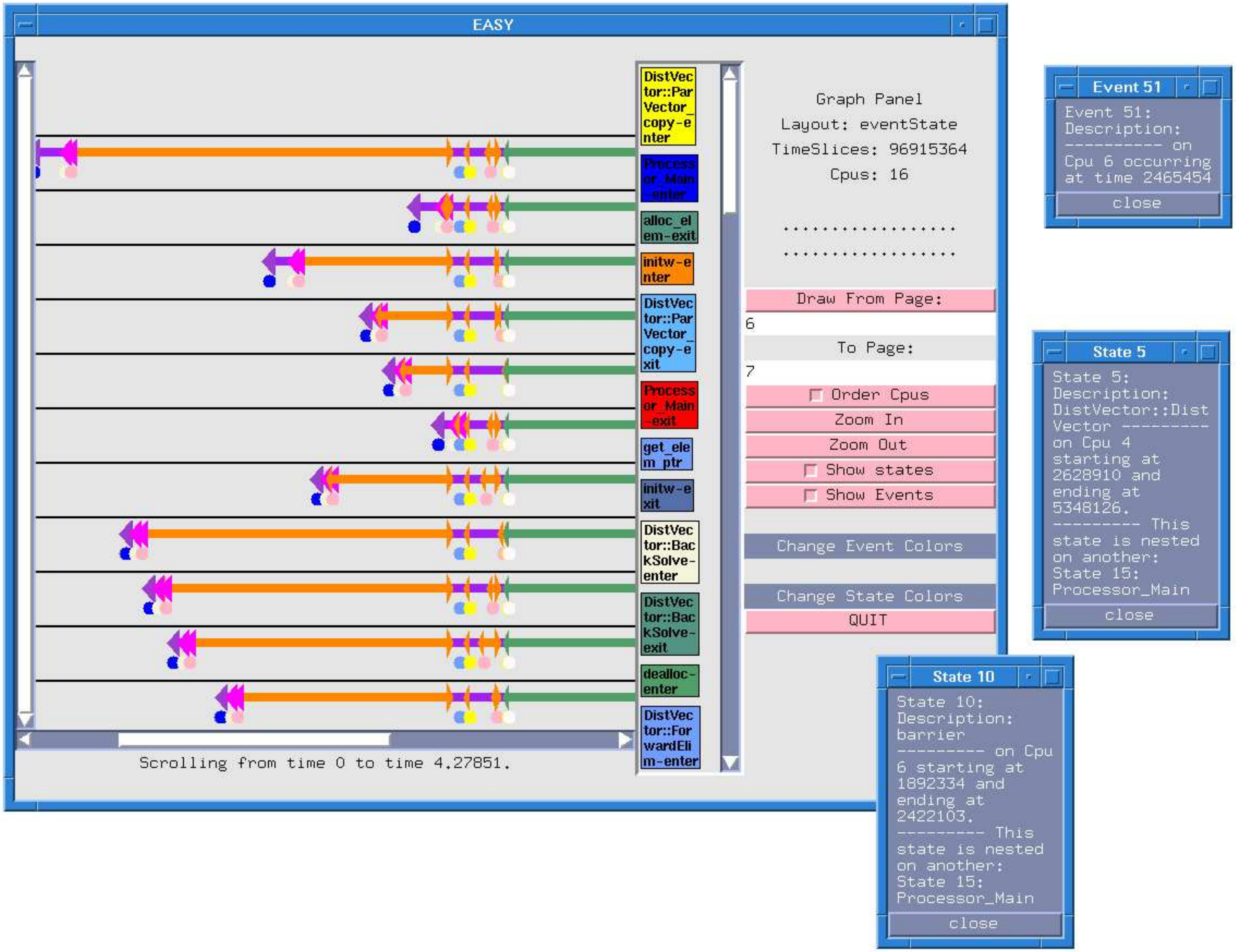
```
NODE 0:
---------------------------------------------------------------
%time   msec total msec  #call usec/call name
---------------------------------------------------------------
100.0 15.761     28.869      1  28869157 Processor_Main
 14.5  4.166      4.172      1   4172459 initw
 12.2  3.512      3.512 560567         6 Vector::get
  6.9      9      2.004      2   1002201 DistVector::DistVector
  6.6  1.919      1.919     44     43628 Barrier
  5.9  1.694      1.694     32     52958 Vector::allocData
  5.5    132      1.578      1   1578414 poisson_solve
  2.9      1        835     32     26117 ParSinTransform
  2.9    638        833     32     26055 Vector::SinTransform

   local   remote collection
accesses accesses num   name
   16619      496   0   F
    9789        5   1   U

// similar output from other 31 processors deleted

FUNCTION SUMMARY (mean):
---------------------------------------------------------------
%time   msec total msec  #call usec/call name
---------------------------------------------------------------
100.0 15.312     28.301      1  28301704 Processor_Main
 14.4  4.058      4.064      1   4064819 initw
 12.4  3.519      3.519 563042         6 Vector::get
  7.9  2.246      2.246     44     51065 Barrier
  7.1      9      2.004      2   1002249 DistVector::DistVector
  5.6    116      1.578      1   1578486 poisson_solve
  4.8  1.348      1.348     32     42138 Vector::allocData
  3.0      1        845     32     26418 ParSinTransform
  3.0    640        843     32     26356 Vector::SinTransform

COLLECTION SUMMARY:
---------------------------------------------------------------
DistVector<Vector> F, collection #0
      513 elements of size 4216, 1-dimensional
      532996 local / 15841 remote accesses
DistVector<Vector> U, collection #1
      513 elements of size 4216, 1-dimensional
      282720 local /   423 remote accesses
```

EASY
an Event Trace Browser

τ

- The inspiration for this tool comes from Upshot, and takes many ideas and direction from that tool.
- Implemented using Tcl/Tk and [incr Tcl]
- A graphical trace analysis tool used to visualize processor states and events from a trace file generated on a multiprocessor architecture.
- A simple format (alog) allows users to define
  - events
  - states
- The states and events are represented graphically on a scrollable canvas. Details for states or events can be retrieved by selecting the interesting graphical object with the mouse. Selection of a particular object causes a window to open giving name, processor, and relevant time(s) of the event or state. In addition, events can display a parameter value if specified in the event format.

EASY
an Event Trace and State Browser



BREEZY
a Barrier Breakpoint Debugger

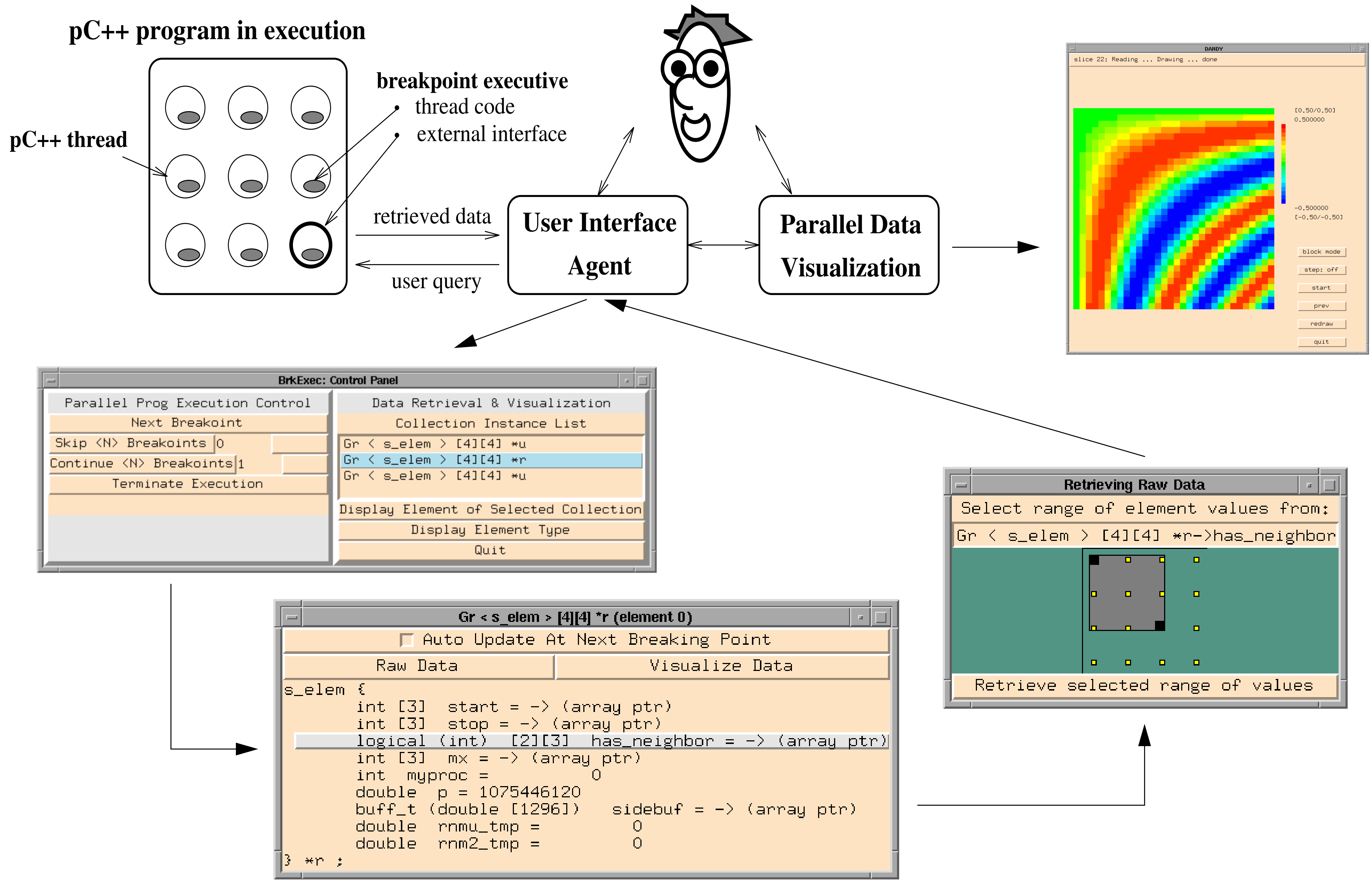
τ

- Goals
  - parallel collection data access during program execution
  - display data values at source code level
  - enable data visualization / animation
  - control parallel program from remote workstation
  - portability

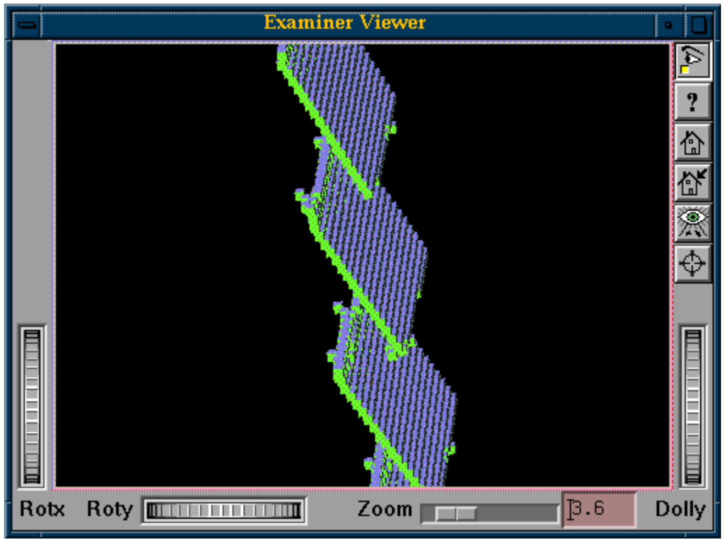
- Implementation
  - access to collection data at breakpoints (barriers)
  - user program instrumentation accomplished with Sage++ and data access implemented in pC++
  - user interface communicates with parallel pC++ program using TCP/IP socket connection
  - spawns separate processes to handle visualization
  - user interface implemented in Tcl/Tk
  - example visualizations implemented using OpenGL

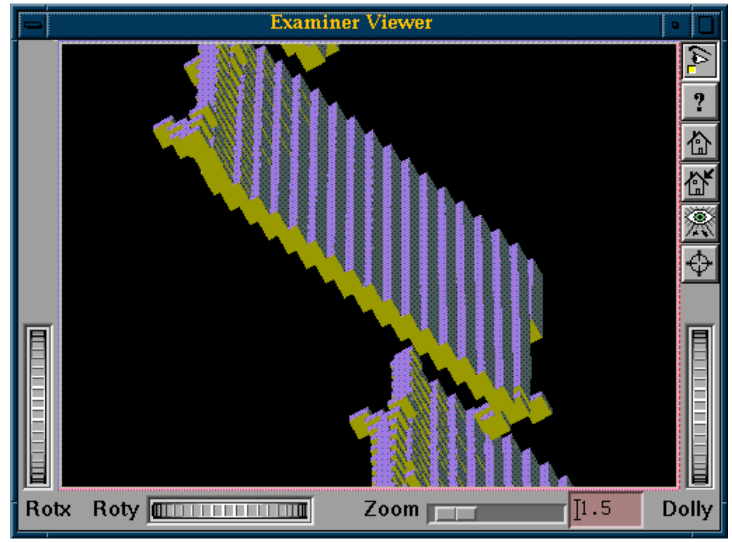
- user interface allows
  - step / step # / continue / exit
  - show all / select currently active collections
  - view / visualize collection data values

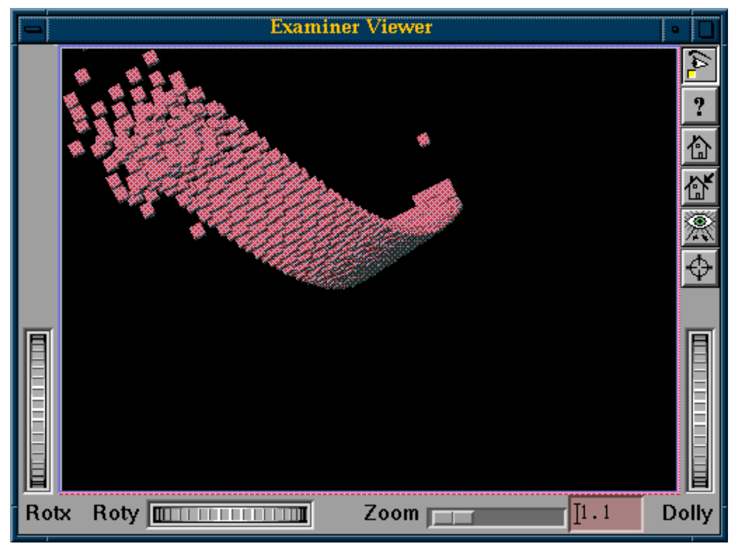
BREEZY
Barrier Breakpoint Debugger



BREEZY
Example Visualizations







Portable C++ Profiling

- general approach:
  - introduce `Profiler` class:

```
class Profiler {
  char *name;
public:
  Profiler(char *n) {name=n; on_enter(n);}
  ~Profiler() {on_exit(name);}
}
```

  - instrument global functions and class methods:

```
void foo() {
  Profiler _prof("foo"); //Profiler var.
  //body of foo
  ...
}
```

- advantages:
  - **efficient**: C++ compiler does the “hard” word
  - **portable**: instrumentation at the language level
  - **flexible**: different implementations possible by simply changing `on_enter` / `on_exit` code

- currently implemented:
  - direct profiling
  - trace-based profiling

INSTR
a Source Code Instrumentor

τ

- for C++ and C++ based languages

- currently supporting instrumentation of
  - method/function entry and exit
  - pC++ collection data access

- selective instrumentation through command file:

  syntax: `<command> <object> <ident>`

  - `<command>` := exclude | include
  - `<object>` := file | class | function
  - `<ident>` := name | regular expression

  examples:

  - ```
    # don't instrument members of class Vector
    exclude class Vector
    ```
  - ```
    # don't instrument operators (functions
    # whose names don't start with a letter)
    exclude function <^[^A-Za-z]>
    ```

- implementation based on Sage++

pC++ Default Instrumentation

- kernel
  - method + function profiling
    (dynamically through instrumentor)
  - element access tracing (local/remote)

- runtime system
  - runtime system function profiling
  - runtime system variables tracing
  - timer calls

- user method + function profiling
  (dynamically through instrumentor)

Portable Software EventTracing Library (Basics)

- basic features
  - large event buffer per node
  - establish / use global clock
  - trace dump to file off-line
  - efficient and fast through
    + keeping it simple and small
    + compact binary trace format

- conditional tracing based on event classes
  - kernel
  - basic runtime
  - (parallel) runtime
  - timers
  - tracing
  - profiling
  - user classes

- operational prototype for
  - UNIX workstations
  - TMC CM-5
  - Intel Paragon
  - IBM SP-1
  - PVM clusters
  - Sequent Symmetry
  - Kendall Square KSR-1 / KSR-2
  - BBN TC2000
  - SGI Oynx + Challenge

Portable Software Event Tracing Library (Implementation)

- standardized function interface

```
pcxx_EvInit(char filename[]);
pcxx_Event(long ev, long parameter);
pcxx_LongEvent(long ev, int len,
               char buf[]);
pcxx_EvClose();
pcxx_EvFlush();

PCXX_EVENT(trcclass, ev, parameter)
```

- recorded information

```
struct event_record {
   long int   eventid;
   short int  nodeid;
   short int  threadid;
   long int   timestamp; /*us*/
   long int   parameter;
}
```

- conditional instrumentation

```
# ifdef TRACING_ON
     extern void pcxx_Event(int, long);
# else
#    define pcxx_Event(id,par)
# endif
```

Event Tracing Support Tools

- **pcxx_merge**:
  - merges node traces to global system trace according to timestamps
  - establishes global timescale, if target system doesn't have global clock

- **pcxx_convert**: trace format converter
  - generic ASCII
  - alog
  - SDDF

Event Trace Tools

- **easy**
  - τ event trace browser for alog format
- **SIMPLE** (University of Erlangen, Germany)
  - trace format independent event trace analysis environment
- **Upshot** (Argonne National Laboratory)
  - simple+portable X11 event trace browser for alog format
- **Pablo** (University of Illinois)
  - sophisticated event trace analysis environment based on SDDF format