Tool support for inspecting the code quality of HPC apps

Thomas Panas, Dan Quinlan
Center for Applied Scientific Computing (CASC)
Lawrence Livermore National Laboratory
Richard Vuduc
LLNL / Georgia Tech

3rd Workshop on Software Engineering for HPC Apps
May 26, 2007

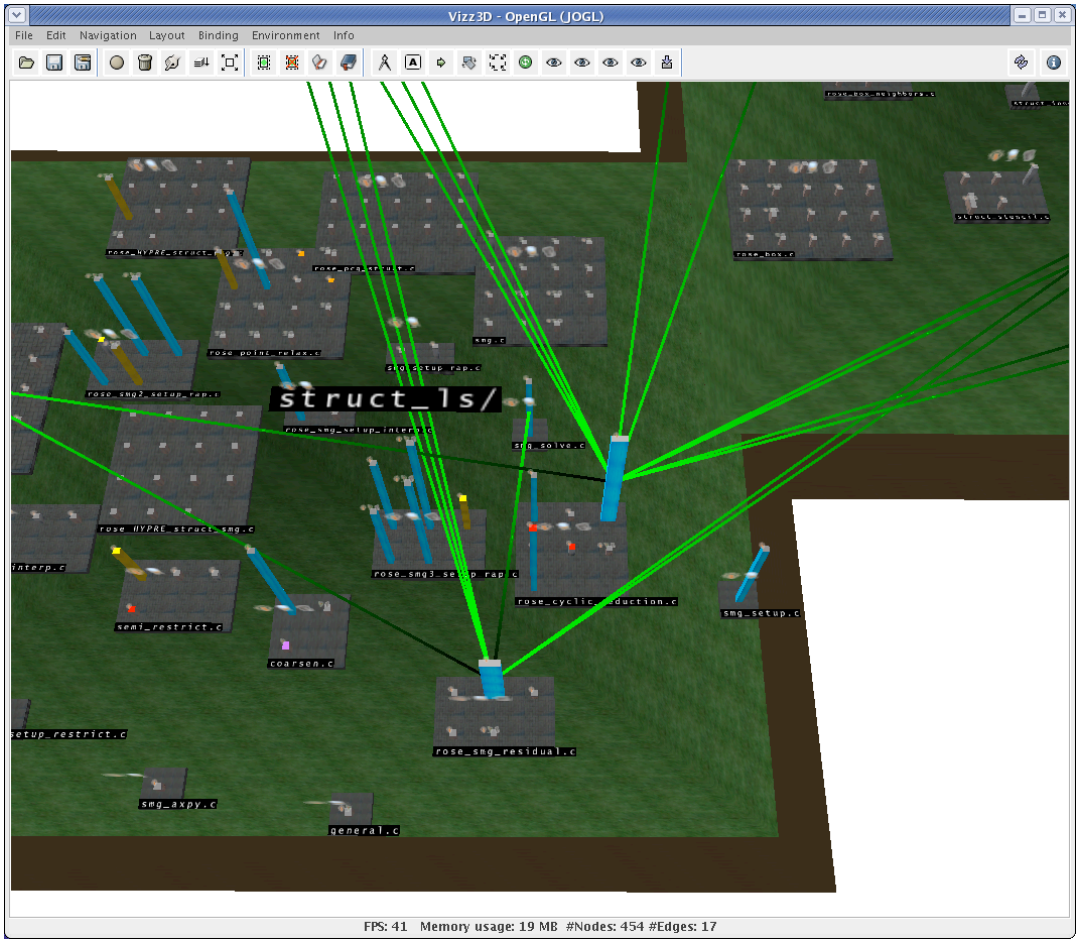
Research motivation and goals

- Context: HPC development is often *ad hoc*, not formal
  - Unknown requirements at project start
  - No training in (or time for) formal processes
  - Developer or team may be sole customer initially
  - Correctness and performance first, maintenance later
  - Whether maintenance considered or not, codes have long lifetimes
- Goal: Communicate code quality unobtrusively
  - Provide on-demand feedback to dev. team
  - Integrate naturally into dev. process
  - Use interactive visualization when appropriate
  - Incorporate team-specific coding guidelines/requirements

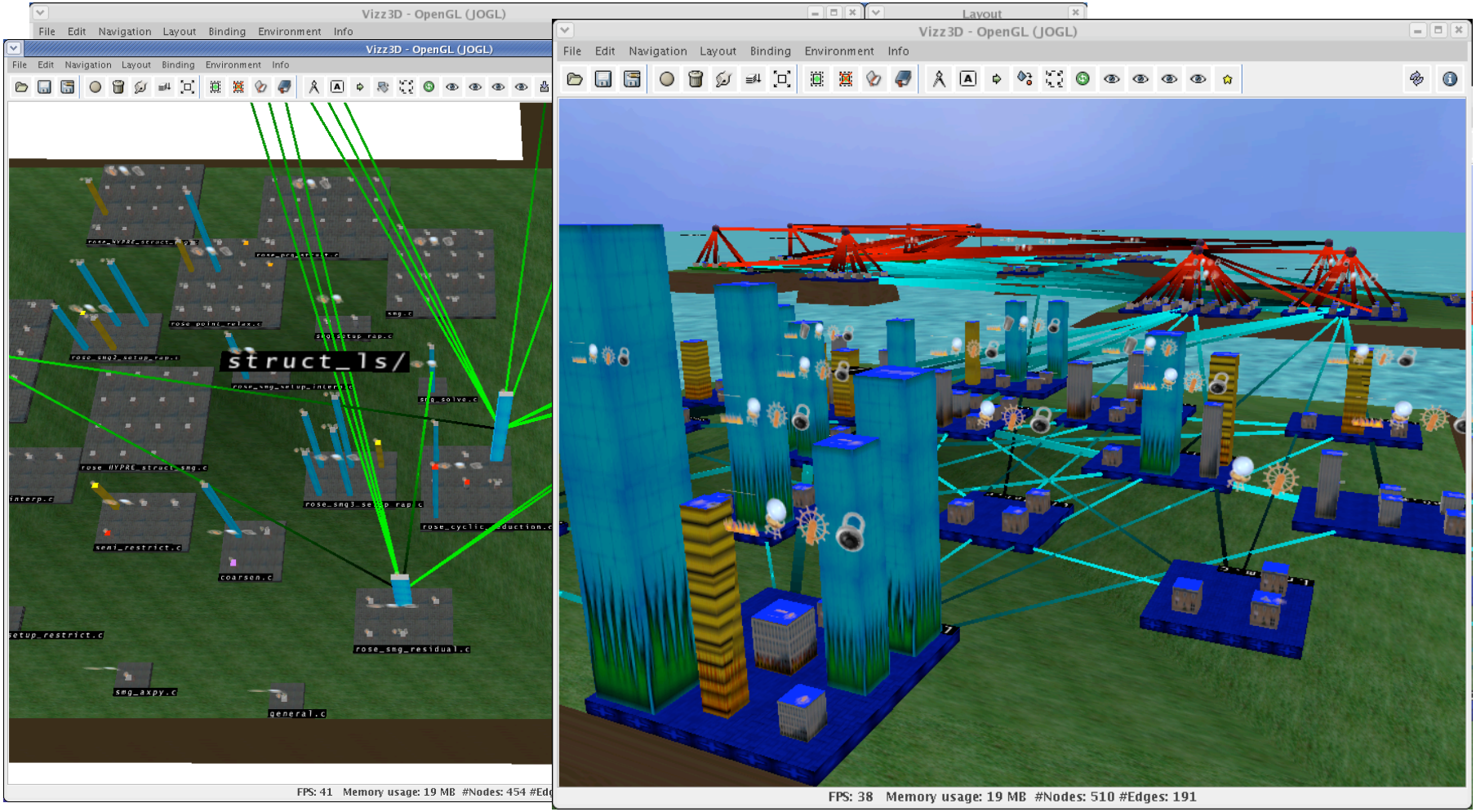
ROSE-VA:
A code quality inspection tool prototype

- Reports code quality metrics
  - Maintainability
  - Component reusability
  - Software metrics
- Visualizes judiciously
  - File or module coherence as static images
  - Software architecture in interactive views
- Implemented using extensible tools
  - ROSE: Source-to-source compiler for C, C++ (Fortran in progress)
  - VizzAnalyzer / Vizz3D: Interactive software viz framework
- This talk: Overview, status, and example (SMG2000)

Architecture-level viz of call graph (SMG2k)



Architecture-level viz of call graph (SMG2k)



Categories of code quality information

- Simple statistics
  - *E.g.*, lines of code, global variable count, no. of security violations
  - Context-insensitive, so displayed as text
- Coherence reports
  - *E.g.*, global variables and files in which they appear
  - Shows simple contextual relationships
  - Shown as text or 2-D static images
- Architecture-level information overviews
  - *E.g.*, module dependencies
  - Visualized interactively, using physical metaphors

Sample "application": SMG2k

- SMG2000: A semicoarsening multigrid benchmark
  - C program from ASC Purple Benchmark suite
  - ~46 KLOC in 5 modules and ~70 files
- Small but interesting example of "one-shot" app
  - Intended for single-use
  - Contains subset of *hypre* library, so should be well-structured

Simple statistics

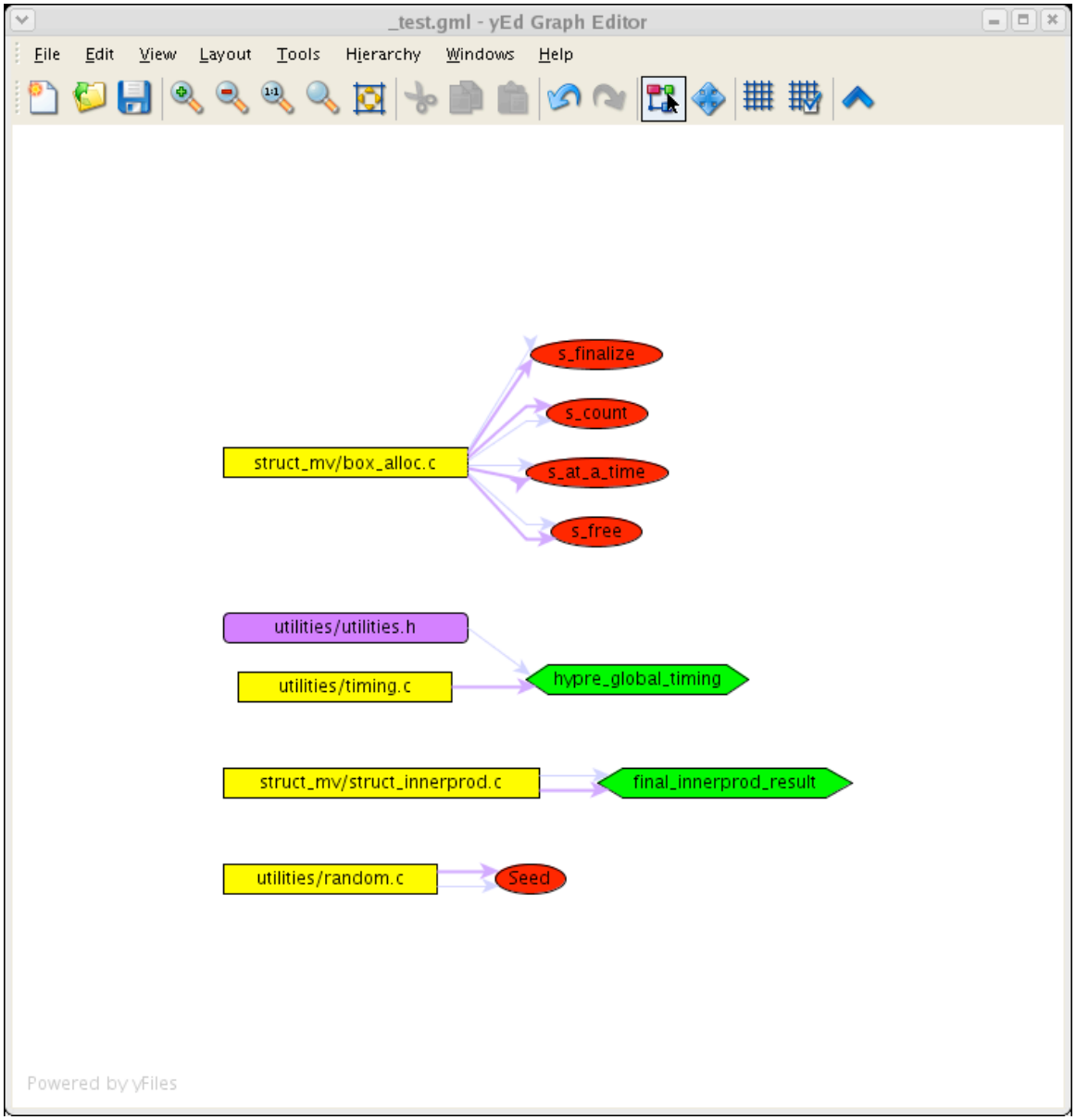
- Global statistics
  - Lines of code; abstract syntax tree (AST) statistics
  - No. of files, types of files (*e.g.*, source *vs.* header)
  - No. and type of classes
  - No. of function calls (static)
- Simple analysis summaries
  - "Security" or API violations (*e.g.*, sprintf, new-delete)
  - Cyclomatic complexity per function or file
  - Global variable analysis
  - Cyclic dependencies
  - Naming convention violations
  - Degree of documentation
  - Arithmetic complexity

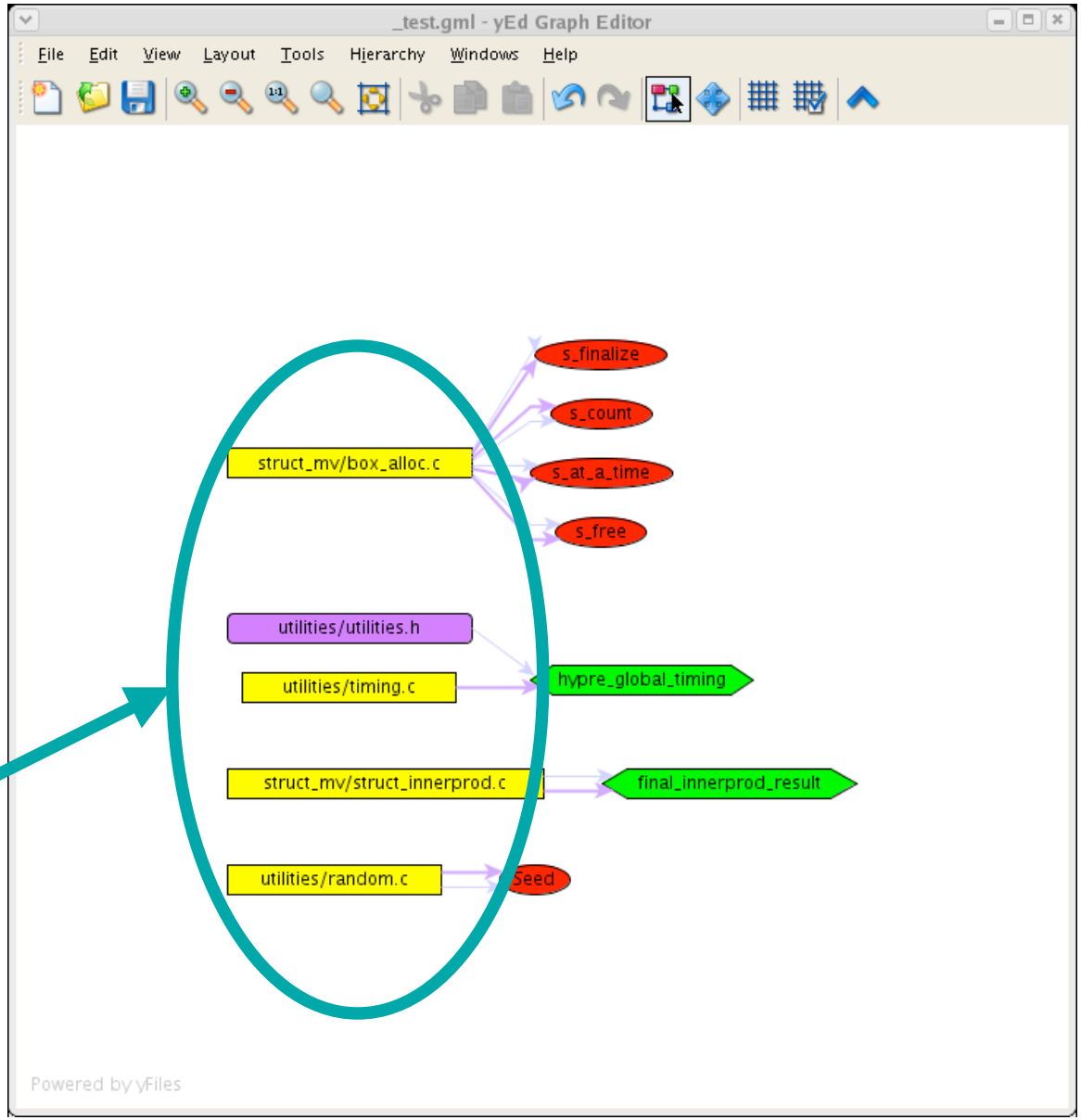
Simple analysis results for SMG2k

Analysis	Threshold	# of violations
Security / API misuse	0	**4**
Cyclomatic complexity per function	< 20	**20**
Global variables	0	**2**
LOC per function	< 200	**19**
Cyclic file dependencies	0	**1**
Naming convention violations	---	**53**

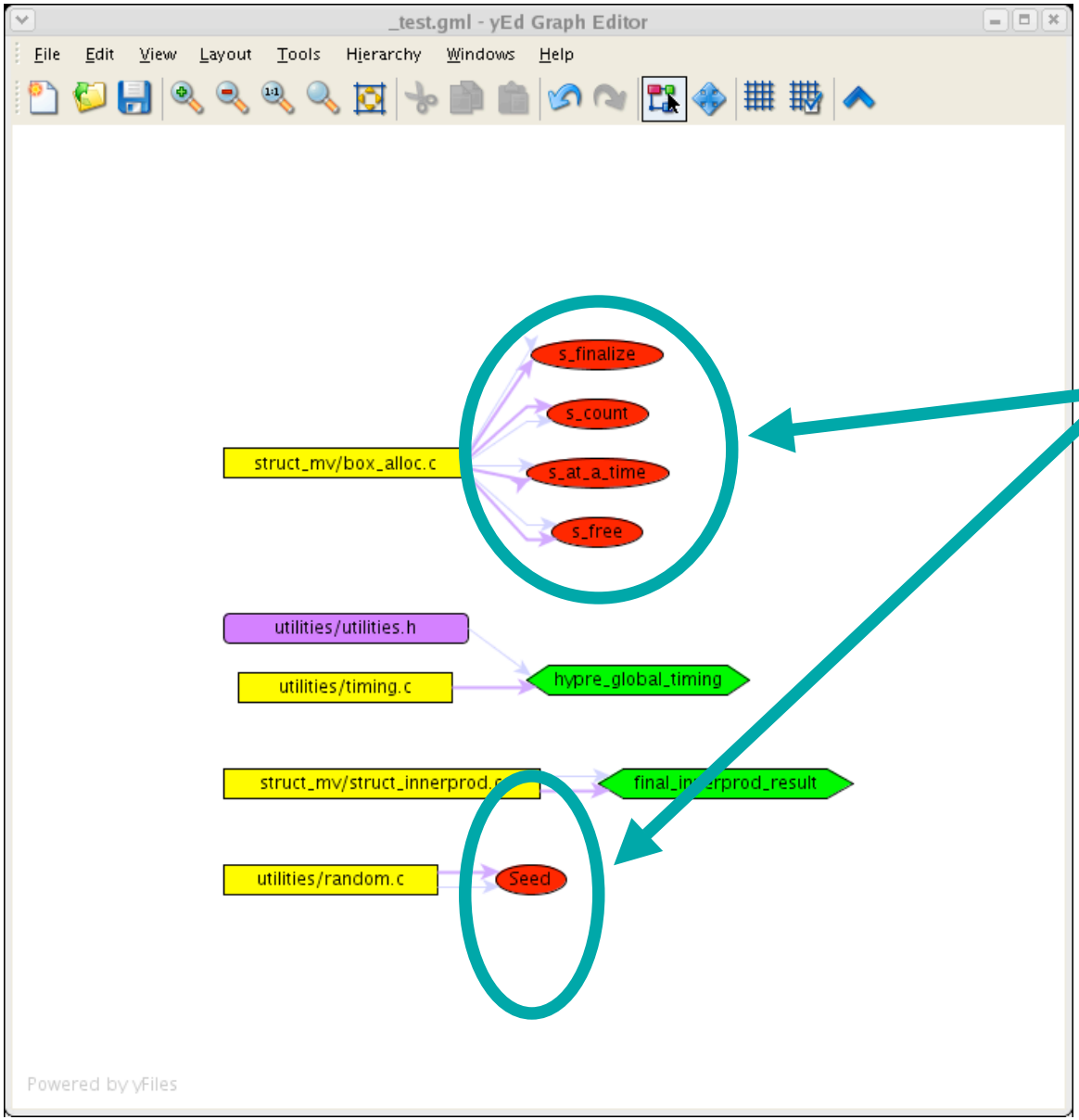
Coherence reporting

- Relationships between a few pieces of information, rendered in 2-D static images
- Example: Where are the global variables?
  - And can we eliminate them?
  - Relate files, global variable references, variable types (static vs. true global)

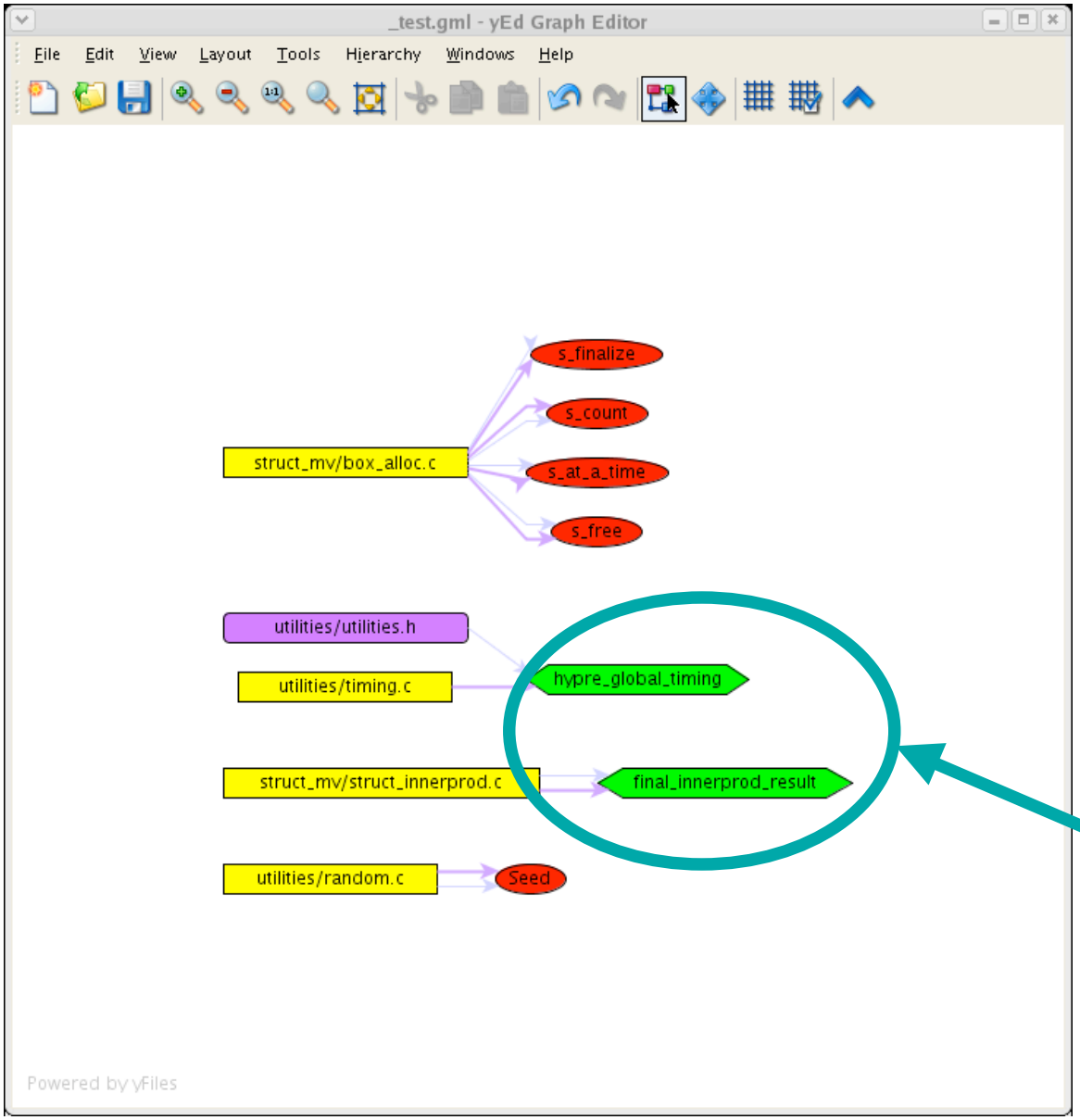

_test.gml - yEd Graph Editor
File
Edit
View
Layout
Tools
Hierarchy
Windows
Help
s_finalize
s_count
struct_mv/box_alloc.c
s_at_a_time
s_free
utilities/utilities.h
utilities/timing.c
hypre_global_timing
struct_mv/struct_innerprod.c
final_innerprod_result
utilities/random.c
Seed
Powered by yFiles


_test.gml - yEd Graph Editor
File
Edit
View
Layout
Tools
Hierarchy
Windows
Help
s_finalize
s_count
struct_mv/box_alloc.c
s_at_a_time
s_free
utilities/utilities.h
utilities/timing.c
hypre_global_timing
struct_mv/struct_innerprod.c
final_innerprod_result
utilities/random.c
Seed
Powered by yFiles

Files

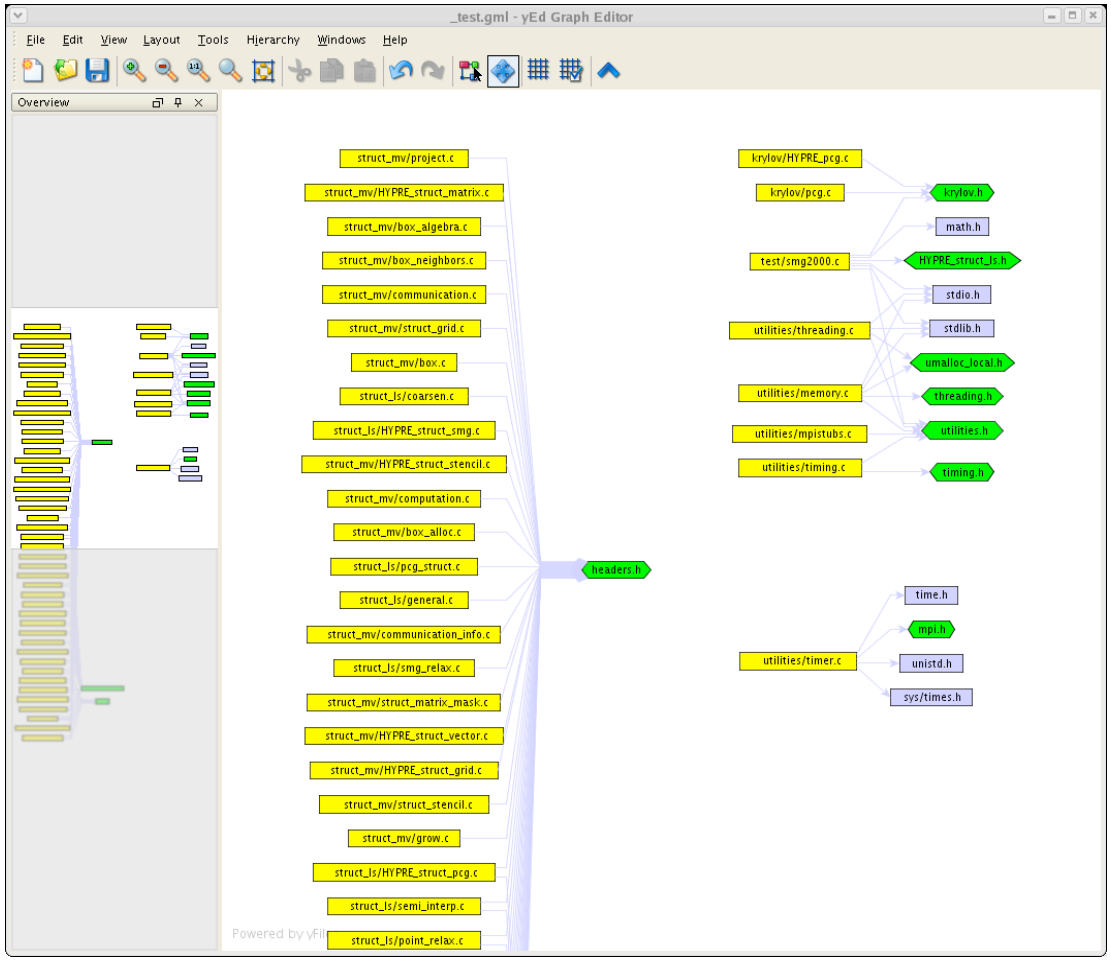


File-scoped global



"True" globals

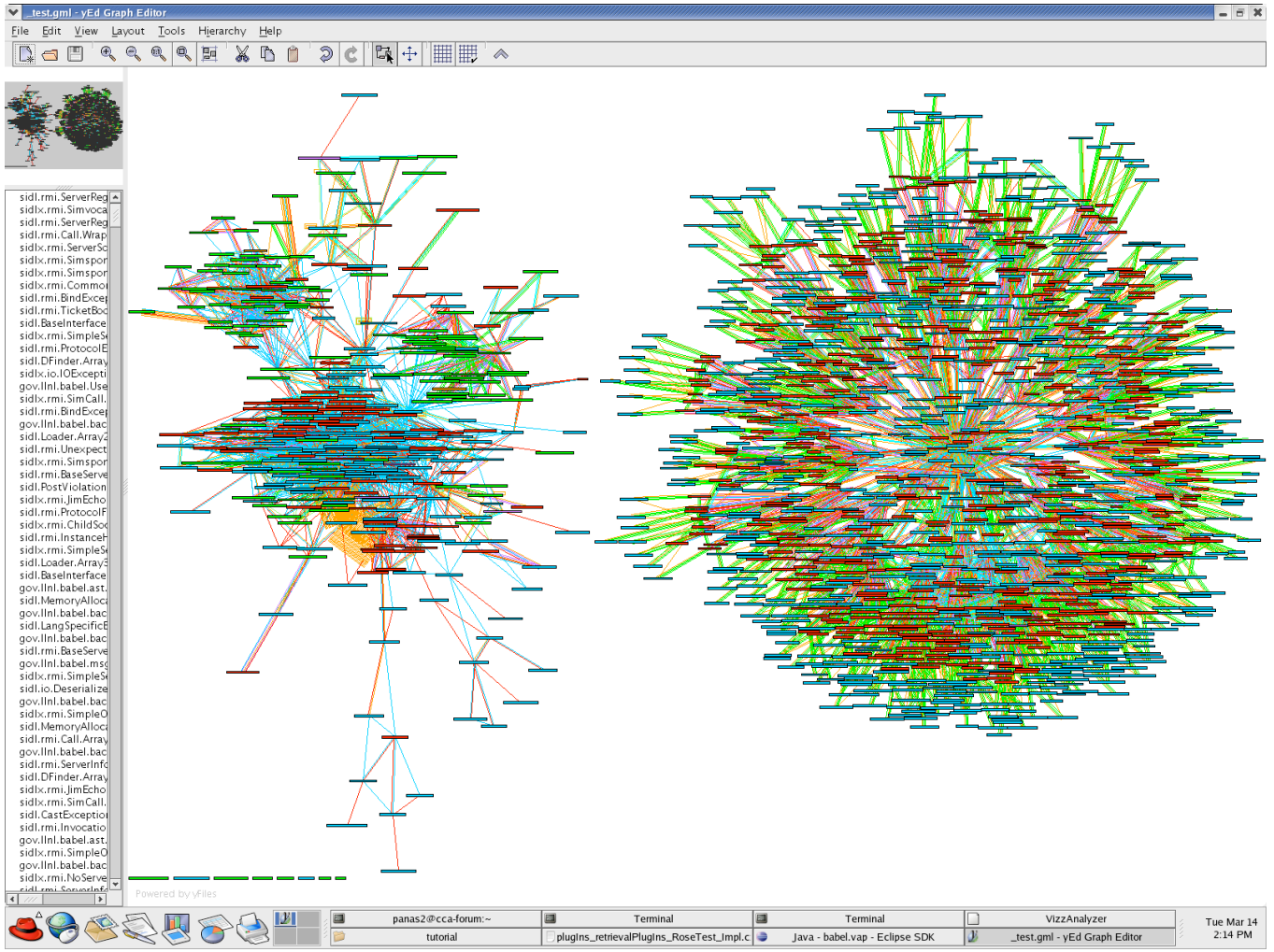
Example: Source and header dependencies (File-include graph)



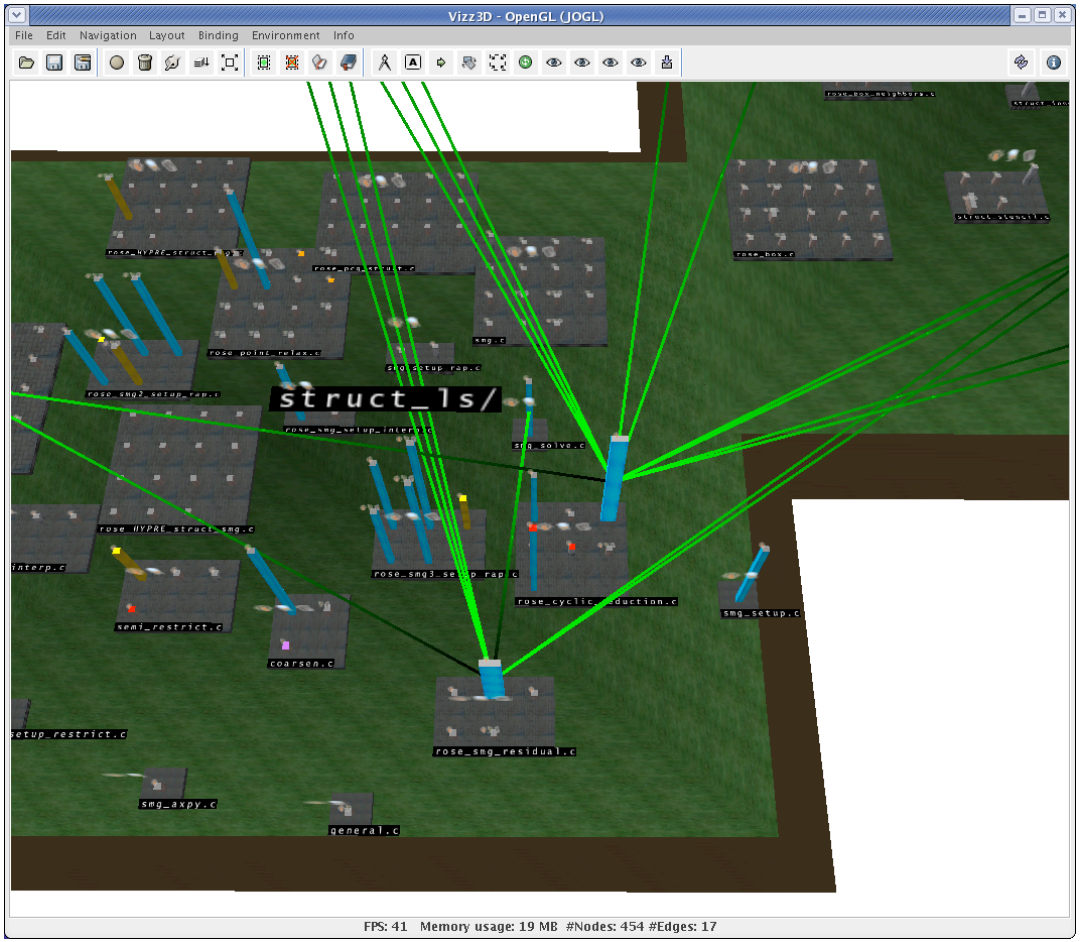
Architecture-level information overviews

- For viewing lots of information in context of software architecture
- Our approach: Interactive viz with “natural” metaphors
  - Interactive 3-D view of graphs
  - Islands (modules), cities (files), bulidings (functions)
- Example: “Where” is the code becoming complex?
  - Large functions and modules?
  - Cyclic module dependencies, based on call graph?

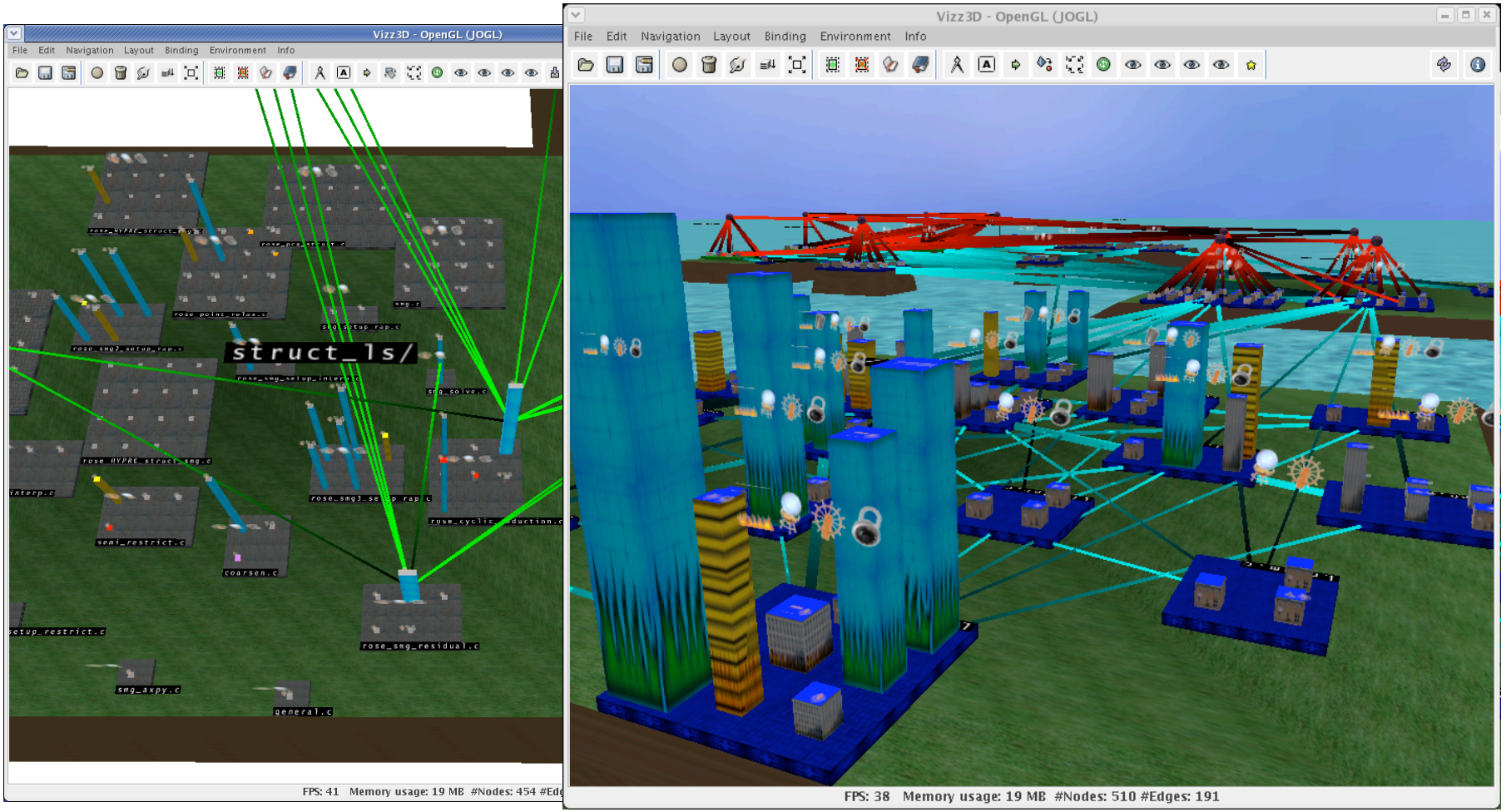
“Classical” call graph viz (SMG2K)



Architecture-level viz of call graph (SMG2K)

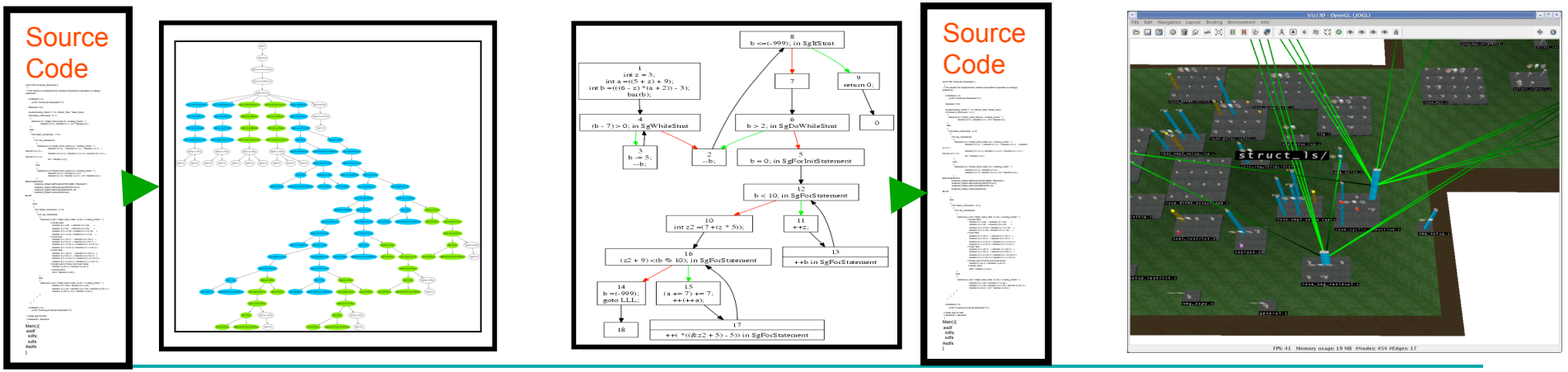


Architecture-level viz of call graph (SMG2K)

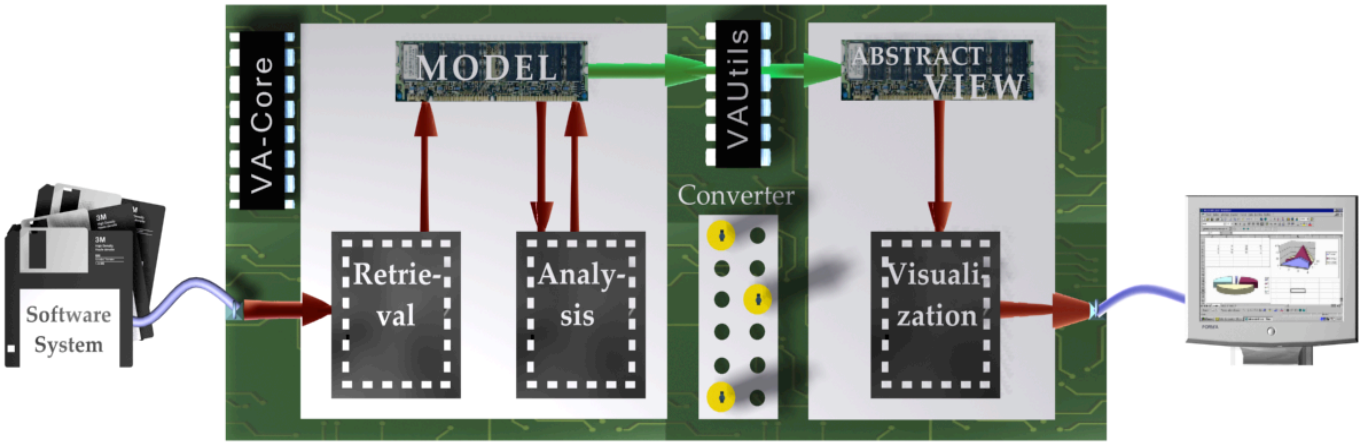


Implementation: ROSE-VA combines compiler & viz tools

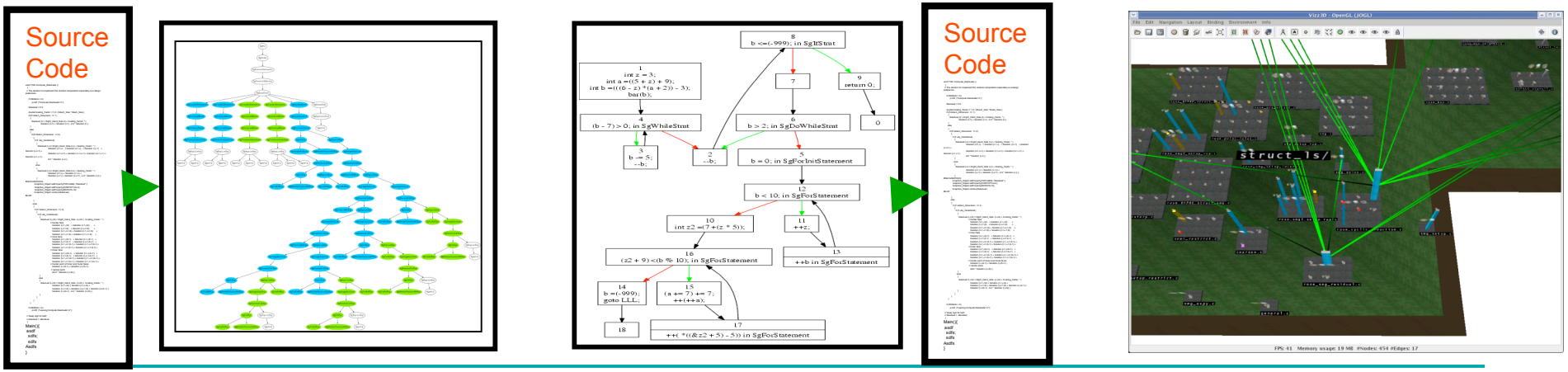
parse: ROSE **analyze: ROSE** **visualize: Vizz3D**



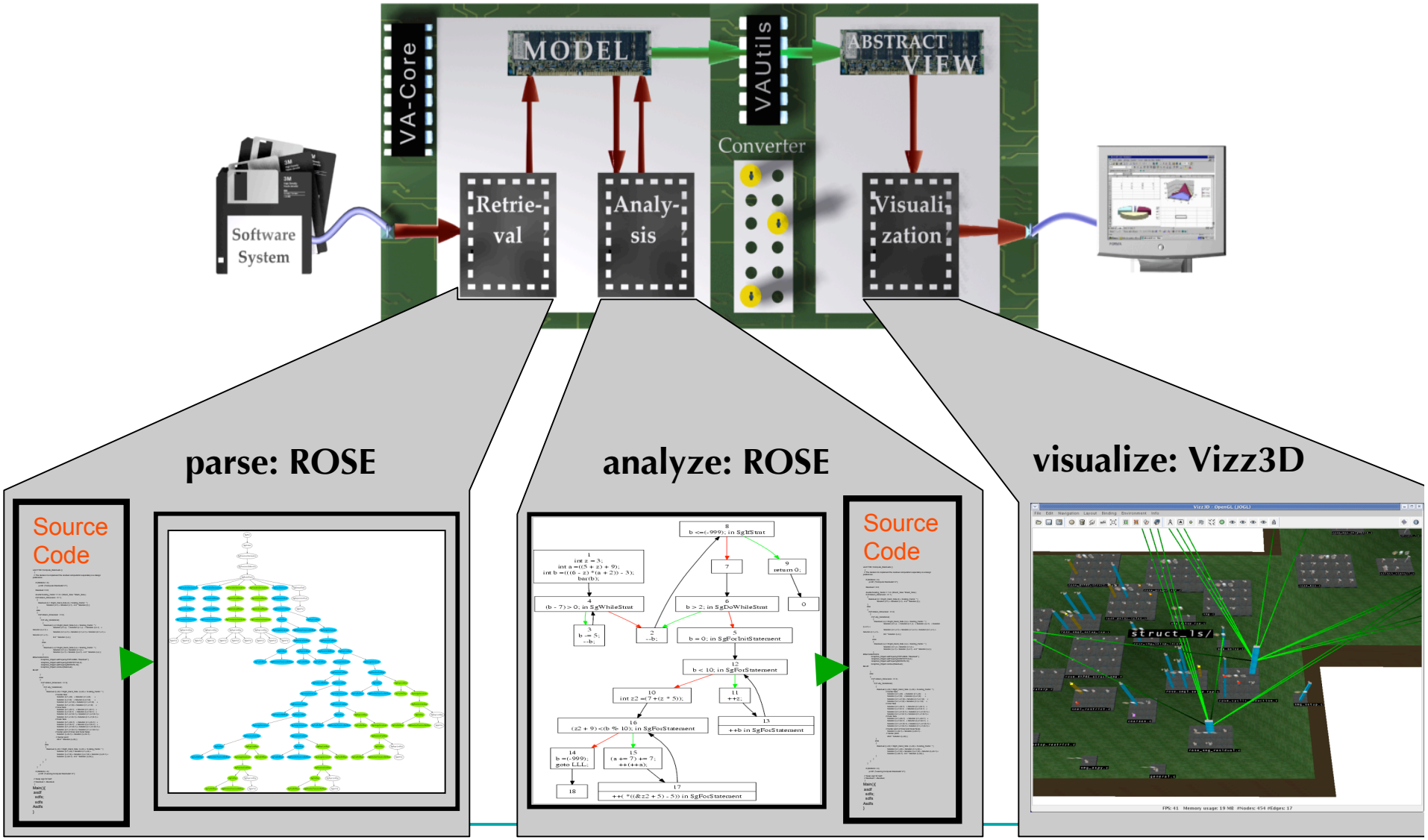
ROSE-VA combines compiler & viz tools



parse: ROSE **analyze: ROSE** **visualize: Vizz3D**



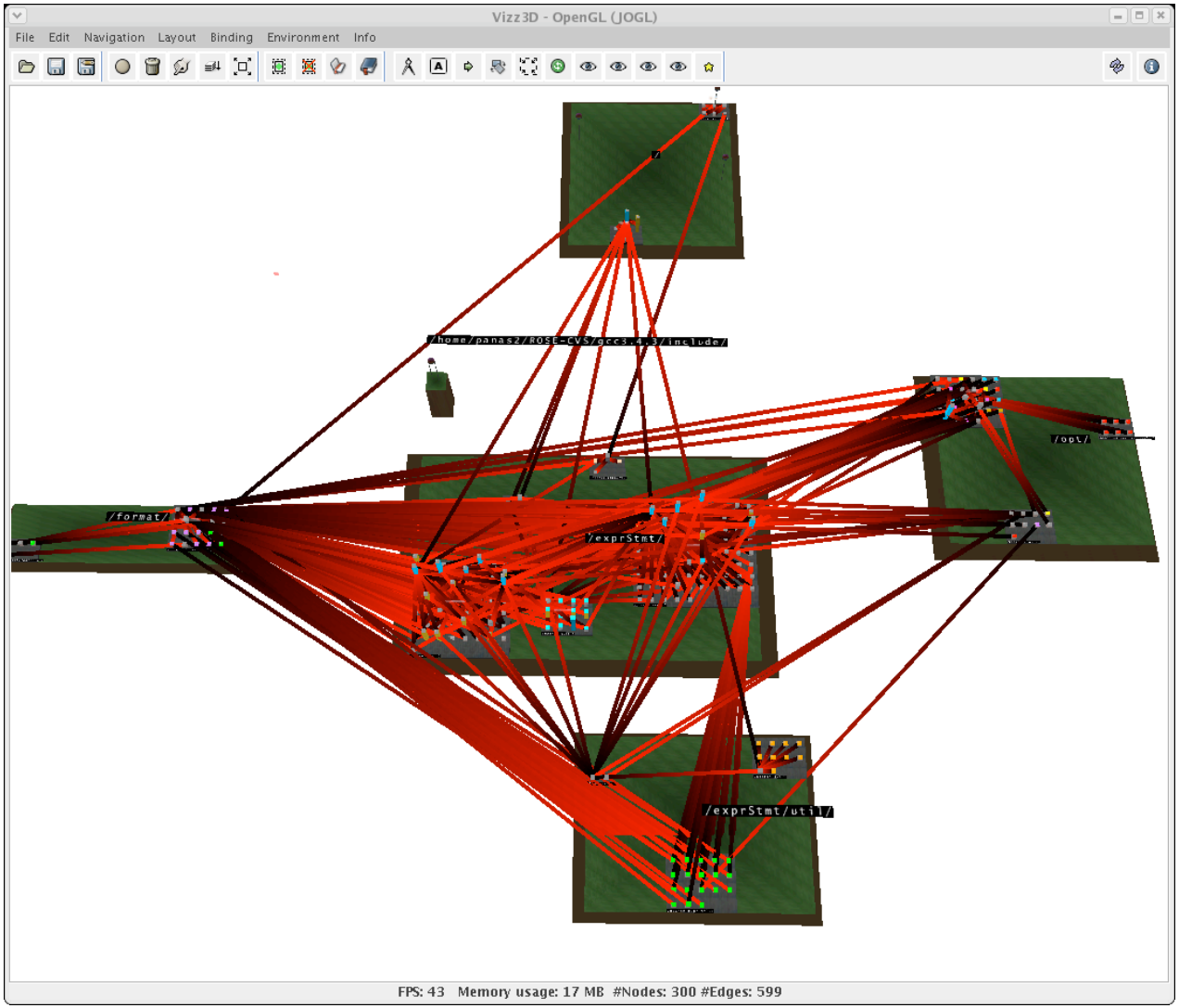
ROSE-VA combines compiler & viz tools



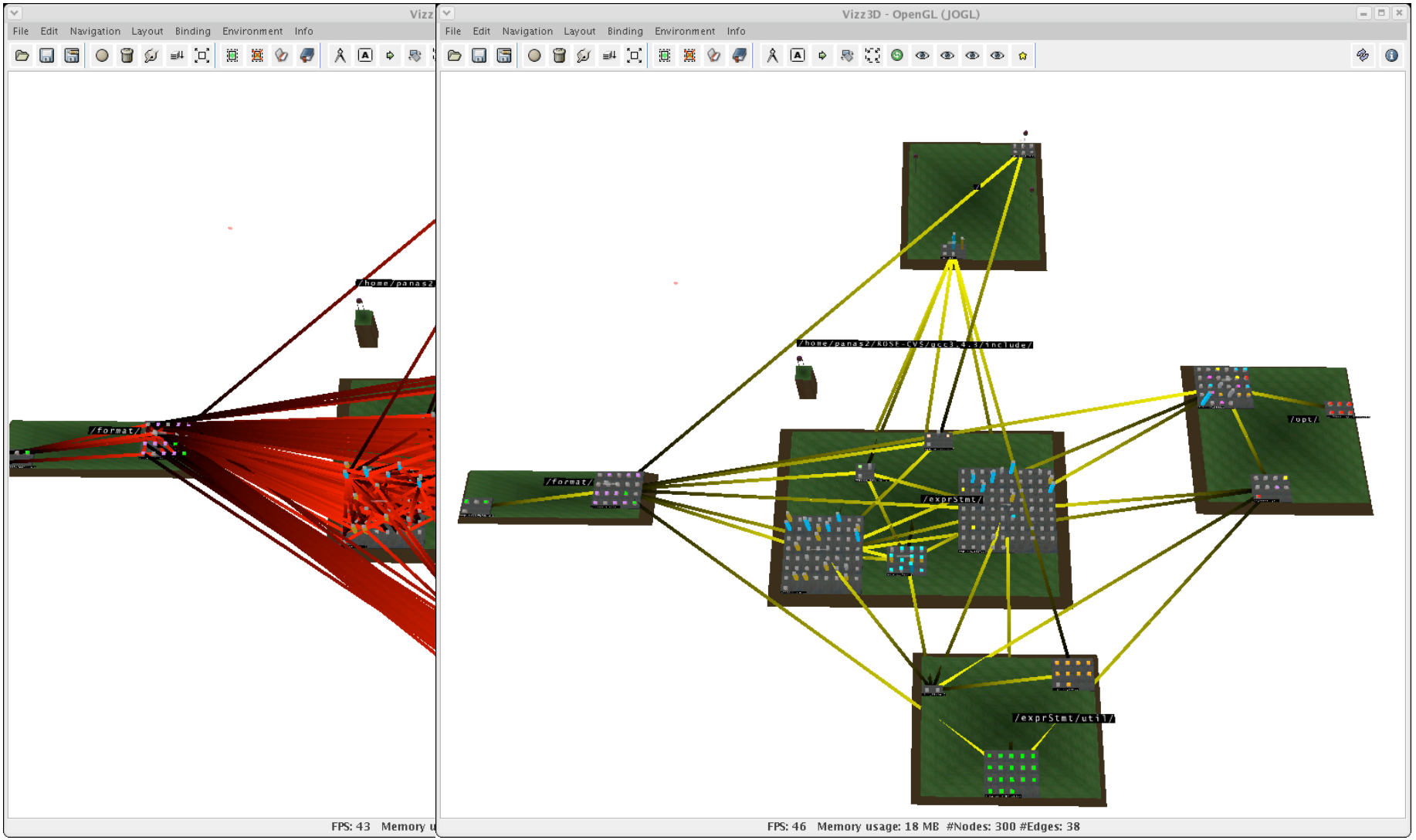
ROSE-VA combines compiler & viz tools

- ROSE: An infrastructure for building customized source-to-source tools
  - Full open-source compiler with basic analysis
  - Support for C, C++ (Fortran 90 in progress)
  - **Targets "non-compiler audience"**
- VizzAnalyzer / Vizz3D: Graph visualization framework
  - Software analysis + information visualization
  - Plug-in architecture
- Babel connects ROSE (C++) and VA (Java)

Re-engineering example: Refactoring ROSE



Re-engineering example: Refactoring ROSE



Summary and future work

- Prototype analysis and viz tool
  - Targets understanding, maintenance, and re-engineering tasks
  - Flexible, unobtrusive, developer-driven usage
  - HPC-specific metrics and analyses? Utility?
- Next steps: External evaluation + custom analyses
  - Applying to larger applications + specific coding guidelines
  - Specifying and finding patterns
  - Including dynamic information (HPCToolkit, Open|SpeedShop)
  - Mining and incorporating source-repository
  - Adding more metrics, *e.g.*, productivity (size, time), defect density

End

Demo?

Collaborators

- DOE Laboratories:
  - LLNL (A-Div (Kull), B-Div (ALE3D, Ares), Overture, Babel)
  - ANL (Paul Hovland)
  - ORNL
- DOE Research Programs:
  - PERC (SLAC, TSTT, C/C++ Optimization, UT, ANL, Dyninst Binary Rewriting)
- Collaborations:
  - Texas A&M (Lawrence Rauchwerger, Bjarne Stroustrup)
  - Rice University (Ken Kennedy, John Mellnor-Crummey)
  - IBM Haifa (Shmuel Ur)
  - Vienna University of Technology (Markus Schordan)
  - University of Tennessee (Jack Dongarra's group)
  - Cornell University (Sally McKee, Brian White)
  - Indiana University (Andrew Lumsdaine, Jeremiah Willcock)
  - University of California at Berkeley (UPC, Kathy Yelick)
  - University of Oslo (Hans, Andreas, Are)
  - University of Maryland (Jeff Hollingsworth, Chadd Williams)
  - Friedrich-Alexander-University Erlangen-Nuremberg (Markus Kowarschik, Nils Thurey)
  - University of Texas at Austin (Calvin Lin)
  - USCD (Scott Baden)
  - London Imperial College (Olav Beckman, Paul Kelly)
  - UC Davis (Su, Bishop)