Software Architecture Recovery from Build Processes

Bram ADAMS
Ghislain Hoffman Software Engineering Lab, INTEC, Ghent University
http://users.ugent.be/~badams

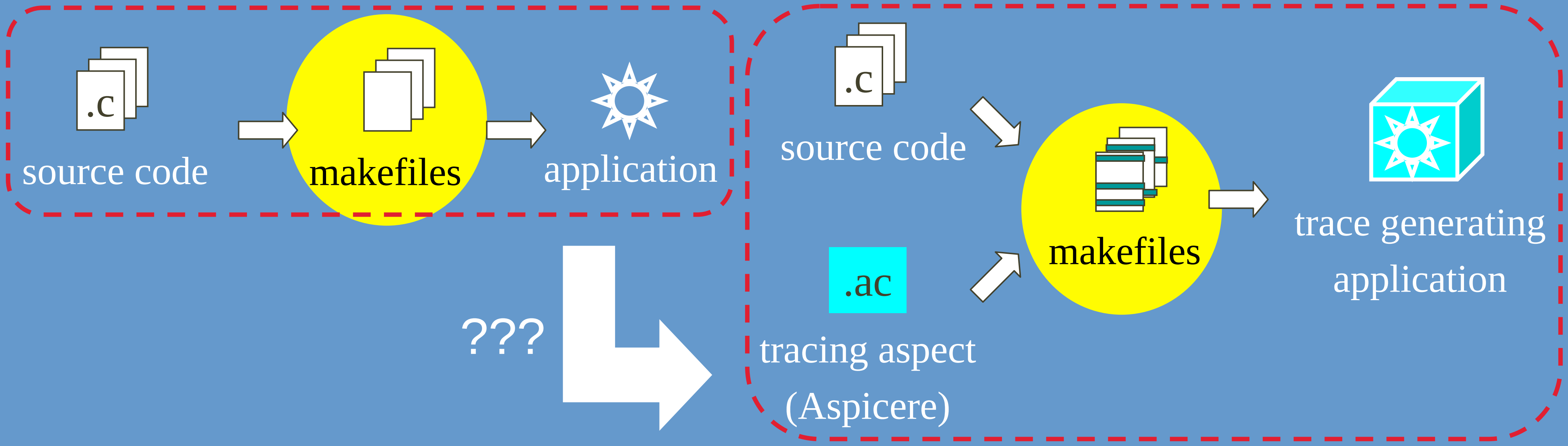
Kris DE SCHUTTER
Lab On REengineering, University of Antwerp
http://faramir.ugent.be/~kdschutt

Outline

1. Why Look At Build Systems?
2. Software Architecture Recovery
3. Make
4. MAKAO
5. Rule-Based Approach
6. General Rules
7. Application-Specific Rules
8. Conclusions and Future Work

1. Why Look At Build Systems?

Case study with Aspicere:



.c

source code

makefiles

application

???

.c

source code

.ac

tracing aspect

(Aspicere)

makefiles

trace generating

application

More general:

- how to easily modify a build system?
- how to gain quick insight into build process?
- how to assess general software architecture?

re-engineering

reverse-engineering

2. Software Architecture Recovery

Software architecture recovery:

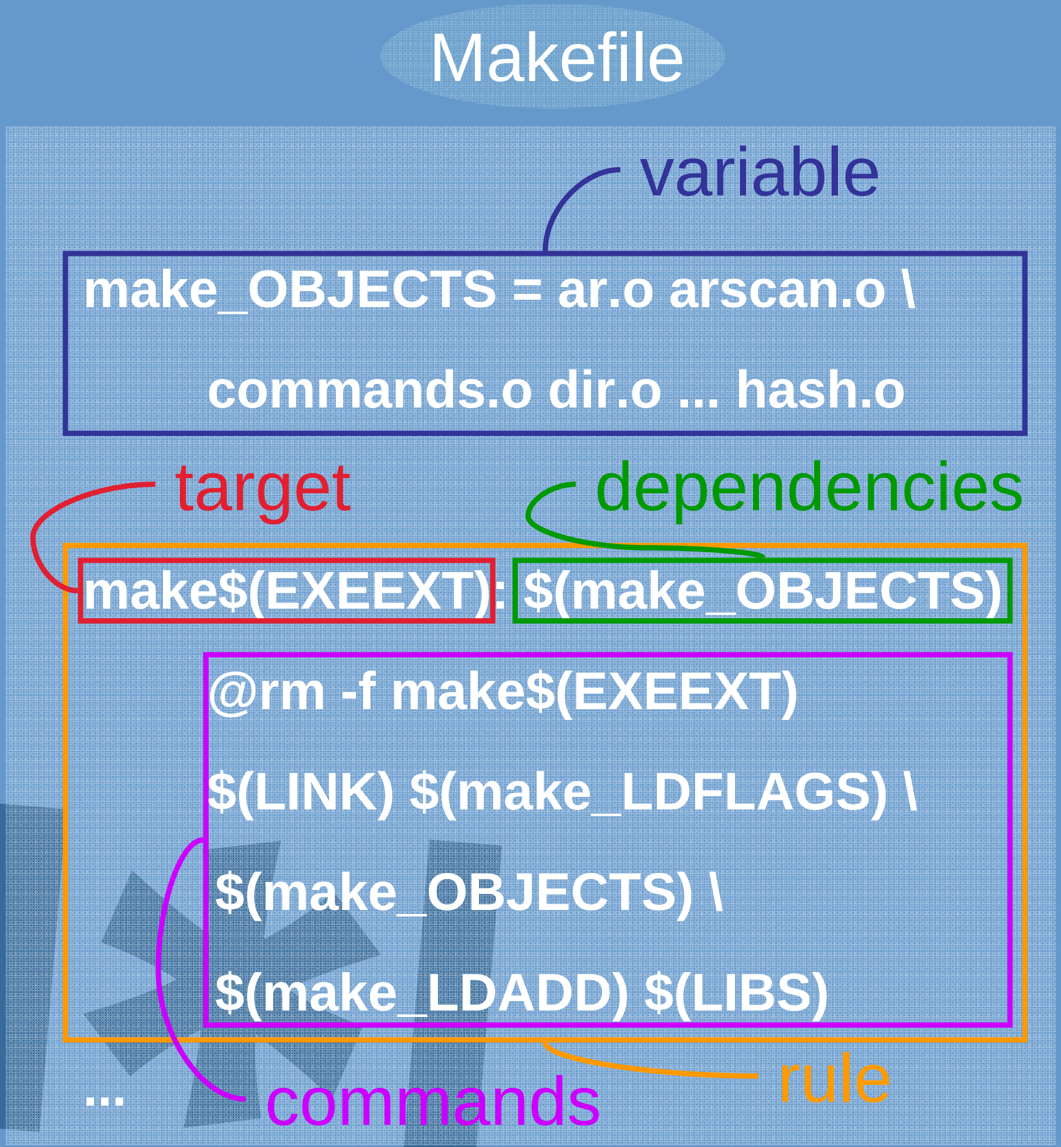
- software and build system co-evolve
- assumptions:
  - correct makefiles
  - modular source files (no giant implementation files)

Related work:

- Build-Time Software Architecture View [Tu01]
- Dali (and Rigi) [Kazman99], Portable BookShelf [Finnigan97], and Desire [Biggerstaff89]
- [Bowman99] Linux kernel architecture
  - conceptual architecture ⇒ concrete architecture
  - tedious discovery and population of subsystems

Feldman. “Make-a program for maintaining computer programs”. Software - Practice and Experience

3. Make



Makefile

variable

```
make_OBJECTS = ar.o arscan.o \
        commands.o dir.o ... hash.o
```

target

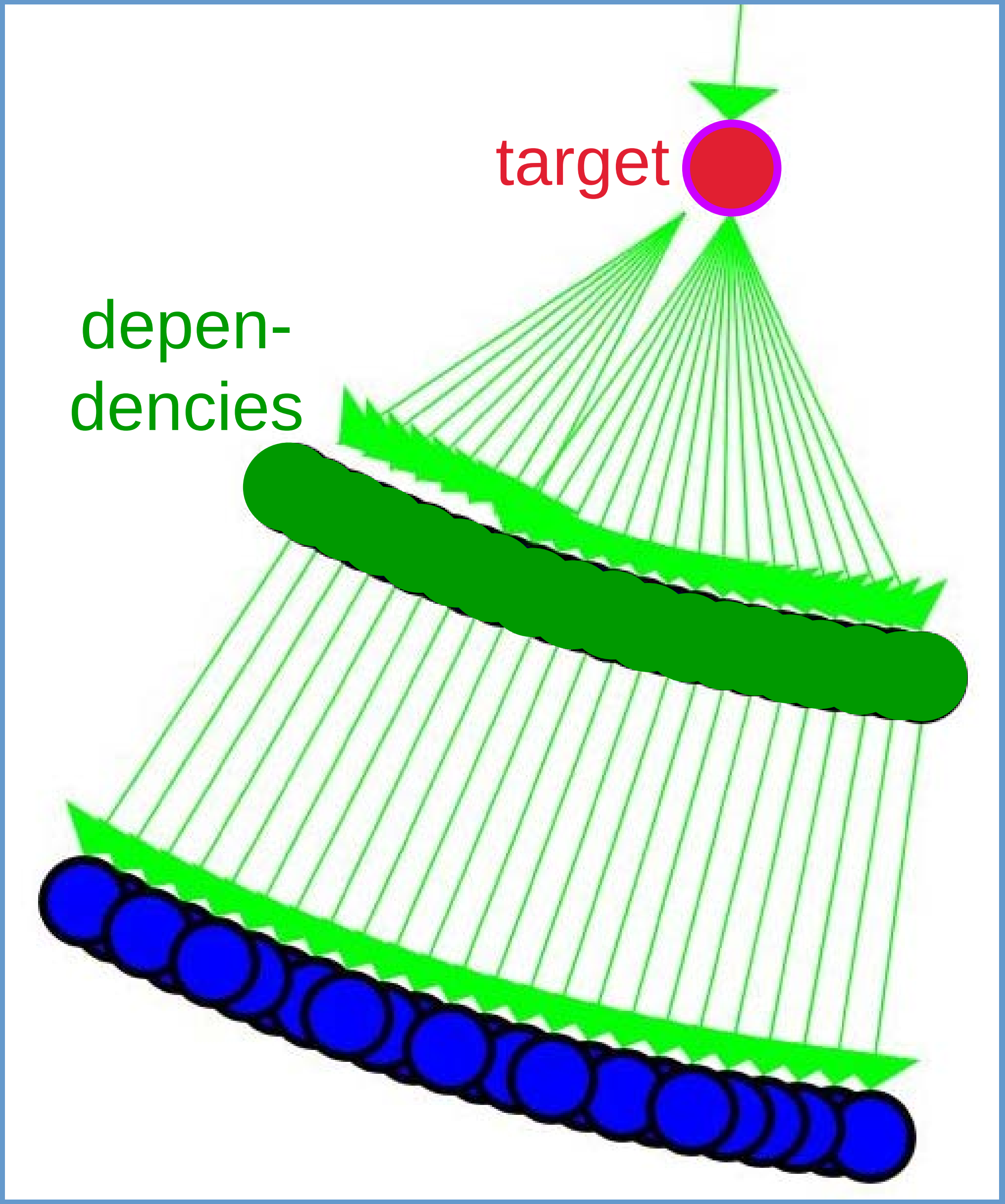
dependencies

```
make$(EXEEXT): $(make_OBJECTS)
        @rm -f make$(EXEEXT)
        $(LINK) $(make_LDFLAGS) \
         $(make_OBJECTS) \
         $(make_LDADD) $(LIBS)
...
```

commands

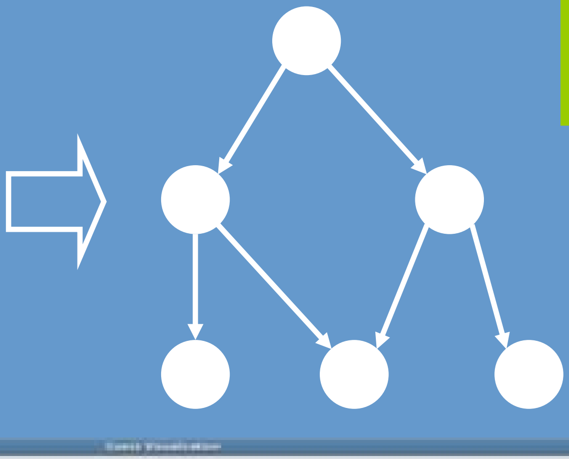
rule

Directed Acyclic Graph (DAG)

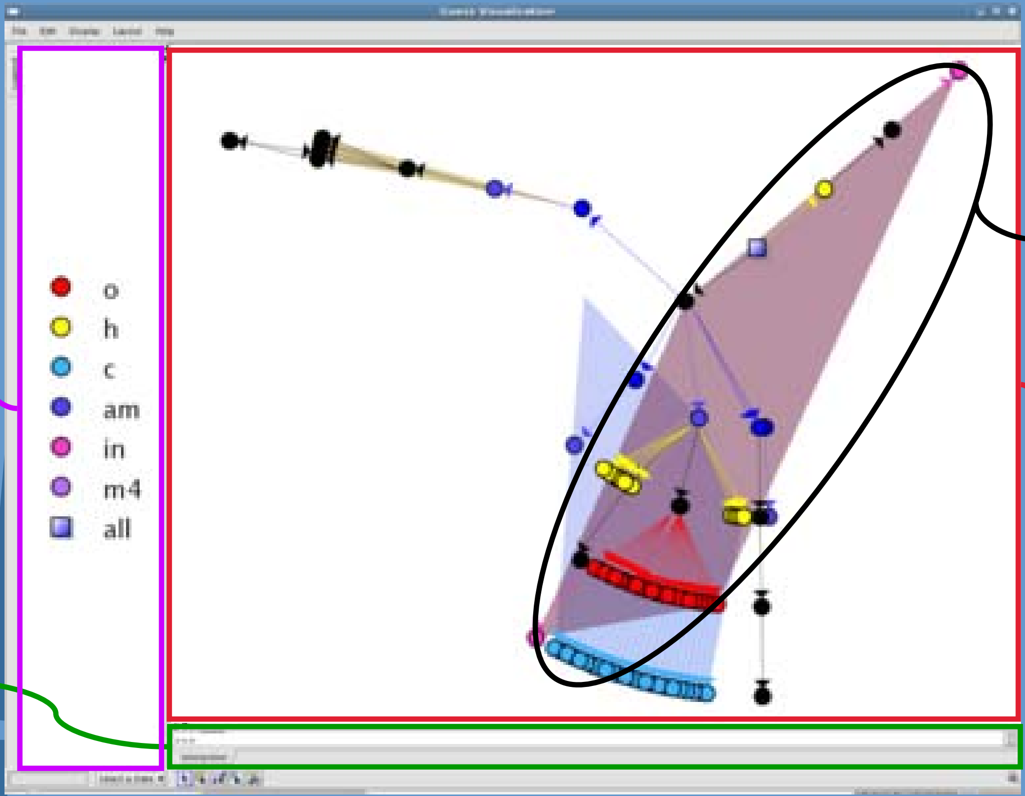


⇒ de facto build tool/process model!

4. MAKAO



Makefile Architecture Kernel for AO

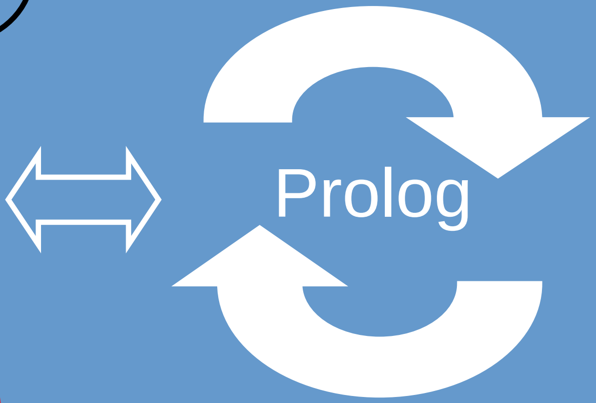


legend

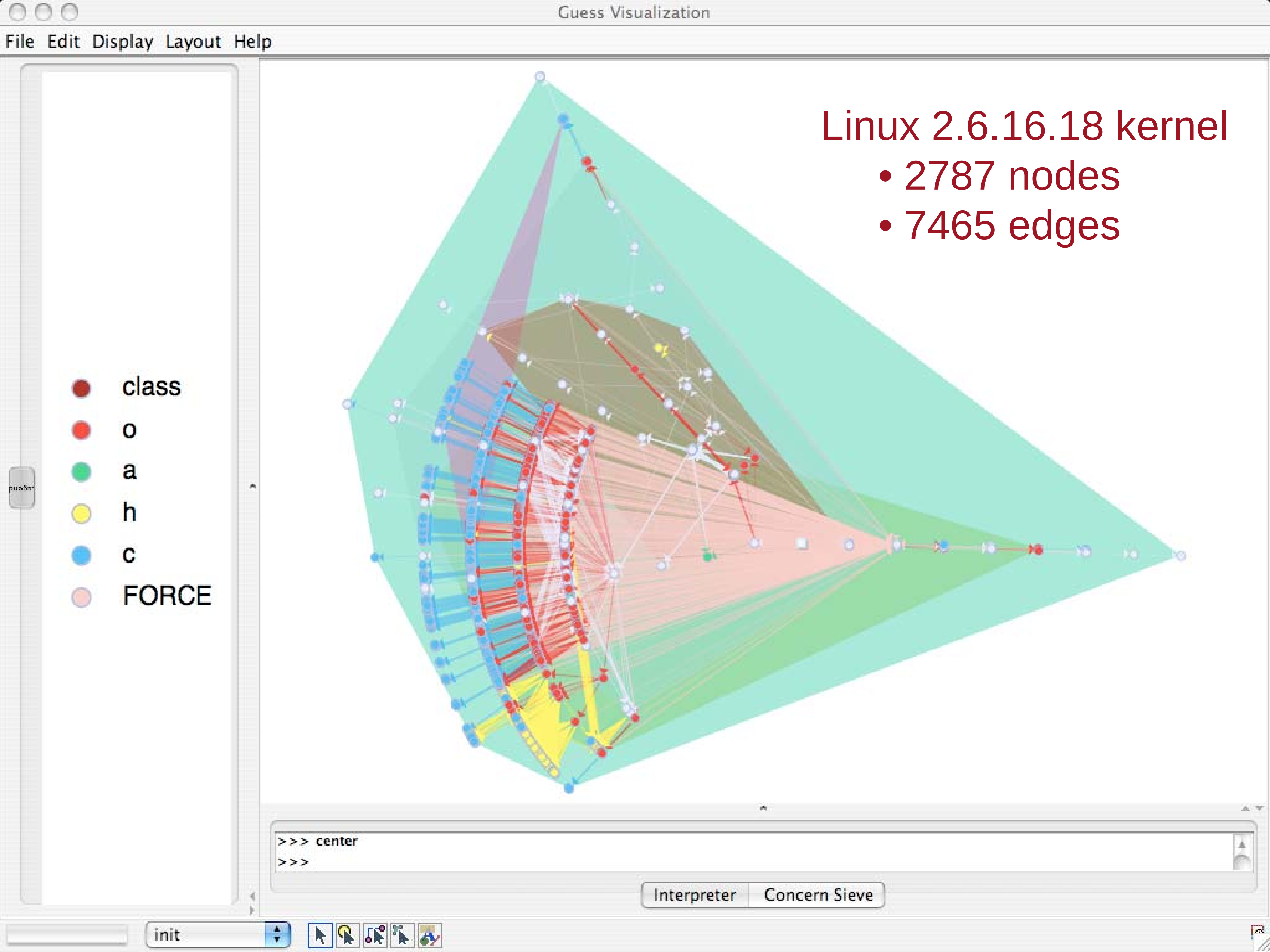
hull

graph

Gython console



Prolog



Linux 2.6.16.18 kernel

- 2787 nodes
- 7465 edges

```
>>> center
>>>
```

5. Rule-Based Approach

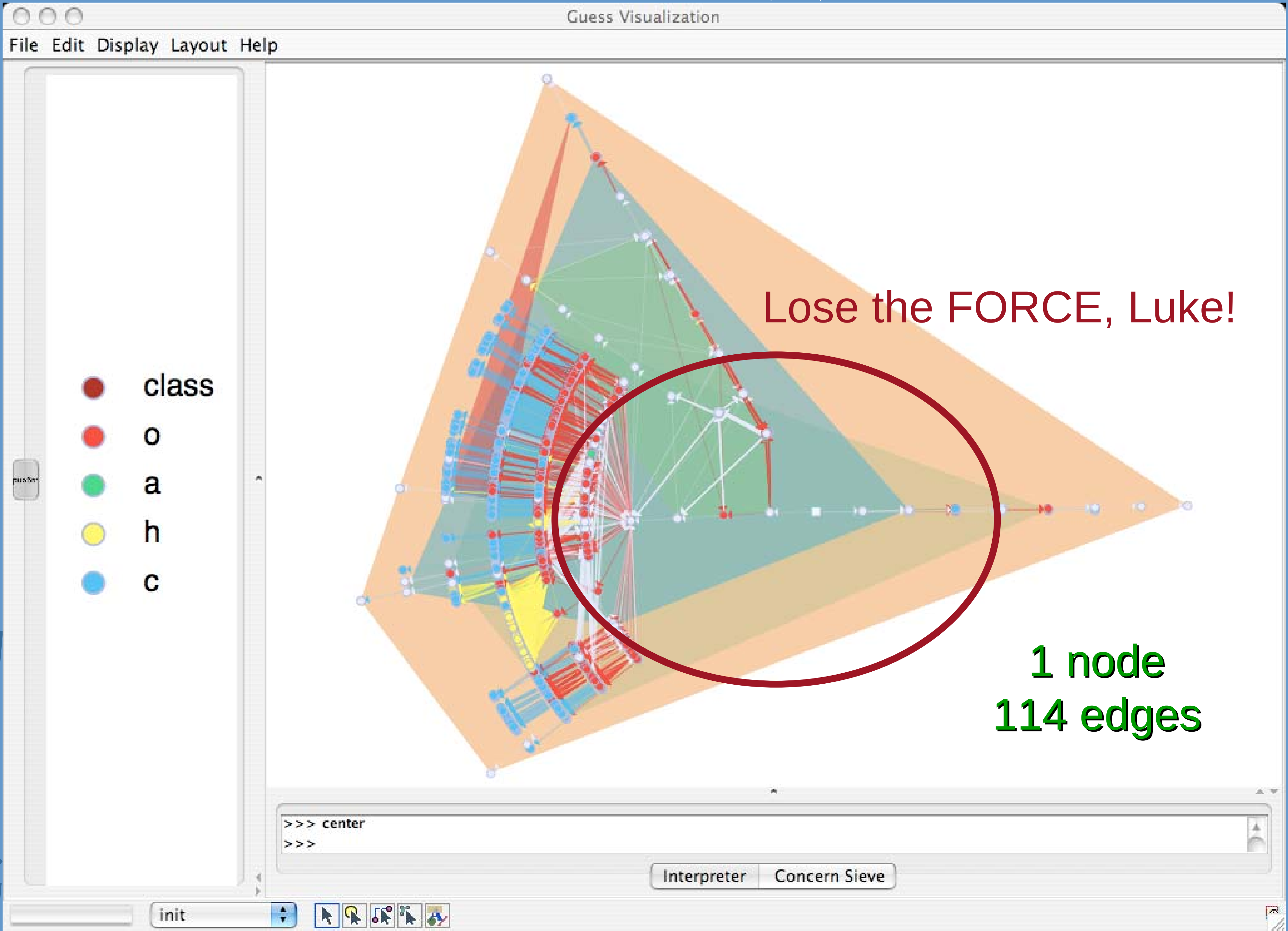
Observations:

- previous slide looks like a mess, even after layouting
- too much detail

Possible solution:

- define rules to modify graph:
  - general vs. application-dependent [Kazman99]
  - semantics-preserving ("cleaning-up") or not
- challenge: don't touch the code ↔ [Bowman99]
- propagate clean-up passes back to build (configuration?) system

6. General Rules (1)



```
>>> center
>>>
```

Lose the FORCE, Luke!

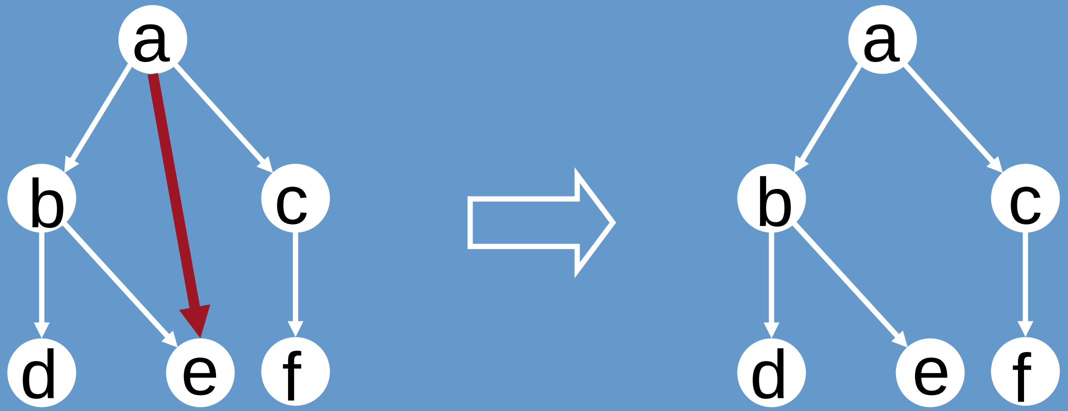
1 node
114 edges

6. General Rules (2)

☺
- semantics-preserving
- faster build
- lose architectural info?

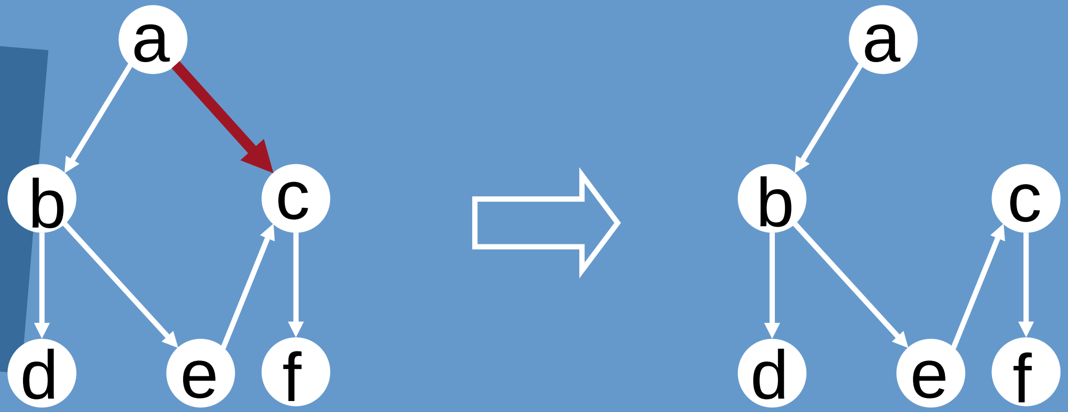
Redundant dependencies:

- simple transitivity



0 nodes
108 edges

- extended transitivity



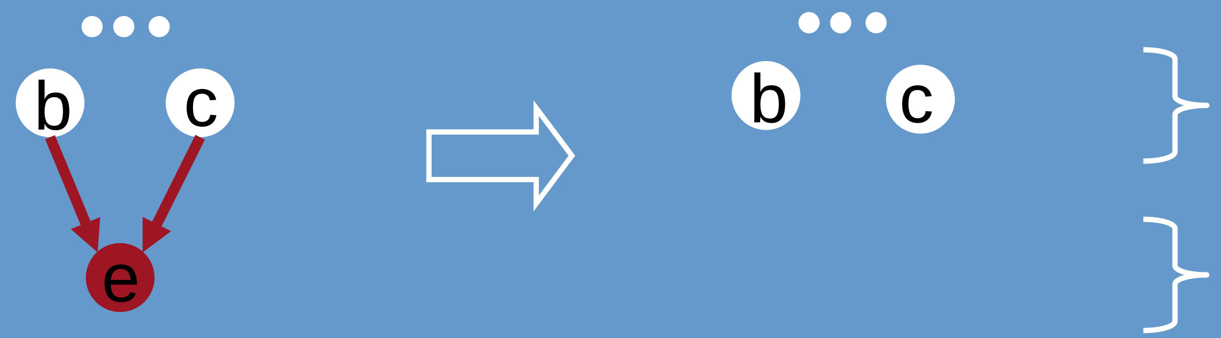
not applied

6. General Rules (3)

Redundant dependencies (cont.):

- obsoleteness

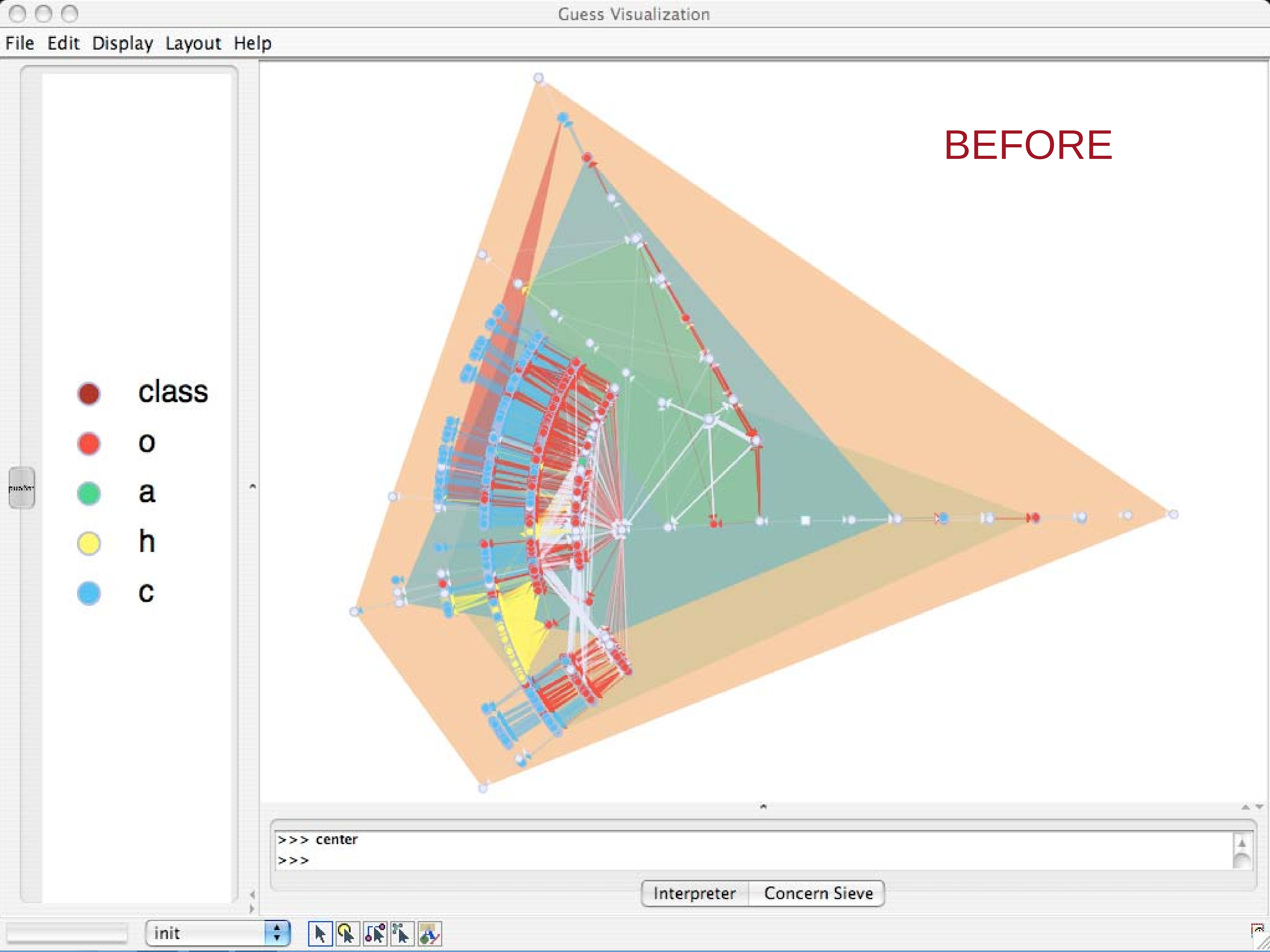
11 nodes
17 edges



no object

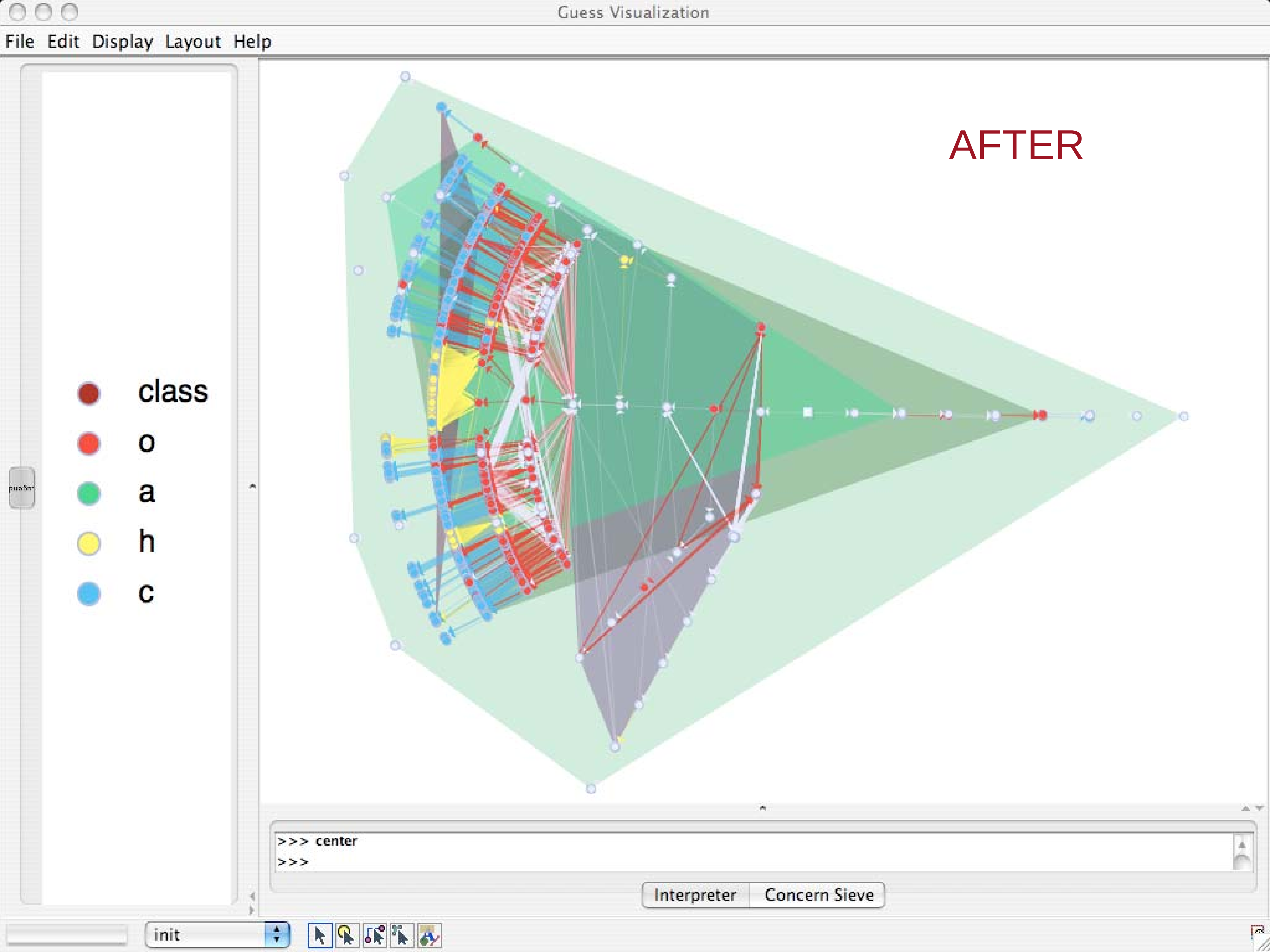
source

- semantics-preserving if no commands tied to source node
- faster build



BEFORE

```
>>> center
>>>
```



AFTER

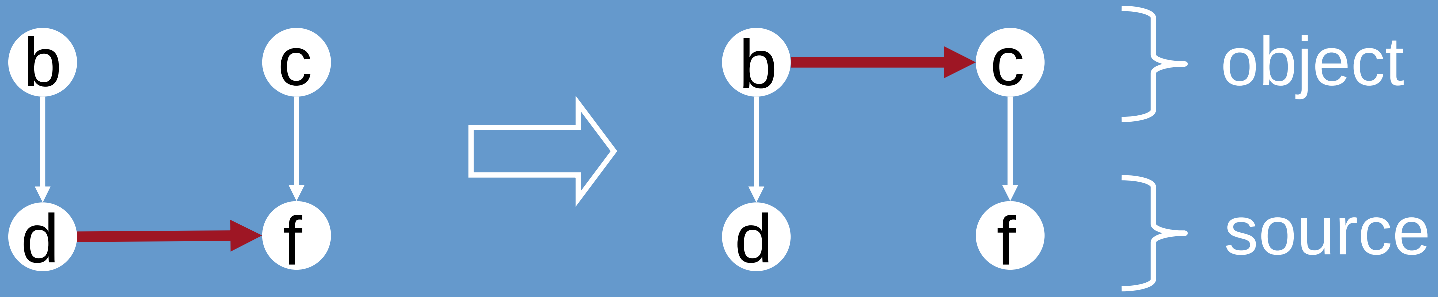
```
>>> center
>>>
```

6. General Rules (4)

Raising level of abstraction:

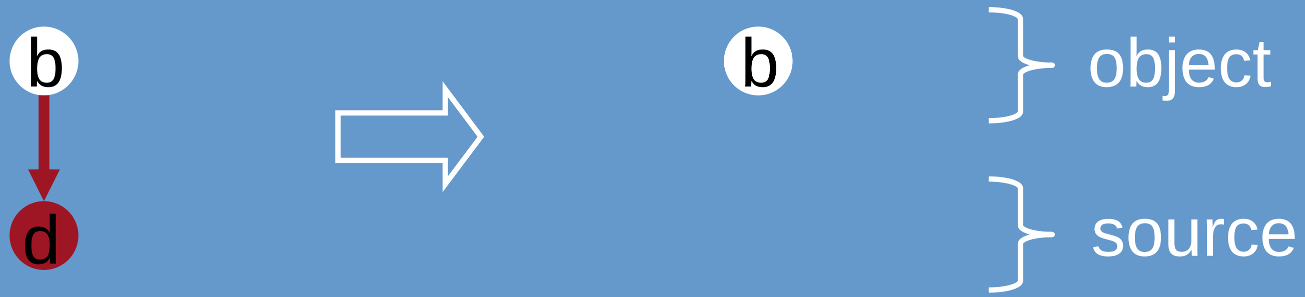
- pulling up source file relations

0 nodes
0 edges



- abstracting away source files

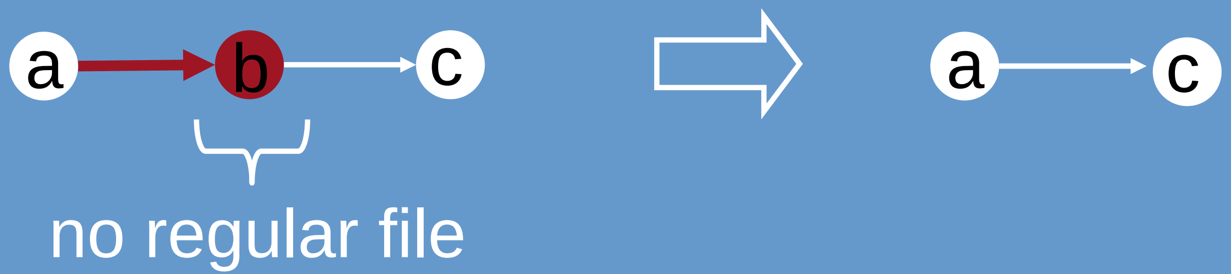
929 nodes
944 edges



6. General Rules (5)

Raising level of abstraction:

- sandwich rule

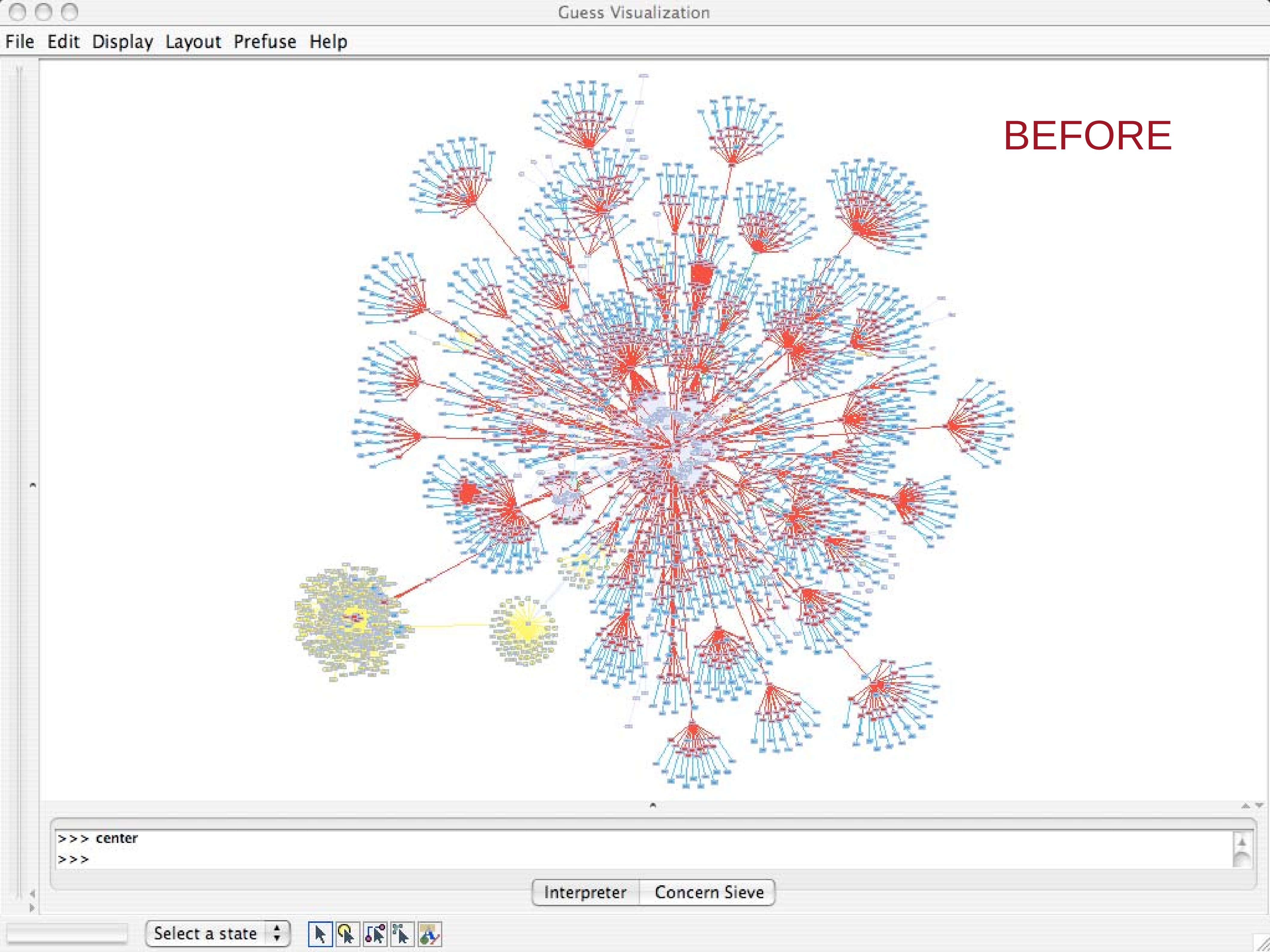


14 nodes
14 edges

☺ abstraction

⚡
- rules influenced by style of build scripts

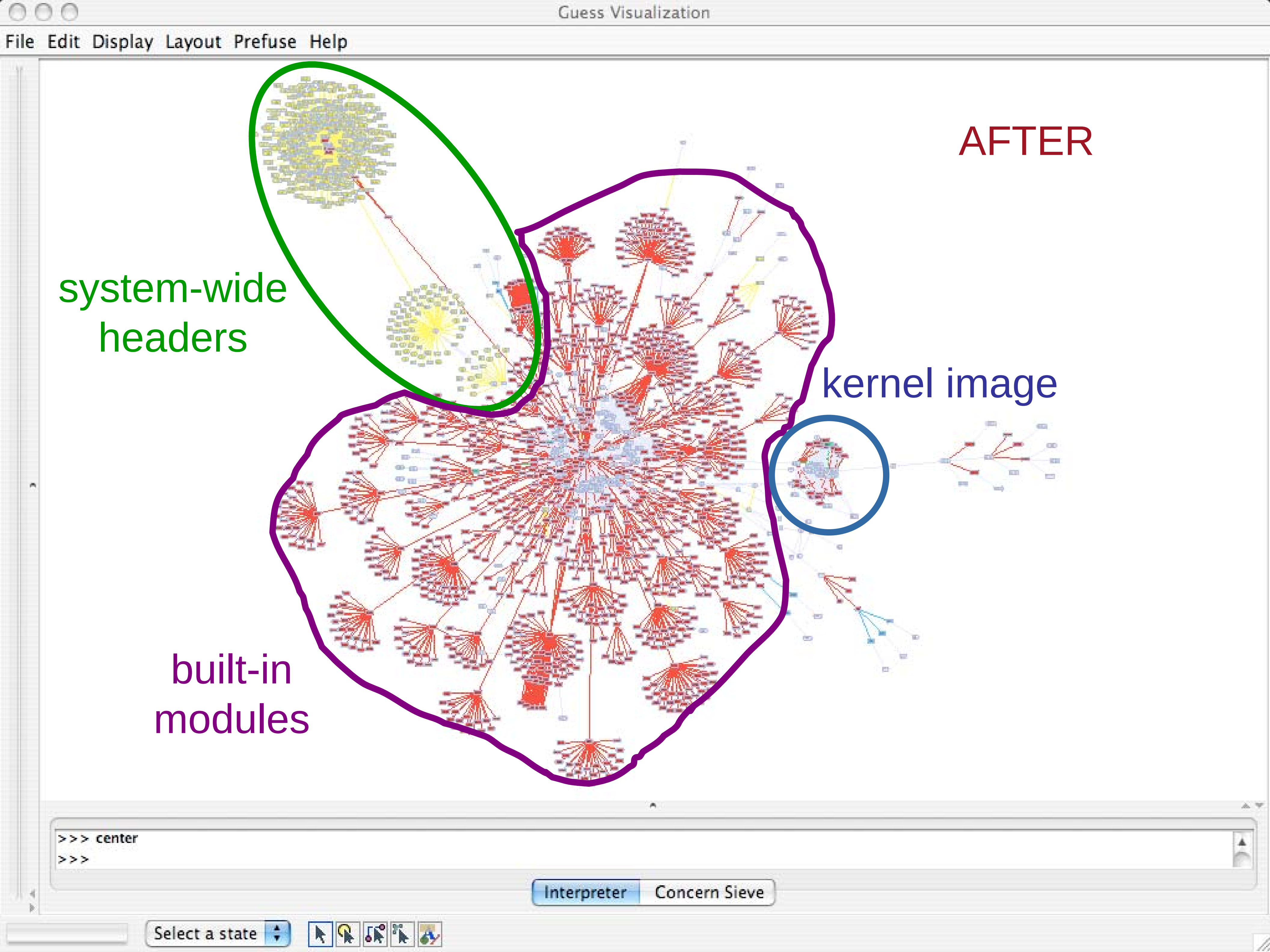
⇒ some build systems have more/less architectural info
- lose architectural info?



Guess Visualization
File Edit Display Layout Prefuse Help
BEFORE

```
>>> center
>>>
```

Interpreter
Concern Sieve
Select a state



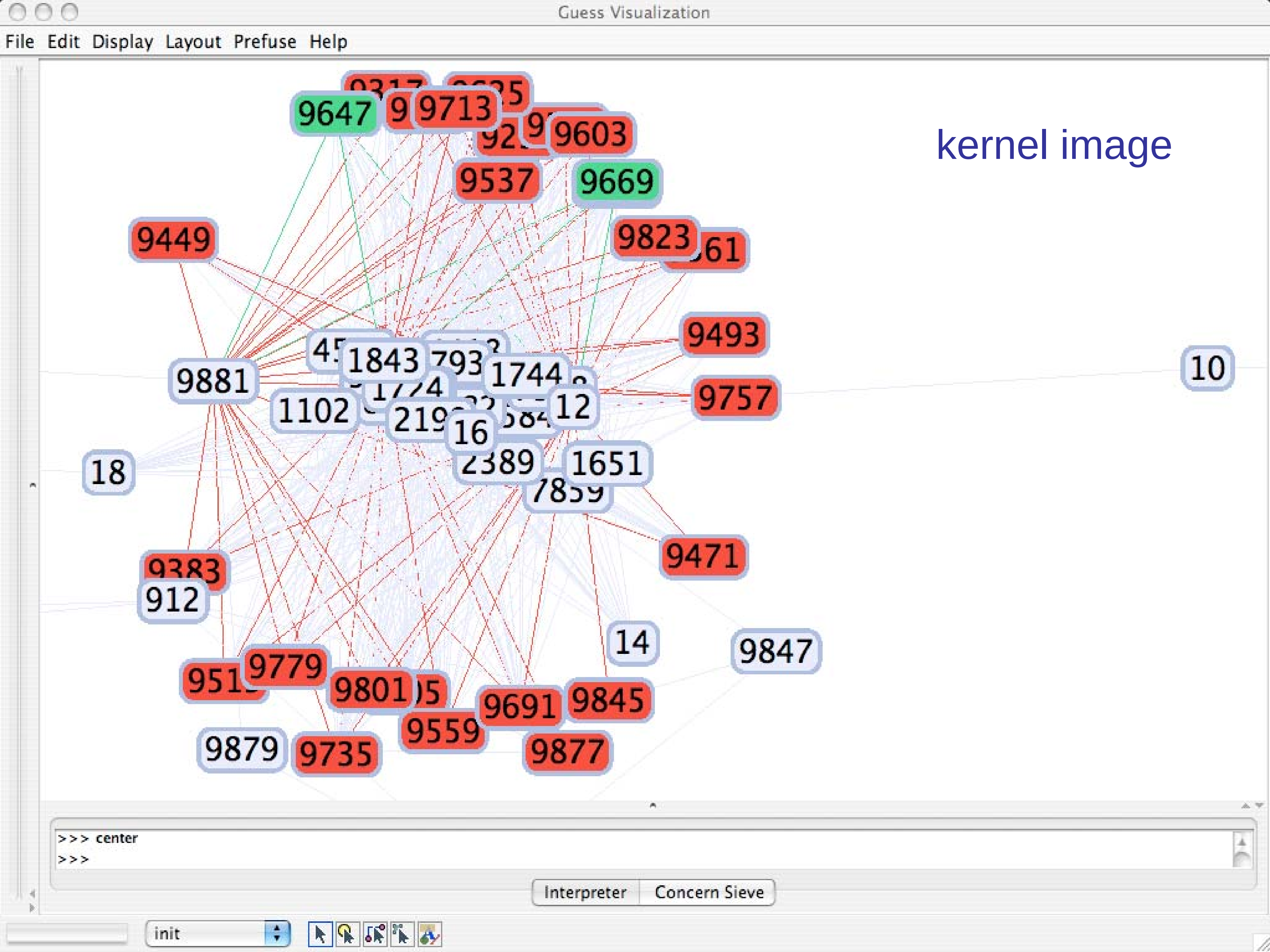
AFTER

system-wide headers

kernel image

built-in modules

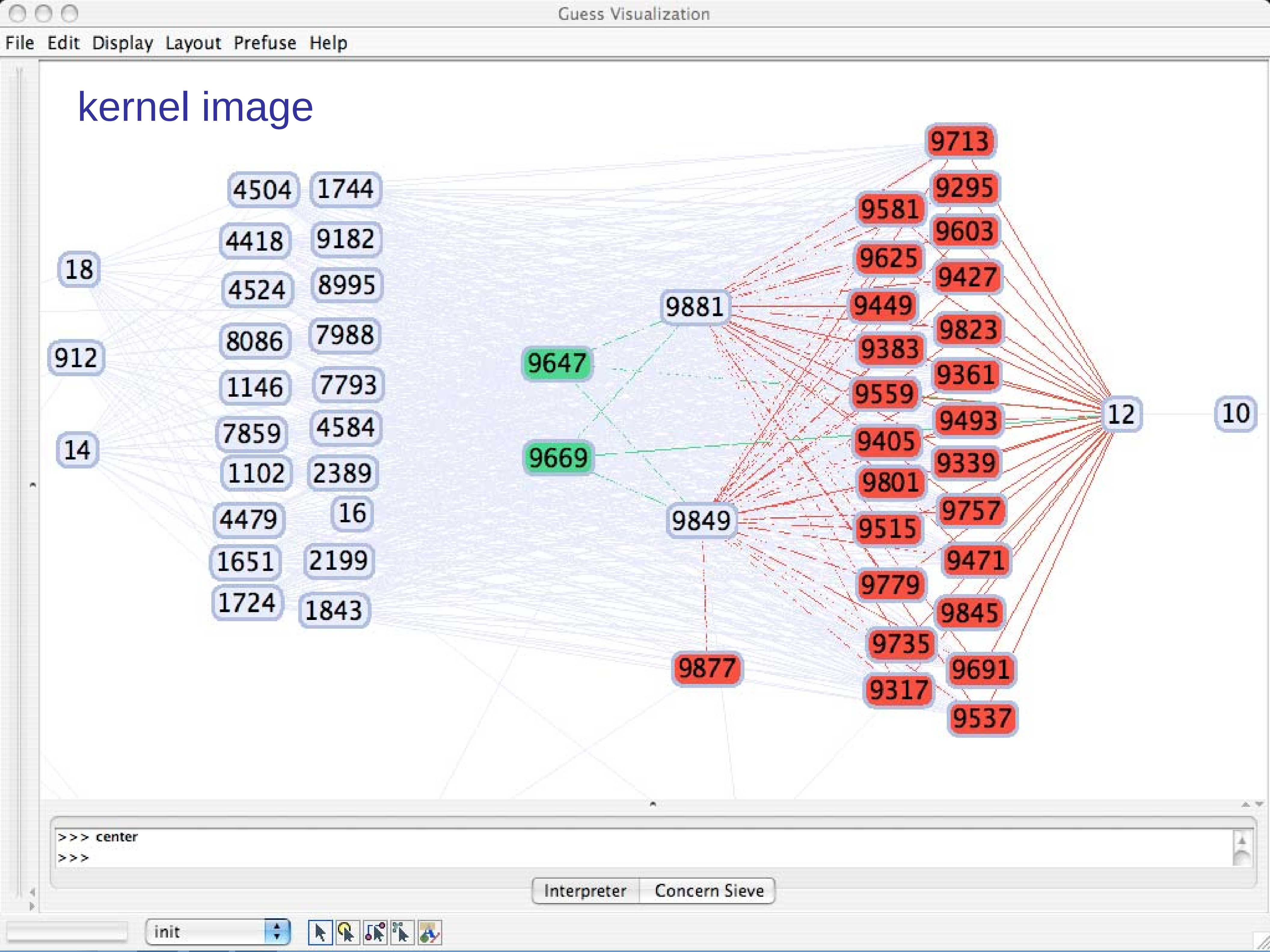
```
>>> center
>>>
```



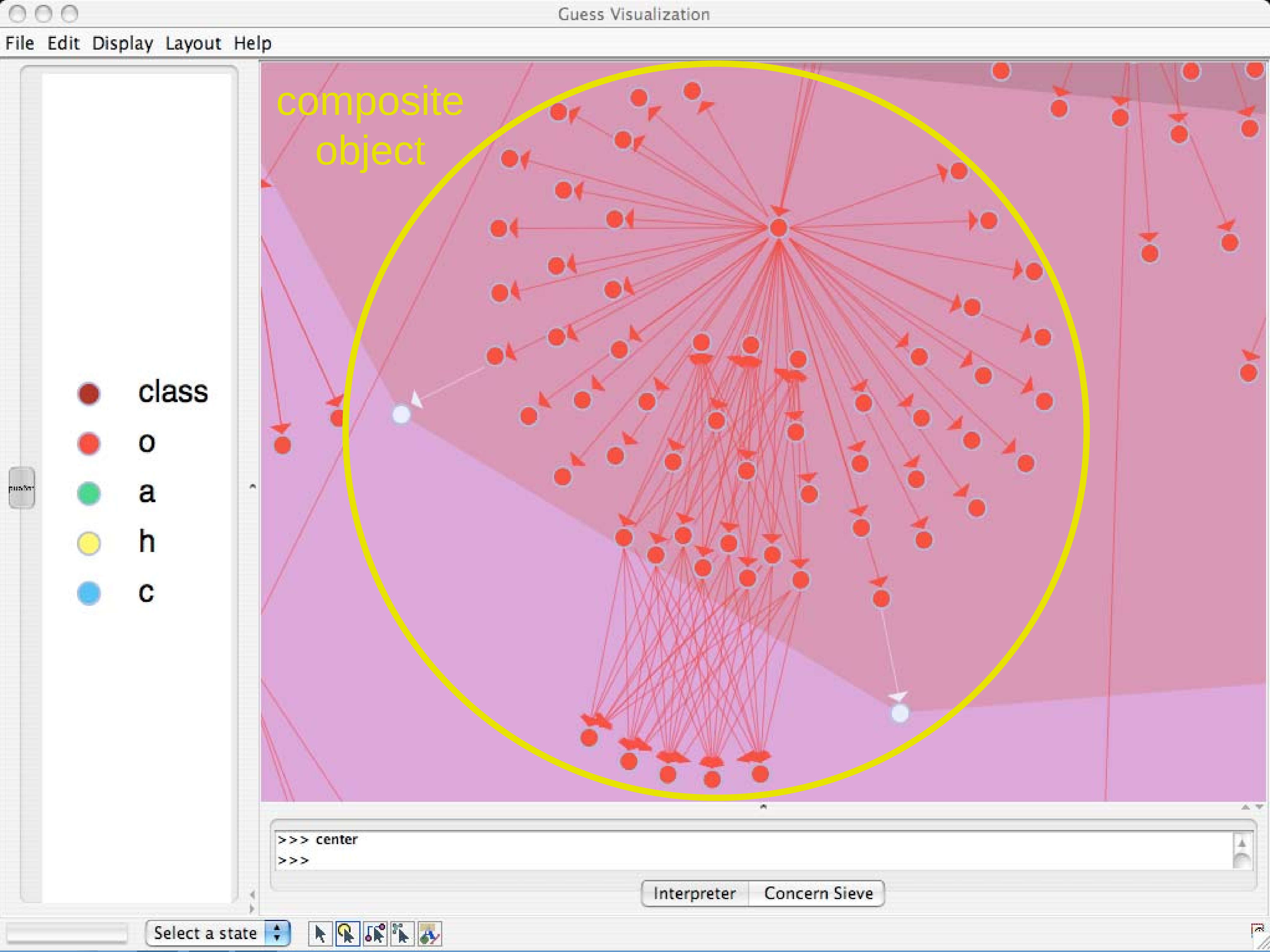
kernel image

```
>>> center
>>>
```

kernel image



```
>>> center
>>>
```



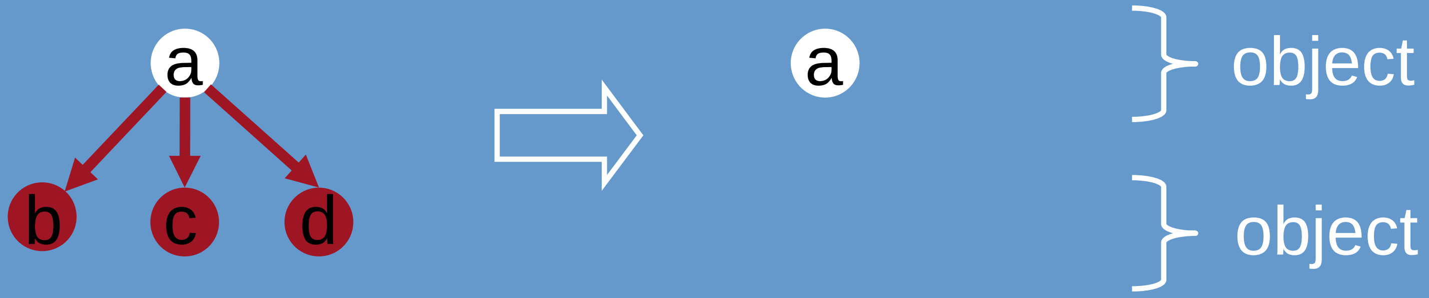
composite object

```
>>> center
>>>
```

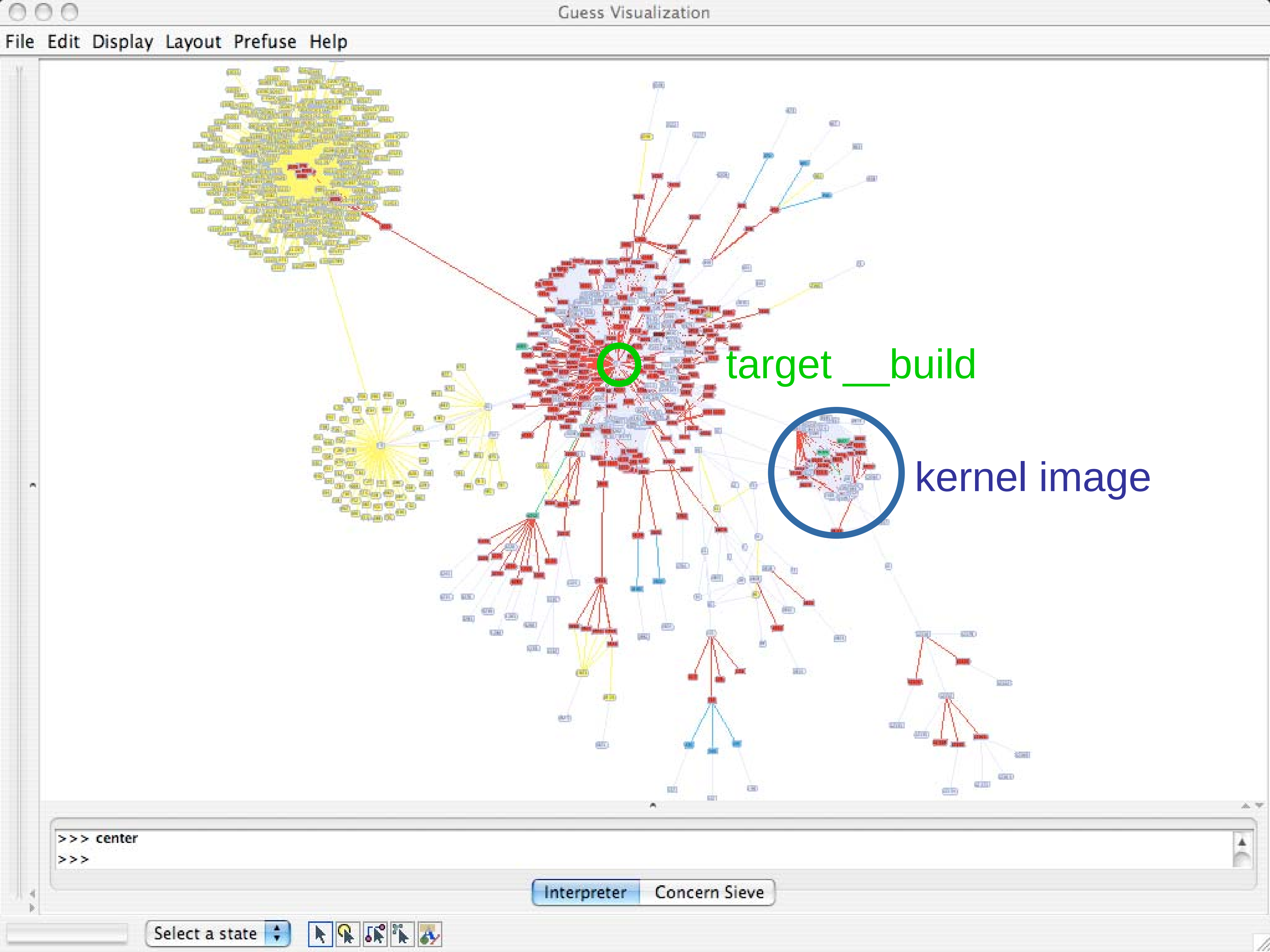
7. Application-Specific Rules (1)

- composite object files

897 nodes
1056 edges



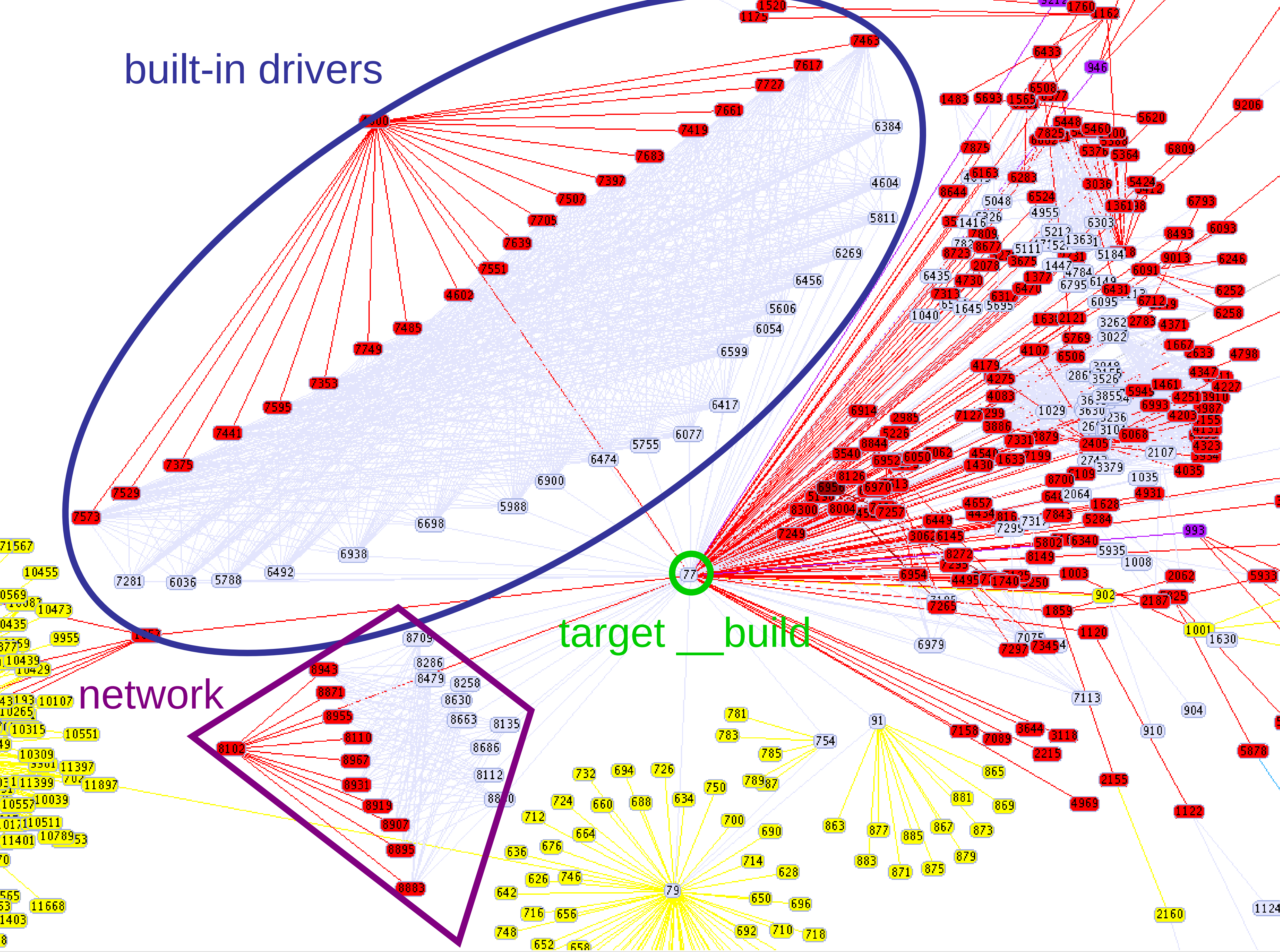
☺ highly effective



target __build

kernel image

```
>>> center
>>>
```



built-in drivers

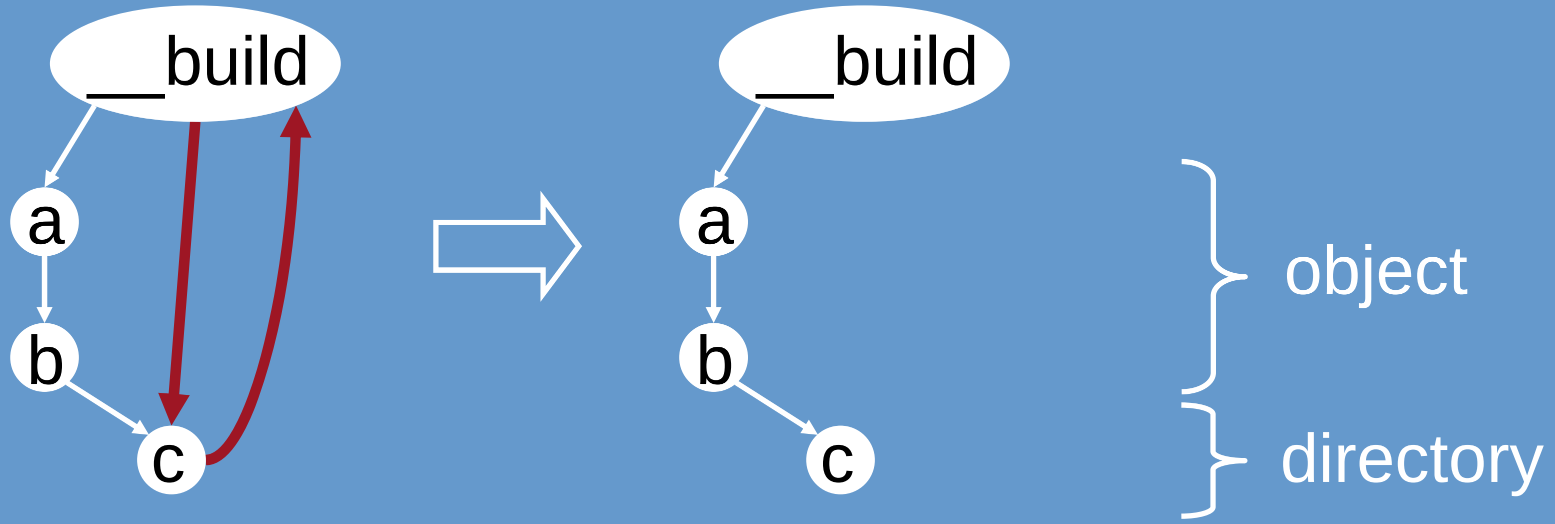
network

target __build

7. Application-Specific Rules (2)

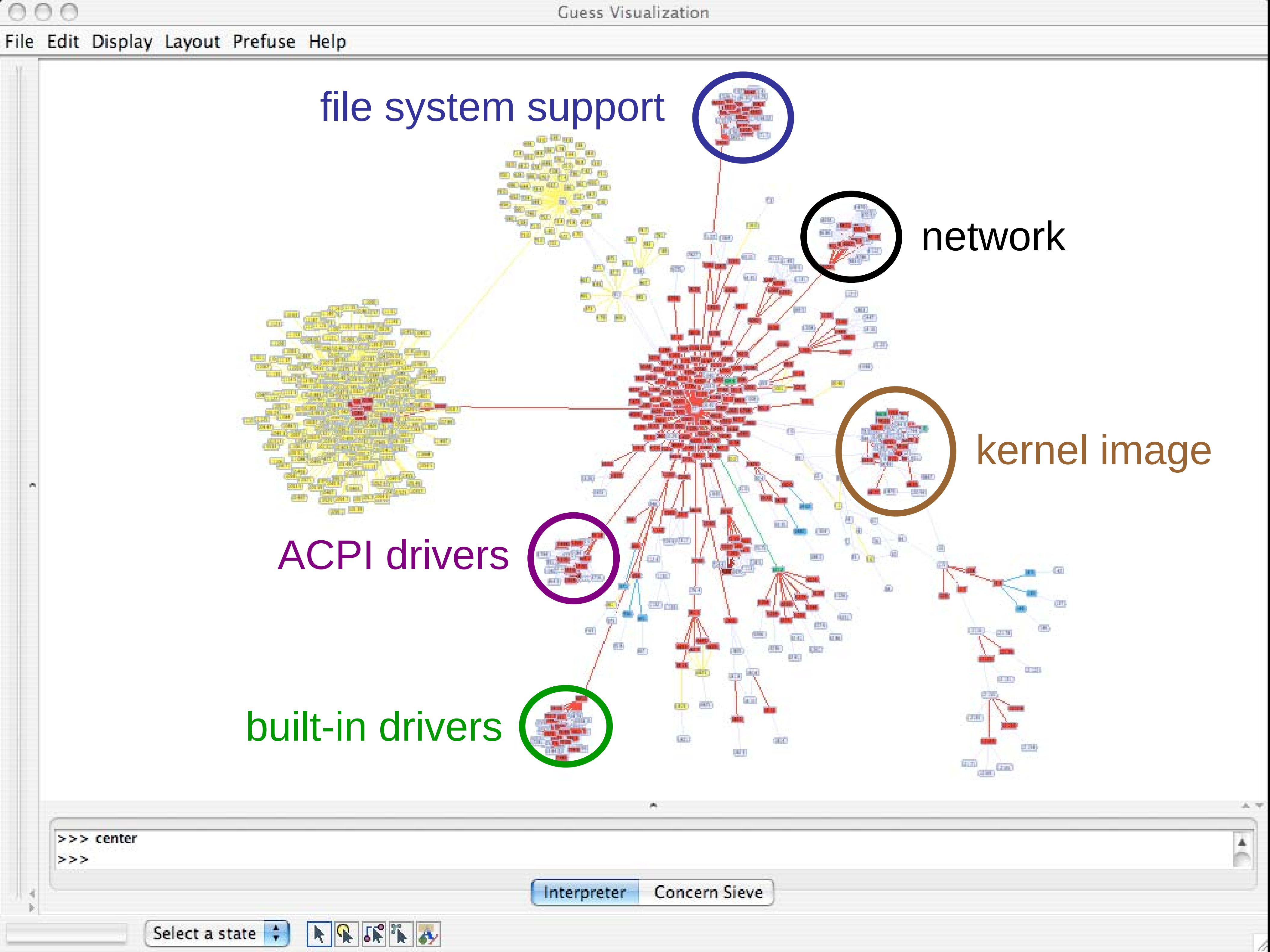
- unchaining redundant cycles

0 nodes
174 edges



☺ decouples tangled clusters

⚡ what does this construct mean?


Guess Visualization
File
Edit
Display
Layout
Prefuse
Help
file system support
network
kernel image
ACPI drivers
built-in drivers
Interpreter
Concern Sieve
Select a state

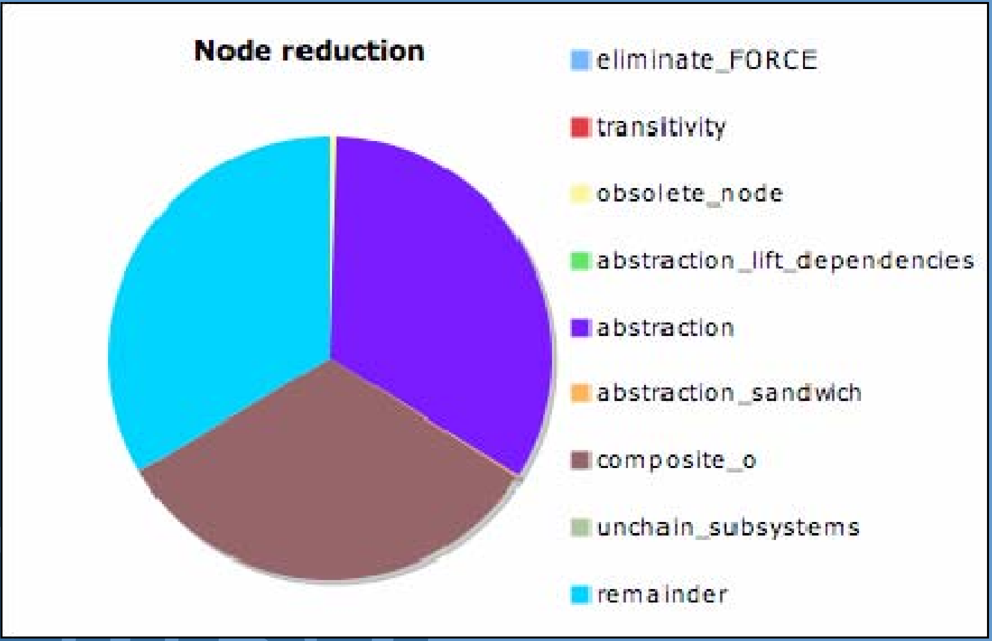
```
>>> center
>>>
```

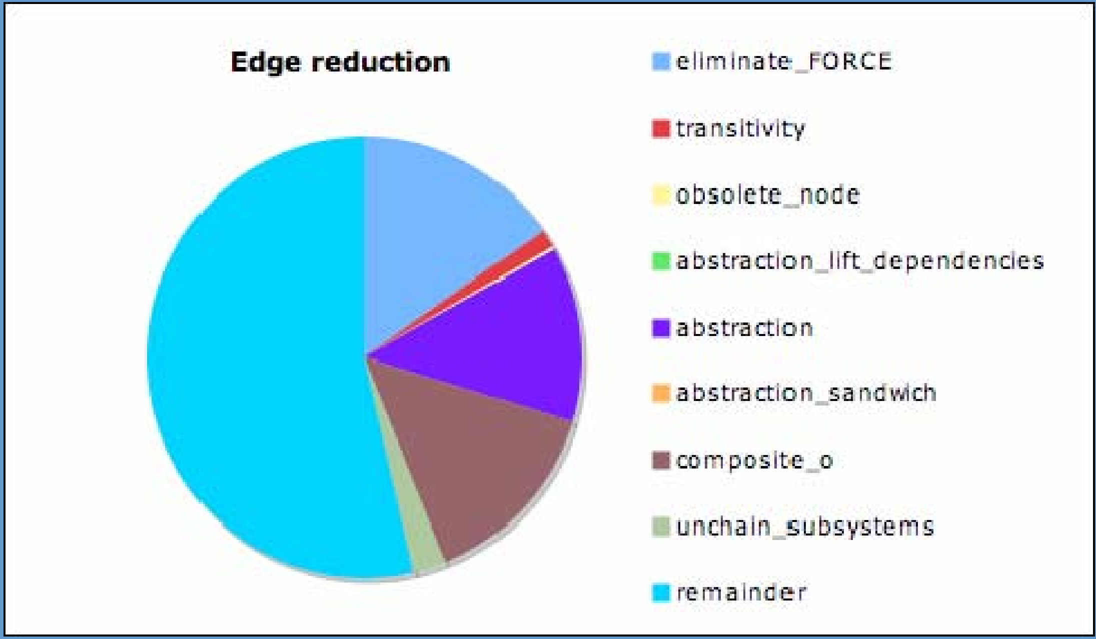
8. Conclusions and Future Work (1)

Conclusions:

- work in progress!
- lots of clean-up and abstraction rules necessary
- build system's knowledge varies per project

Rules' effectiveness:





8. Conclusions and Future Work (2)

Future work:

- working out dependencies of kernel image
- other cases (GCC, vim, KDE, ...)
- applying clustering techniques
- feed clean-up rules back to build scripts
- come up with new rules
- does order of rules play a role?
- ...

References

[Biggerstaff89] *Ted J. Biggerstaff. Design Recovery for Maintenance and Reuse. Computer Journal, Vol. 22, No. 7 (p. 36-49), 1989*

[Bowman99] *Ivan T. Bowman, Richard C. Holt and Neil V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. Proc. of ICSE 1999 (p. 555-563)*

[Finnigan97] *P. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. Mueller, J. Mylopoulos, S. Perelgut, M. Stanley, and K. Wong. The Software Bookshelf. IBM Systems Journal, Vol. 36, No. 4 (p. 564-593), November 1997*

[Kazman99] *Rick Kazman and S. Jeromy Carrière: Playing detective. Reconstructing software architecture from available evidence. Proc. of ASE 1999 (p. 107-138)*

[Tu01] *Qiang Tu and Michael W. Godfrey. The Build-Time Software Architecture View. Proc. of ICSM 2001 (p. 398-407)*