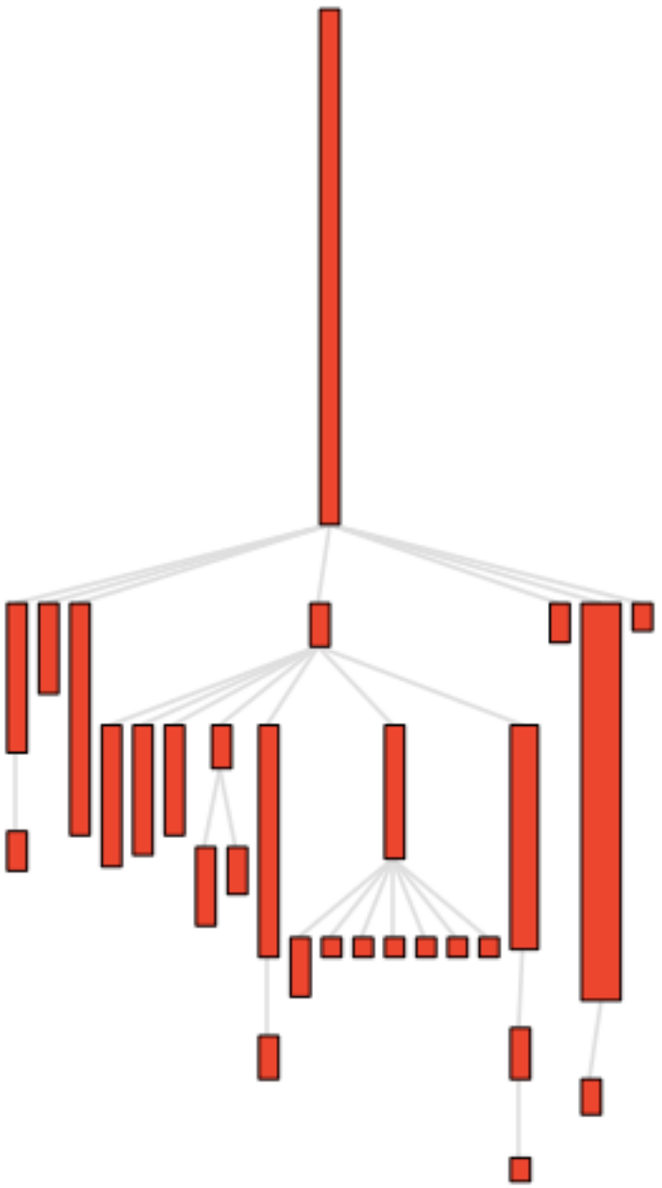
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Identifying Shareable Objects

Alejandro Infante
ainfante@dcc.uchile.cl



```
| b |
b := RTMondrianViewBuilder new.
b shape rectangle
        width: #numberOfVariables;
        height: #numberOfMethods;
        fillColor: [ Color new r: 1 g: 0 b: 0 ].
b nodes: Collection withAllSubclasses.
b edges: Collection withAllSubclasses from: #superclass to: #yourself.
b treeLayout.
b build.
b view open.
```

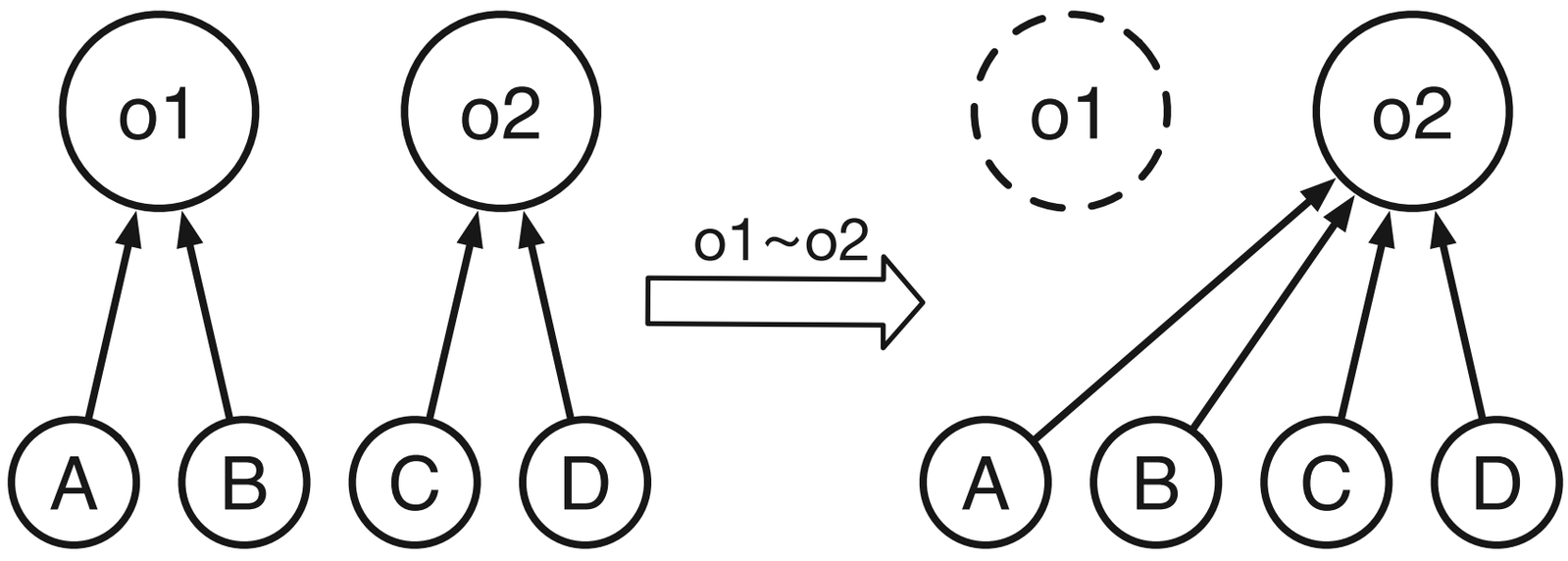
(Color new r: 1 g: 0 b: 0)

Related Work

- Object Equality Profiling *(Marinov and O'Callahan)*
  - Take a snapshot of the object graph and record last time an object has been written or read to identify potential shareable objects
- Cachetor: Detecting Cacheable Data to Remove Bloat *(Nguyen and Xu)*
  - *Use a novel technique called Value Abstracted Dependency Analysis at instruction level, then use the graph to detect cacheable values and structures*

Shareable Objects

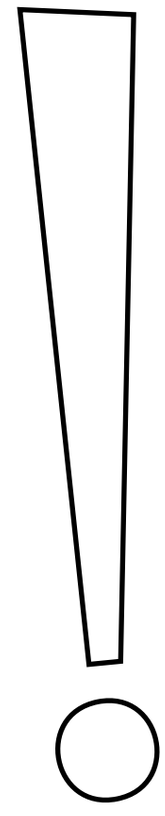
o1 ~ *o2* if all objects referencing *o1* can change their reference to *o2* preserving the program behavior.



Shareable Objects

$$o1 \sim o2 :$$

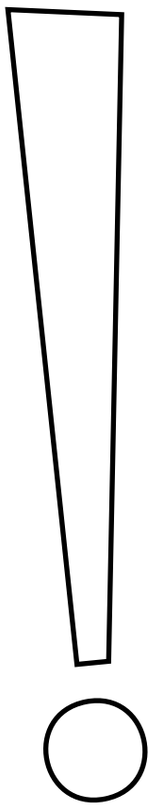
$$\begin{gathered}
\text{class(o1)} = \text{class(o2)} \wedge \\
(\forall f \in instVars(o1).(o1.f \sim o2.f)) \wedge \\
\text{immutable(o1)} \wedge \\
\text{immutable(o2)}
\end{gathered}$$



Defining Immutability

o is immutable only if:

$$\nexists f \in instVars(o)$$
$$\exists t_1, t_2 \wedge t_1 < t_2$$
$$\mathrm{o.f}_{t_1} \neq nil \wedge o.f_{t_1} \neq o.f_{t_2}$$



Summarizing: Instance variables change only ONCE

Dynamic Analysis

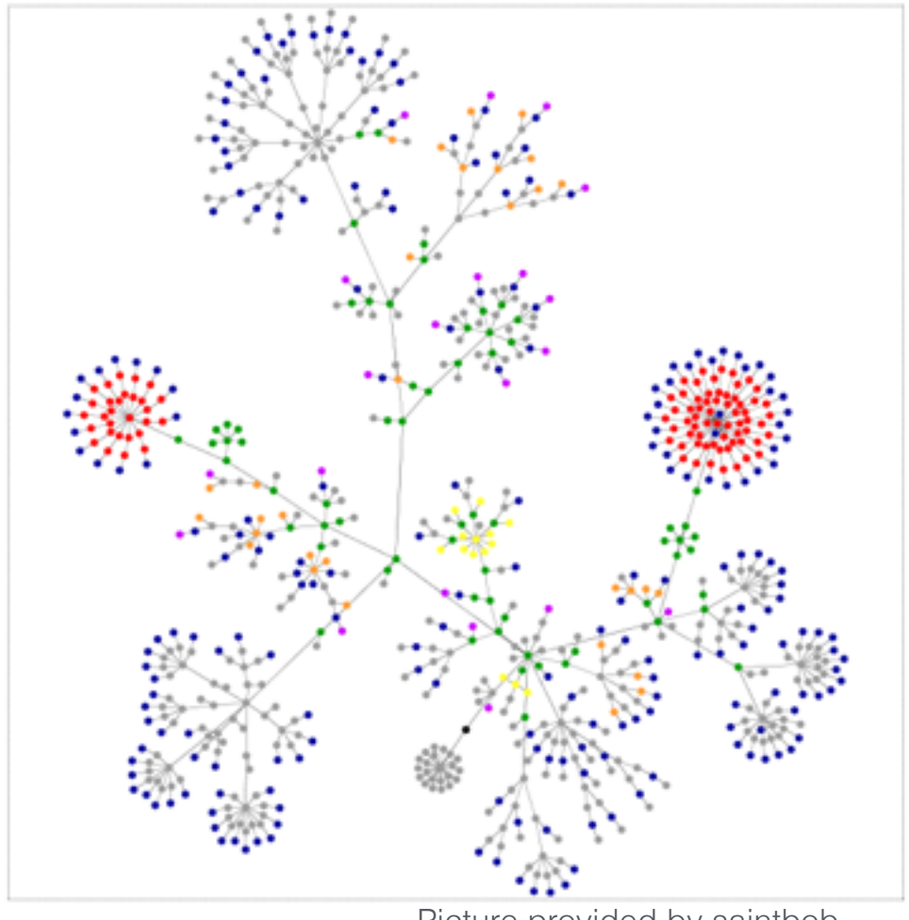
Definition:

- ➡ Object
- ➡ Primitive Value
- ➡ External Classes
  - String*
  - Empty Arrays*

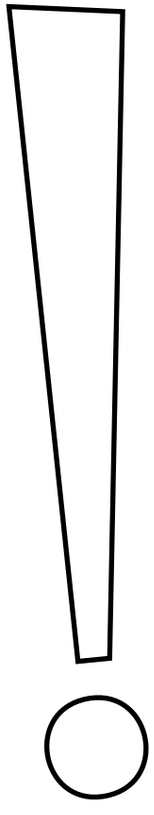
Events:

- ➡ Object Creation
- ➡ Object first time seen
- ➡ Instance Variable Writing
- ➡ Object Garbage Collection

Object Graph in time T

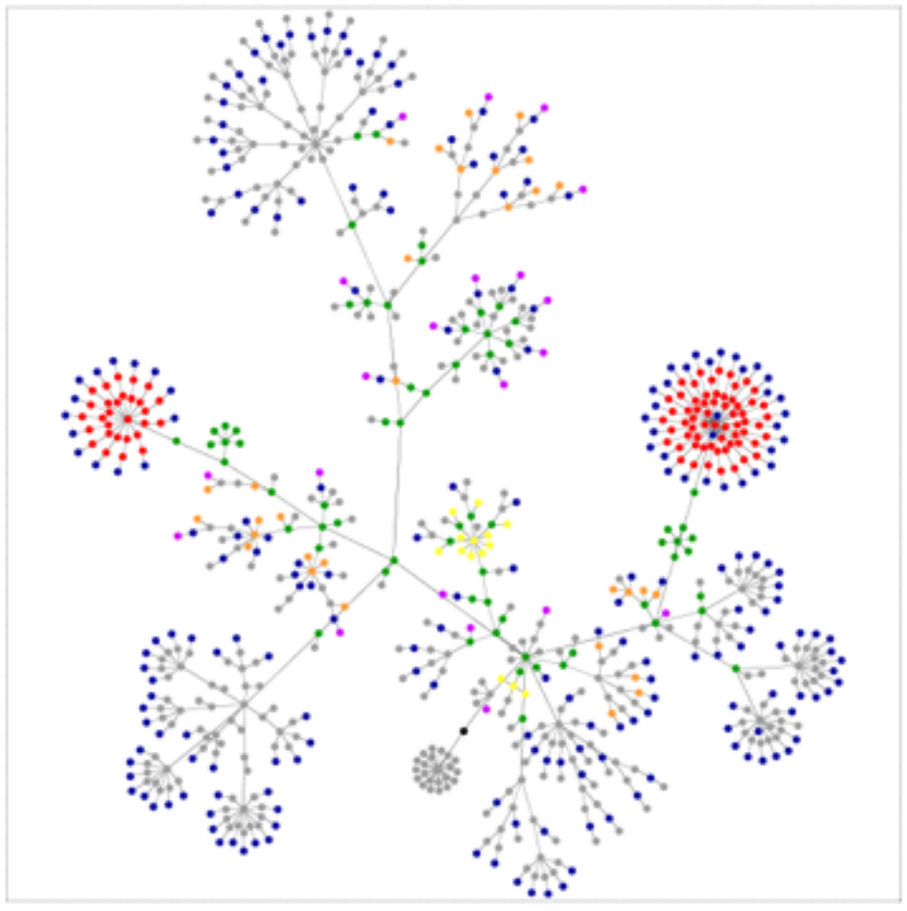


Picture provided by saintbob



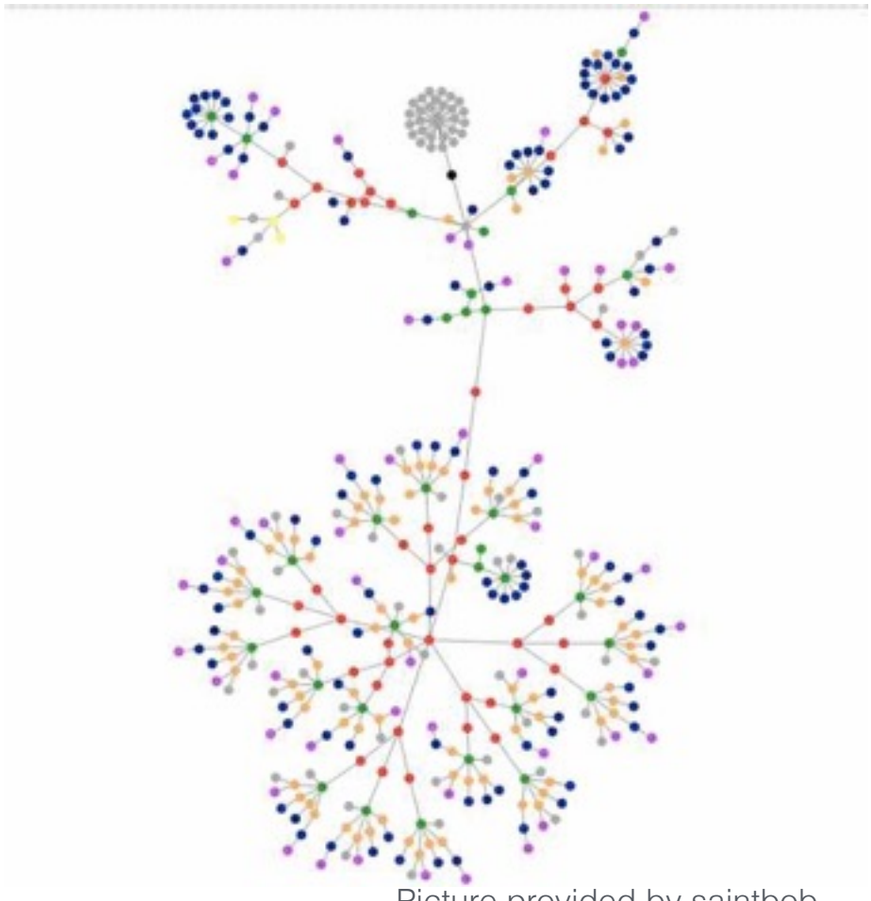
Dynamic Analysis

Object Graph in time T

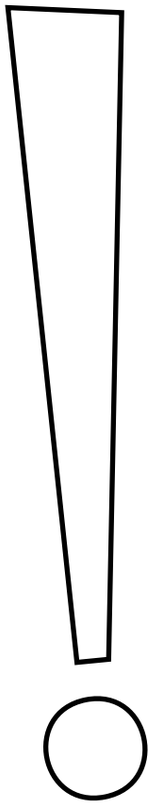


Picture provided by saintbob

Condensed Graph



Picture provided by saintbob

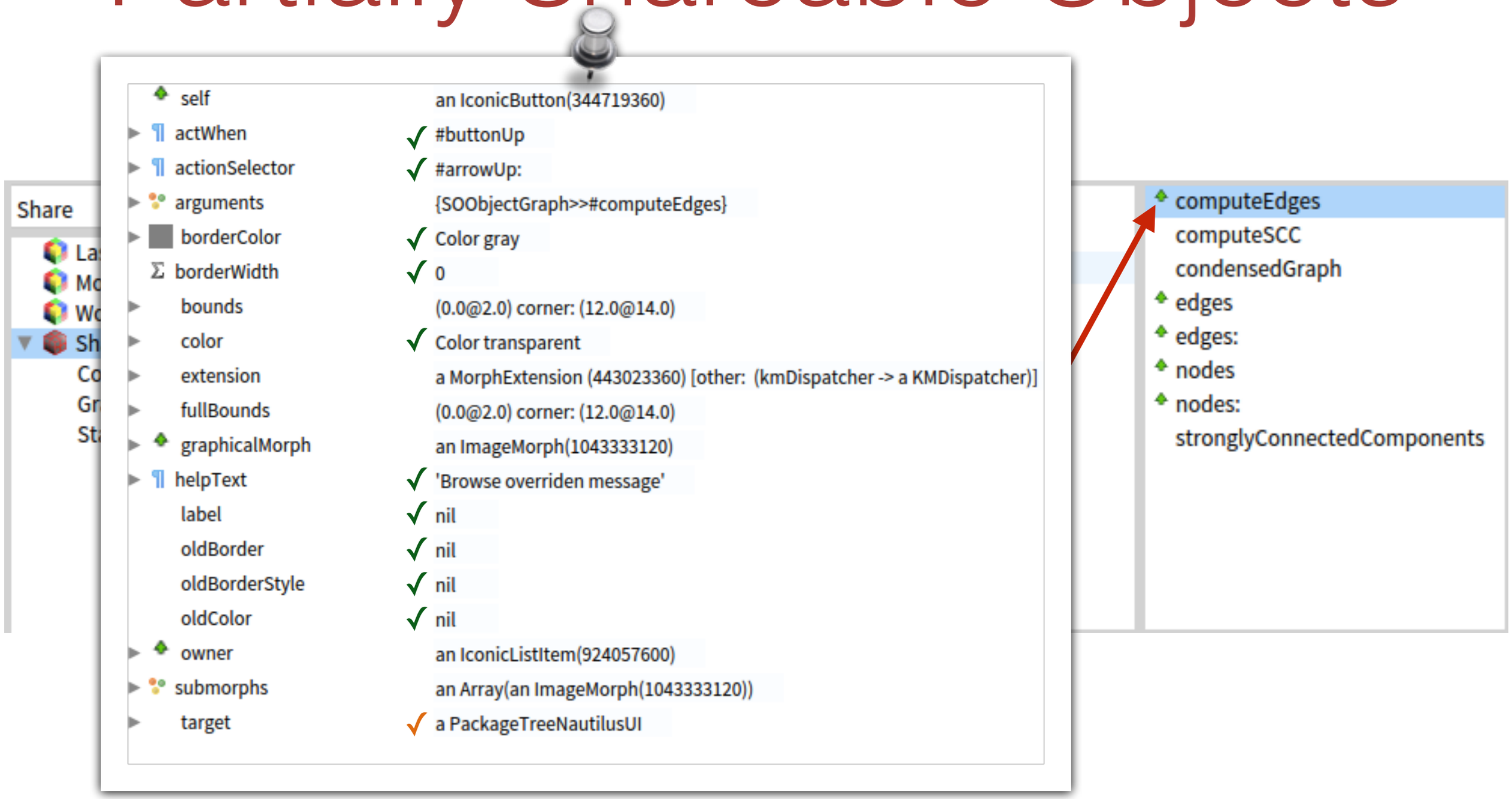


Reduce to acyclic graph (ignore the cycles)

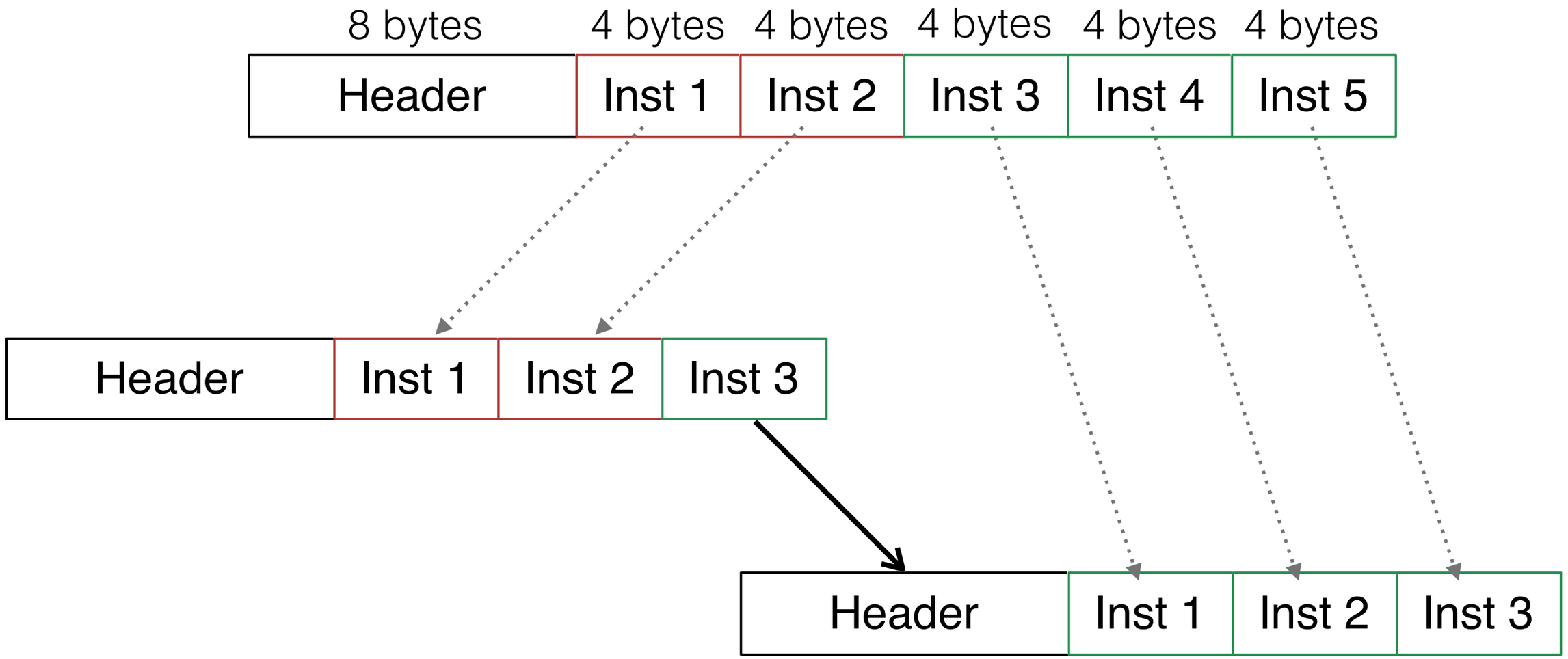
Results

	Objects	SCC	Avg SCC size	Shareable Sets	Savings	% Saving
Roassal	8.710	722	2,00	4	723	8.3%
Nautilus	144.287	530	4,19	205	982	0.7%
Famix	181.450	22.055	3,5	83	135	0.1%

Partially Shareable Objects



Partially Shareable Objects



For 1000 objects this reduce memory consumption by 29.5%

Partially Shareable Objects

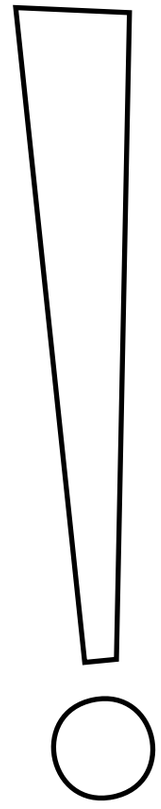
o1 ~ o2 are partially shareable if 2 or more instance variables of o1 are shareable with the instance variables of o2

Computing this is too expensive!

My Approach

Compute all possible values for each instance variable:

- How many possible values instance variable #a has?
- How is it related to the number of instances of the class?
- How many instance variables have less than 3 possible values for class A?



Results

	Instances	Inst Vars	Inst Vars <= 7	Inst Vars <= 3	Inst Vars = 1
RTSVGPath	169	8	6	6	6
MorphExtension	3567	15	8	6	2
MorphTreeNode Morph	428	17	9	9	8
TableLayoutProp erties	1.713	16	14	14	3

Conclusions

- Opportunities of optimization for fully shareable objects are not as frequent as we have thought
- Lots of arbitrary choices made in the research that are worth of discuss, or change to implement new heuristics
- Partially shareable objects promise interesting research opportunities
- The main problems about working with an object graph is Collections and Strongly Connected Components (cycles)

What have I learnt

- Try new possibilities when facing adverse results
- Need for a reliable Smalltalk benchmark for validating results
- Keeping track of objects history is an open problem, since it usually force us to choose between:
    - Efficiency
    - Quality of information (Level of detail)
    - Easiness to design and implement