



DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN  
UNIVERSIDAD DE CHILE

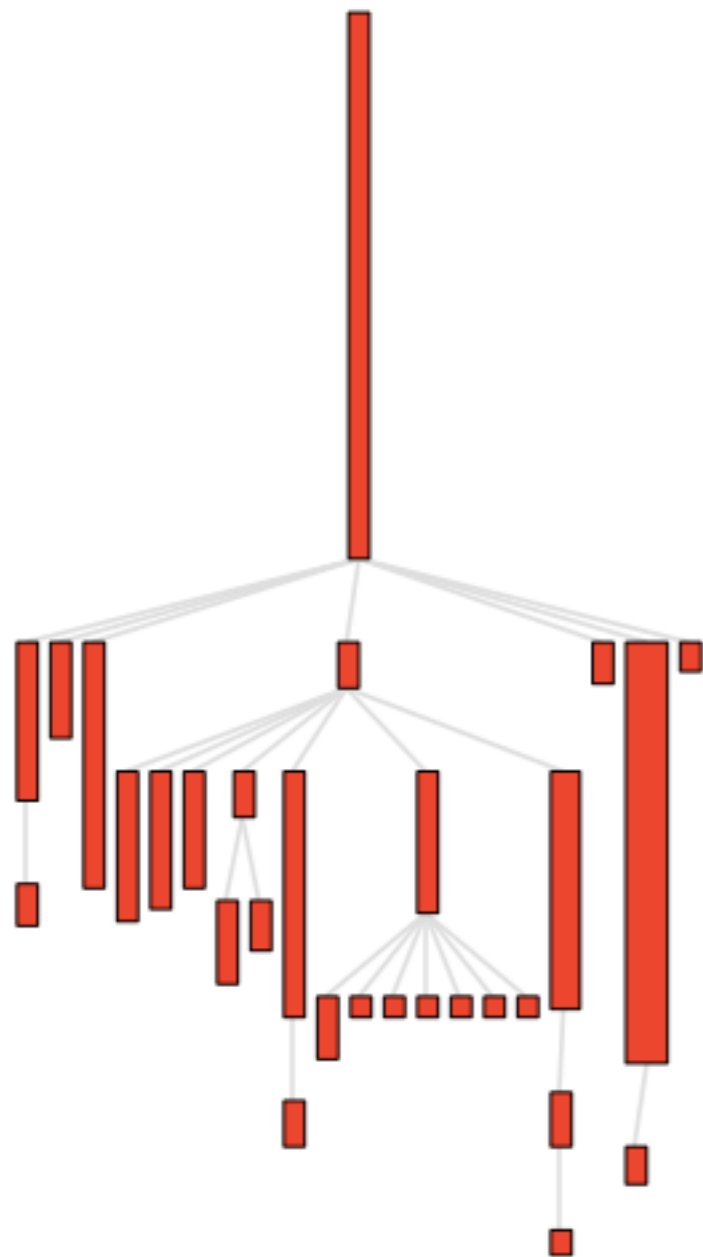
*u<sup>b</sup>*

---

<sup>b</sup>  
UNIVERSITÄT  
BERN

# Identifying Shareable Objects

Alejandro Infante  
[ainfante@dcc.uchile.cl](mailto:ainfante@dcc.uchile.cl)



| b |

b := RTMondrianViewBuilder new.

b shape rectangle

width: #numberOfVariables;

height: #numberOfMethods;

fillColor: [~~Color new r: 1 g: 0 b: 0~~] → ( Color new r: 1 g: 0 b: 0 )

b nodes: Collection withAllSubclasses.

b edges: Collection withAllSubclasses from: #superclass to: #yourself.

b treeLayout.

b build.

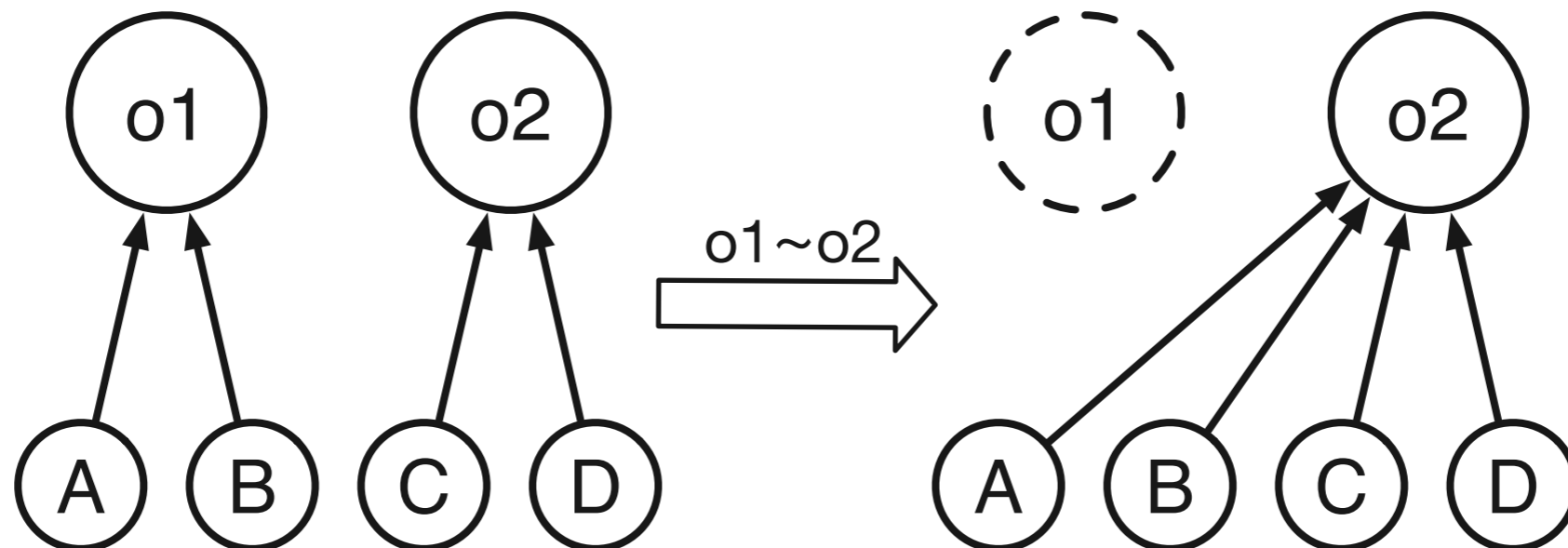
b view open.

# Related Work

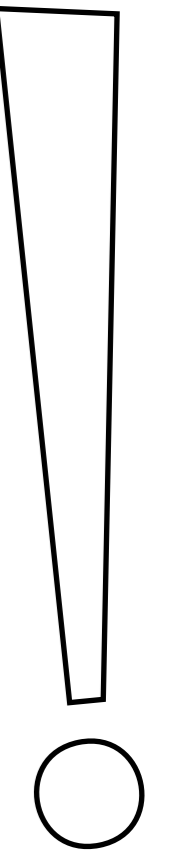
- Object Equality Profiling (*Marinov and O'Callahan*)
  - Take a snapshot of the object graph and record last time an object has been written or read to identify potential shareable objects
- Cachetor: Detecting Cacheable Data to Remove Bloat (*Nguyen and Xu*)
  - Use a novel technique called Value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

# Shareable Objects

$o1 \sim o2$  if all objects referencing  $o1$  can change their reference to  $o2$  preserving the program behavior.



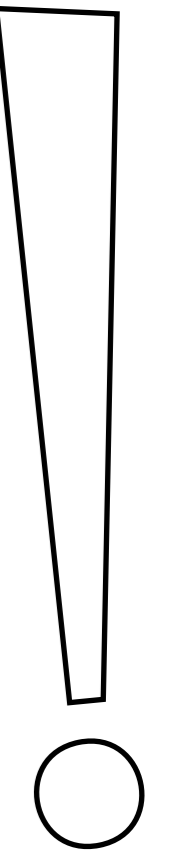
# Shareable Objects

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


# Defining Immutability

$o$  is immutable only if:

$$\begin{aligned} & \nexists f \in \text{instVars}(o) \\ & \exists t_1, t_2 \wedge t_1 < t_2 \\ & o.f_{t_1} \neq \text{nil} \wedge o.f_{t_1} \neq o.f_{t_2} \end{aligned}$$



Summarizing: Instance variables change only ONCE

# Dynamic Analysis

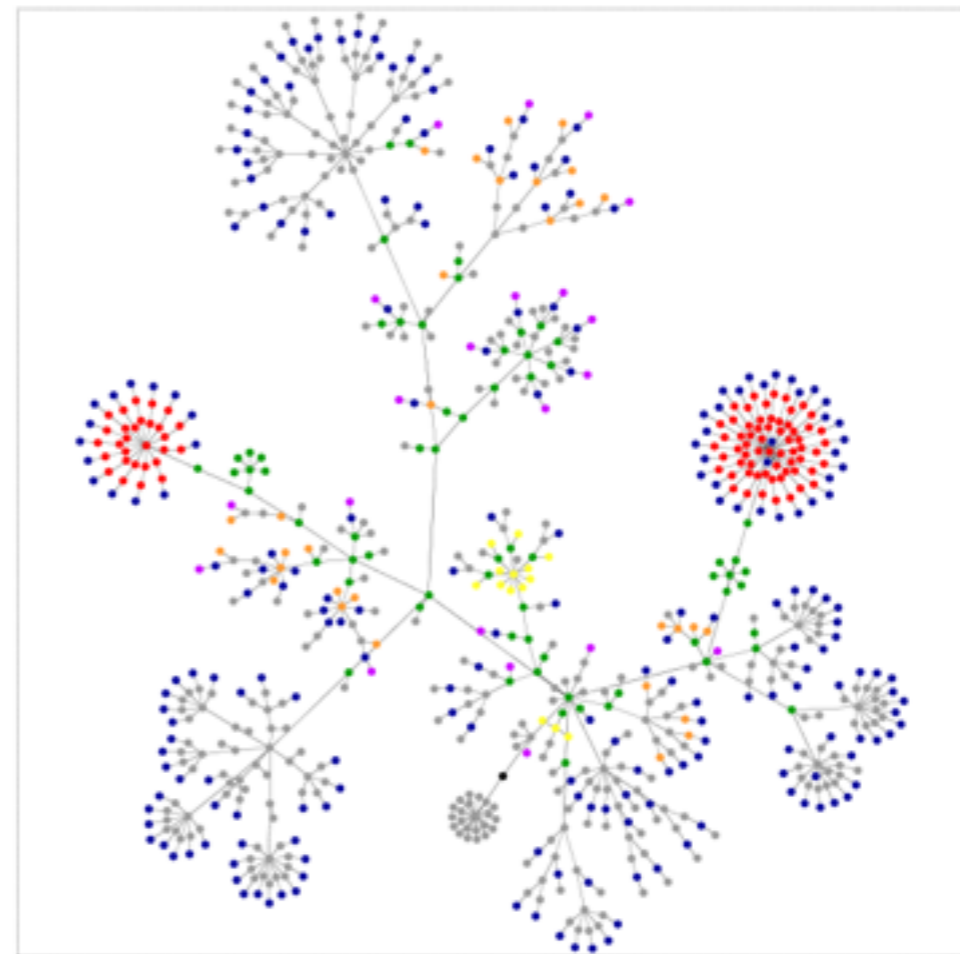
## Definition:

- ➔ Object
- ➔ Primitive Value
- ➔ External Classes
  - String\*
  - Empty Arrays\*

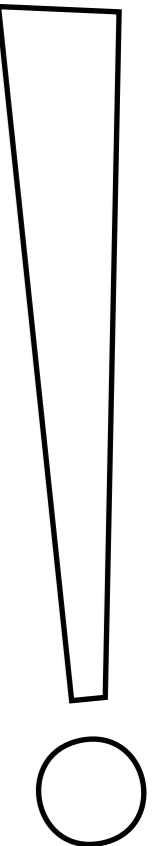
## Events:

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

## Object Graph in time T

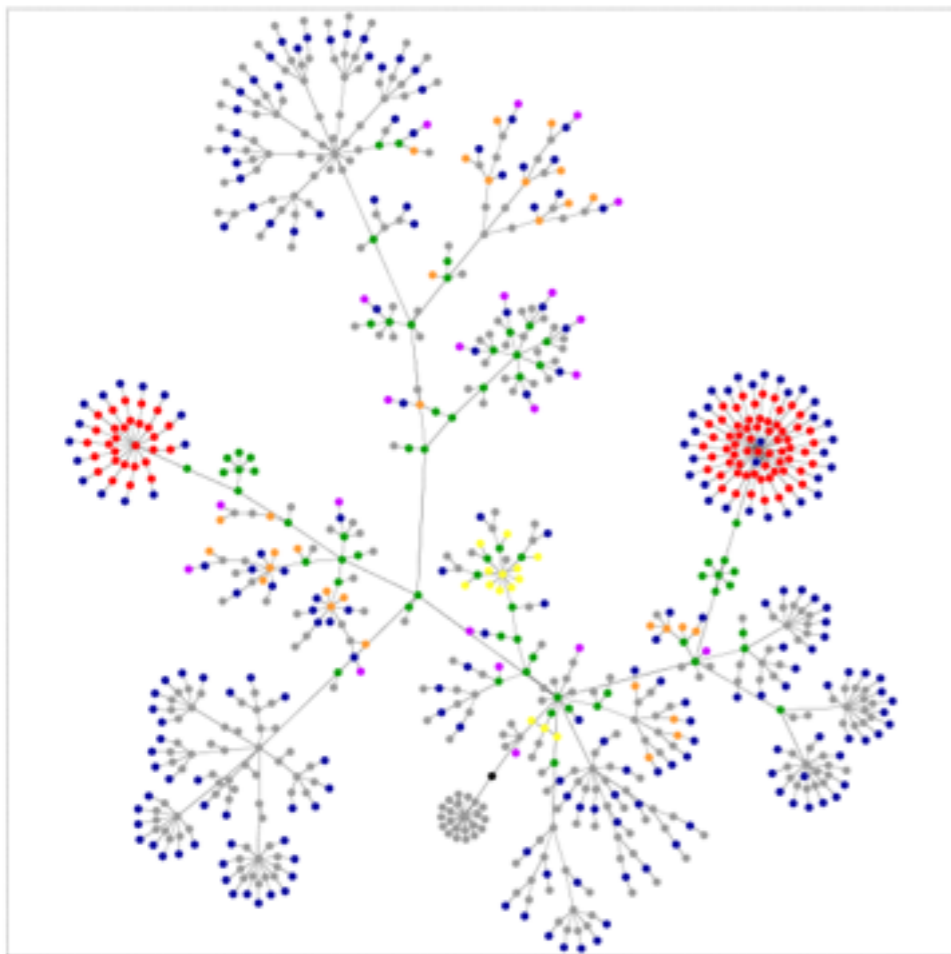


Picture provided by saintbob



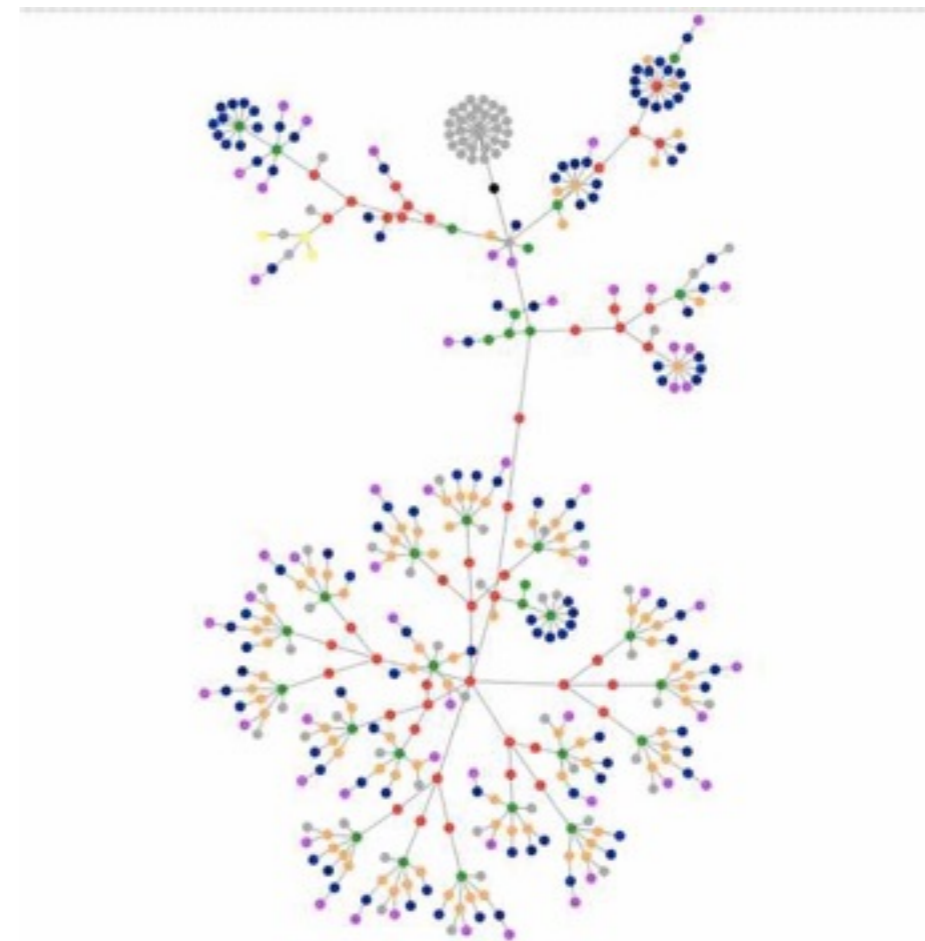
# Dynamic Analysis

Object Graph in time T



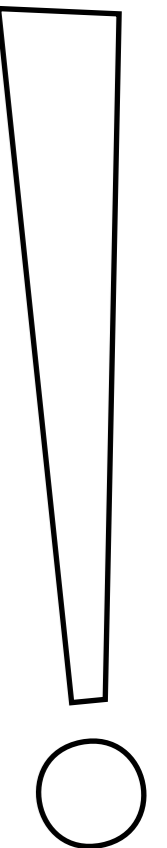
Picture provided by saintbob

Condensed Graph



Picture provided by saintbob

Reduce to acyclic graph (ignore the cycles)





# Results

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

# Partially Shareable Objects

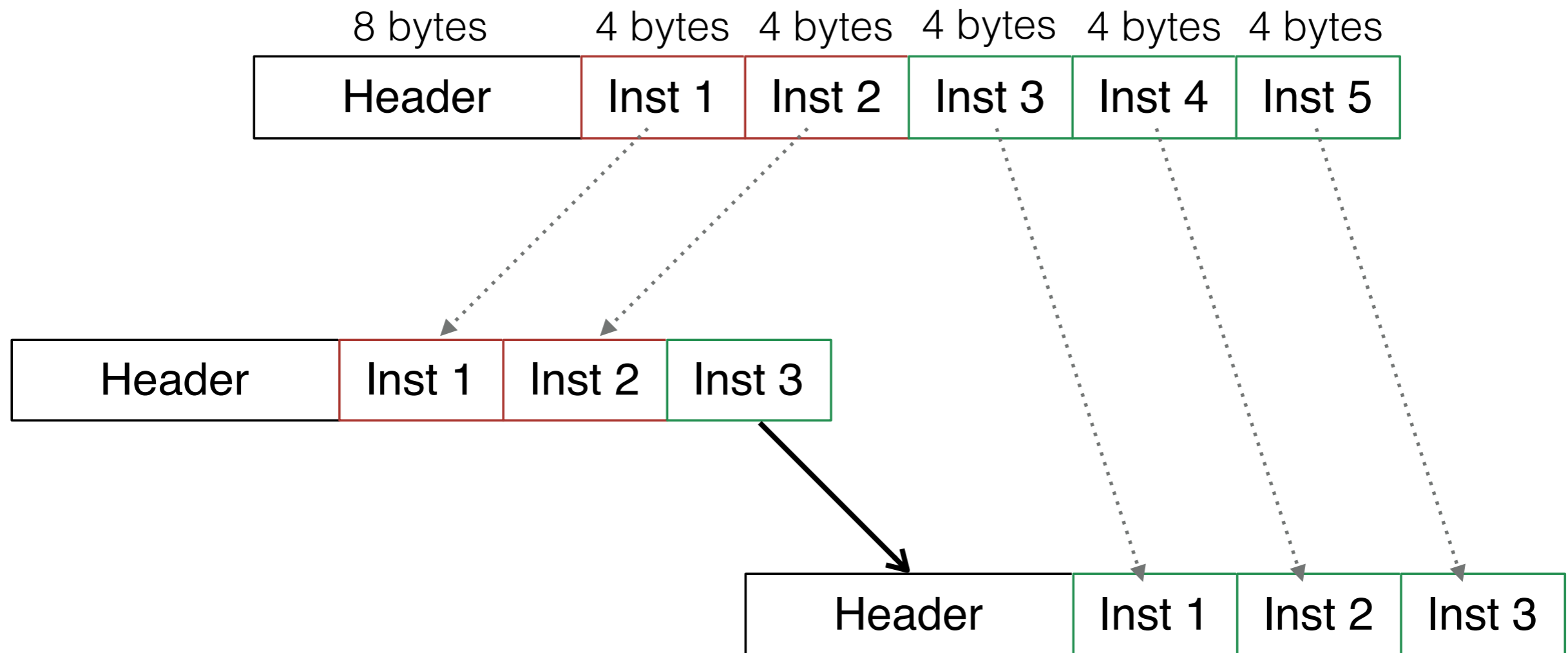
The image shows a software interface with two panels. The left panel displays a list of object attributes and their values, while the right panel shows a list of methods. A red arrow points from the 'computeEdges' method in the right panel to the 'arguments' field in the left panel.

Attribute	Value
self	an IconButton(344719360)
actWhen	✓ #buttonUp
actionSelector	✓ #arrowUp:
arguments	{SObjectGraph>>#computeEdges}
borderColor	✓ Color gray
borderWidth	✓ 0
bounds	(0.0@2.0) corner: (12.0@14.0)
color	✓ Color transparent
extension	a MorphExtension (443023360) [other: (kmDispatcher -> a KMDispatcher)]
fullBounds	(0.0@2.0) corner: (12.0@14.0)
graphicalMorph	an ImageMorph(1043333120)
helpText	✓ 'Browse overridden message'
label	✓ nil
oldBorder	✓ nil
oldBorderStyle	✓ nil
oldColor	✓ nil
owner	an IconicListItem(924057600)
submorphs	an Array(an ImageMorph(1043333120))
target	✓ a PackageTreeNautilusUI

Method List:

- computeEdges
- computeSCC
- condensedGraph
- edges
- edges:
- nodes
- nodes:
- stronglyConnectedComponents

# Partially Shareable Objects



For 1000 objects this reduce memory consumption by 29.5%

# Partially Shareable Objects

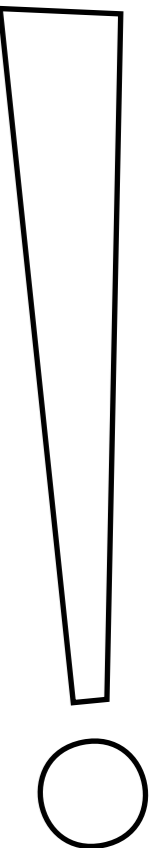
$o1 \sim o2$  are partially shareable if 2 or more instance variables of  $o1$  are shareable with the instance variables of  $o2$

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

	<b>Instances</b>	<b>Inst Vars</b>	<b>Inst Vars <math>\leq 7</math></b>	<b>Inst Vars <math>\leq 3</math></b>	<b>Inst Vars = 1</b>
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