

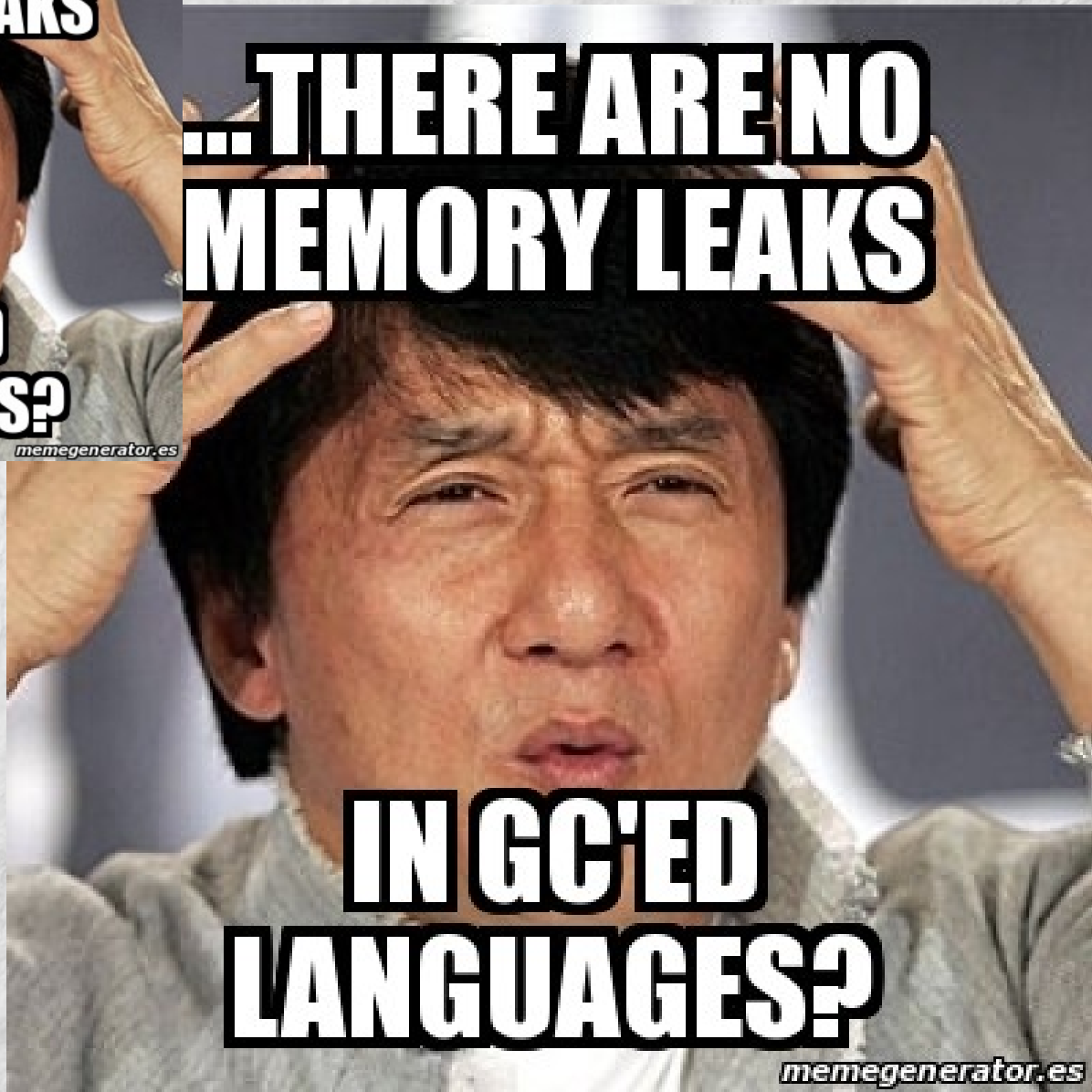
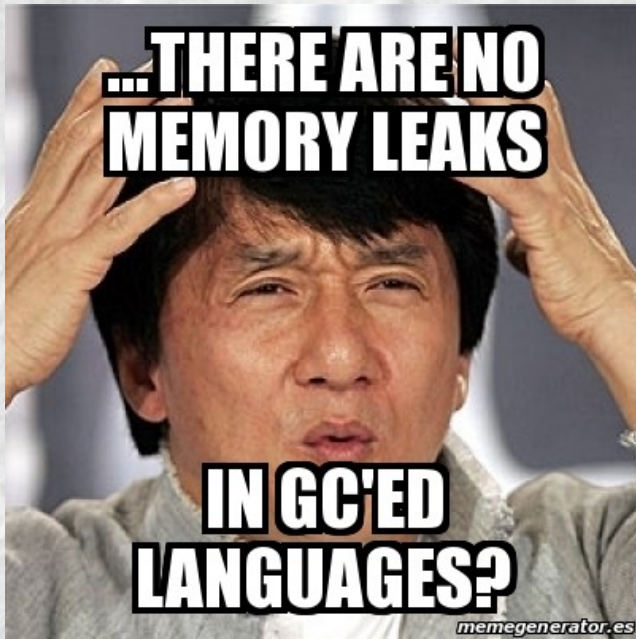


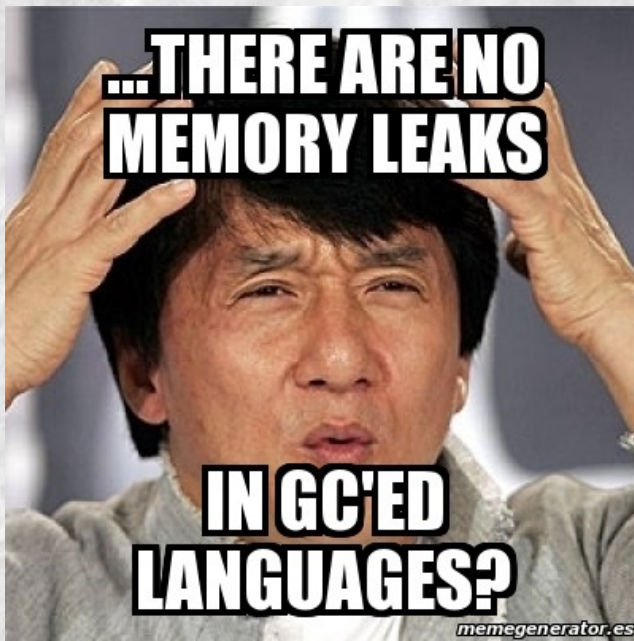
Weak Pharo Story



Smalltalk is a
GC'ed
language





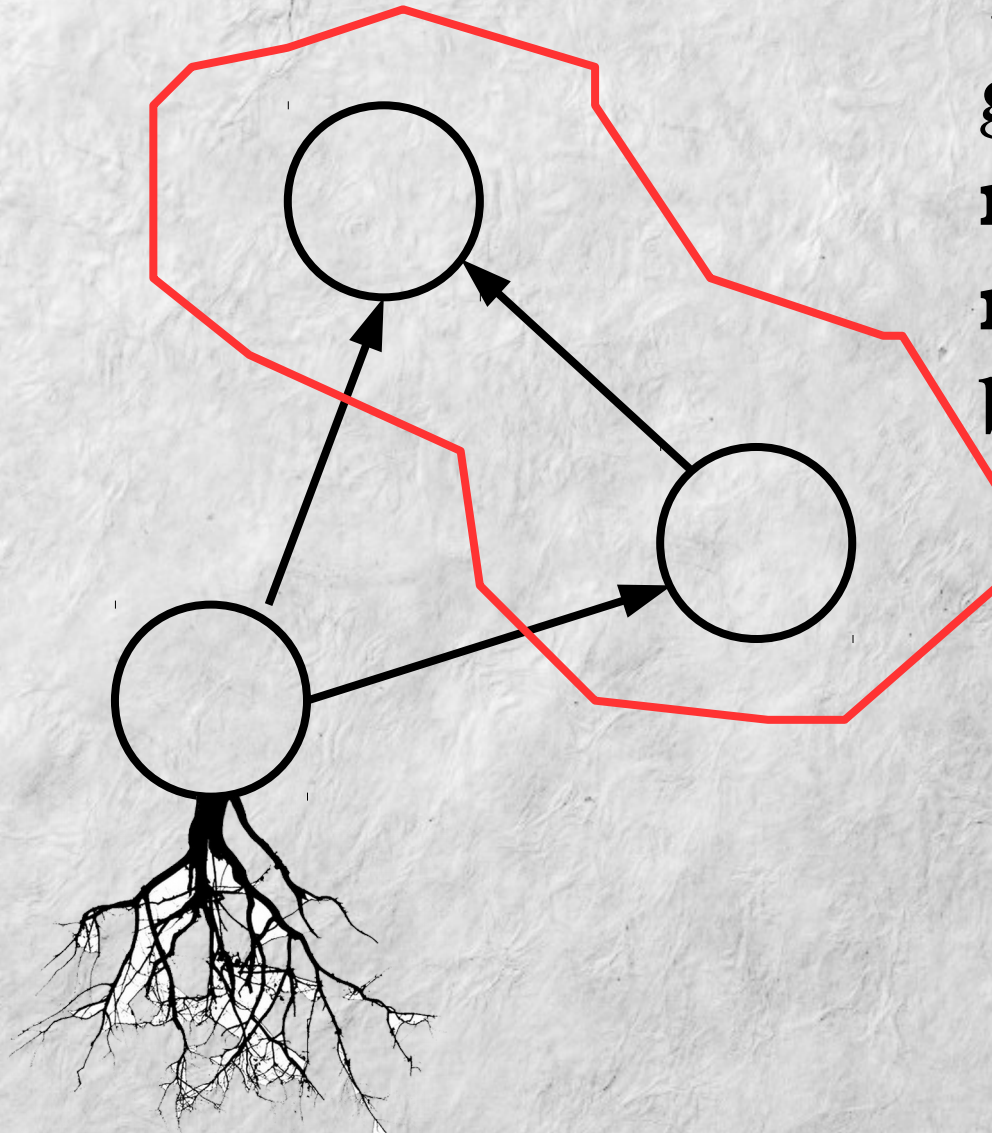


Two kind of leaks

- Leak **application objects**
e.g., your domain objects, collections...
- Leak **external objects**
e.g., sockets, files, memory allocated in C heap



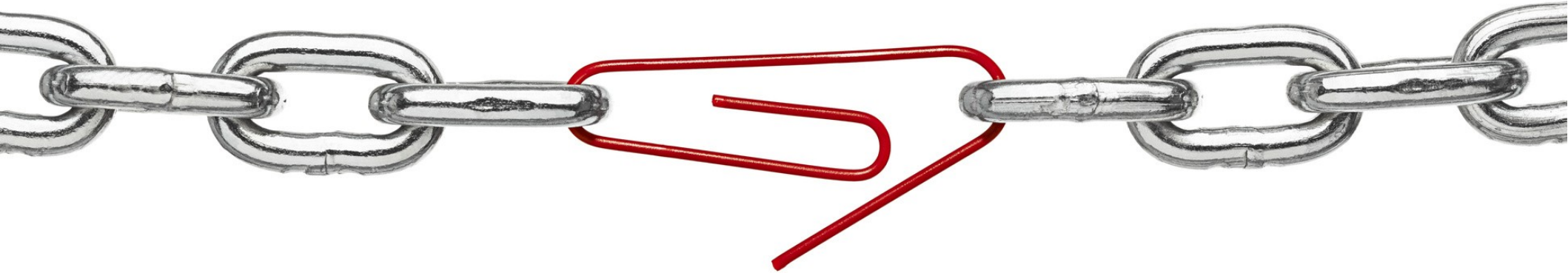
Root objects hold yours!



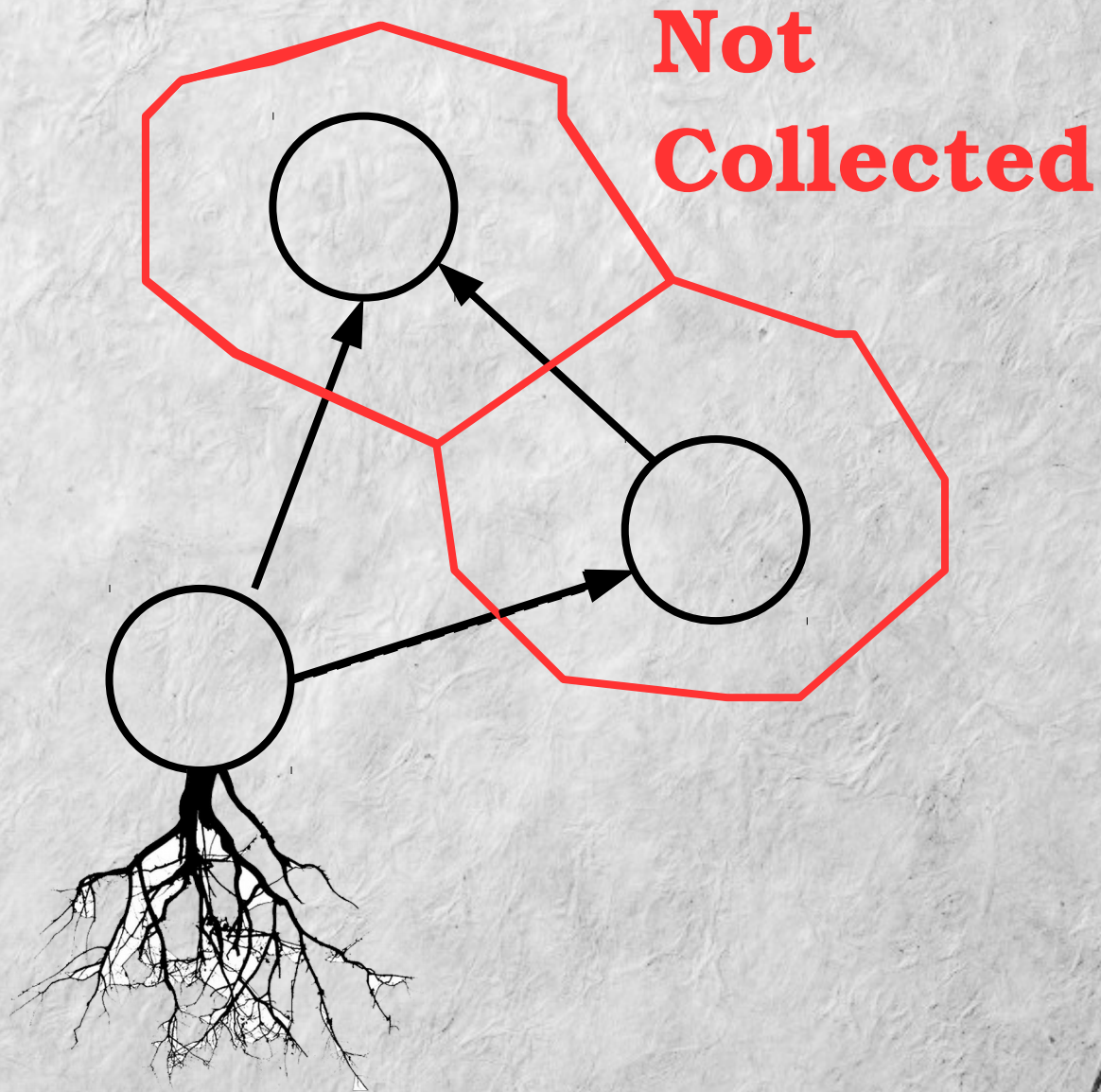
These two
guys **in the
red area** are
never going to
be **collected**



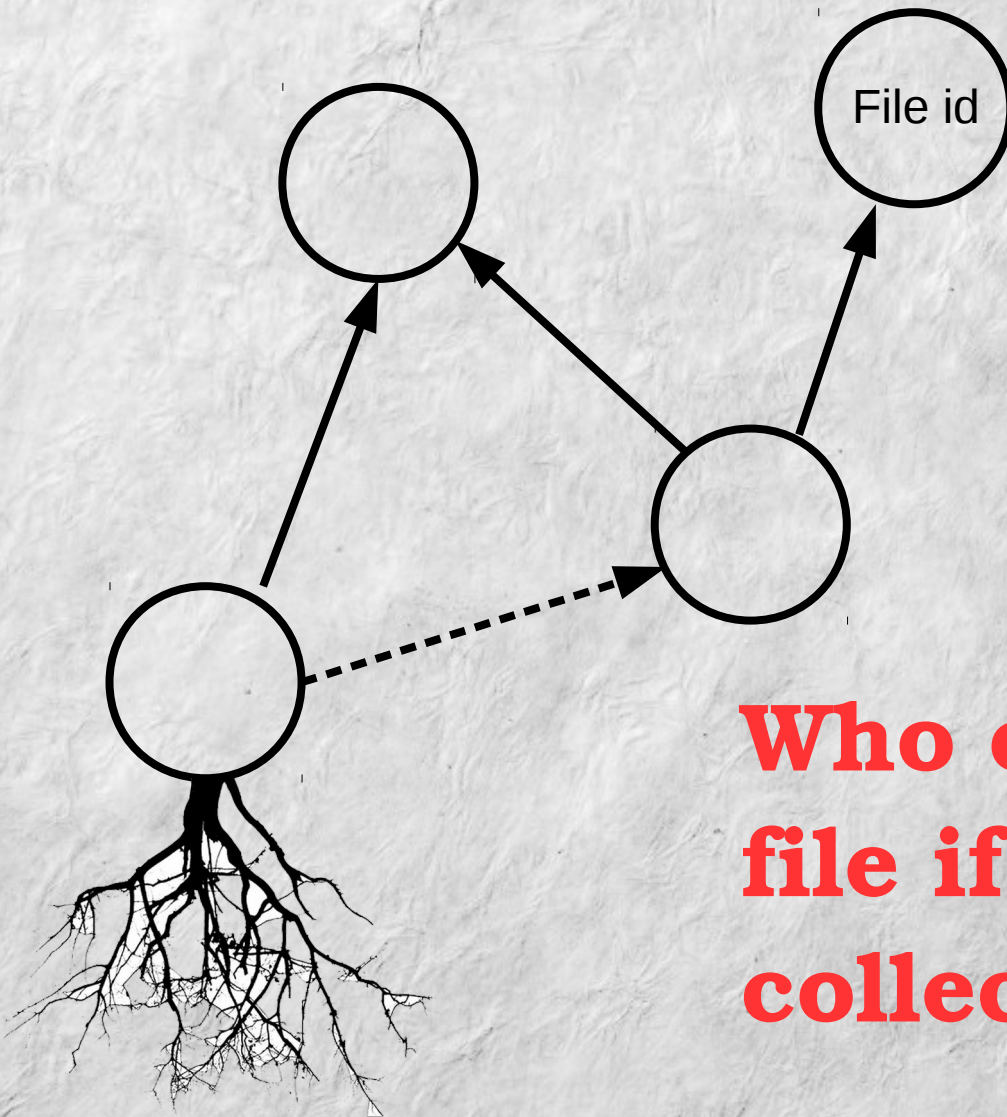
But... we have *Weak References*



Weak References In One Slide



What about external objects?



Who closes the file if it gets collected?



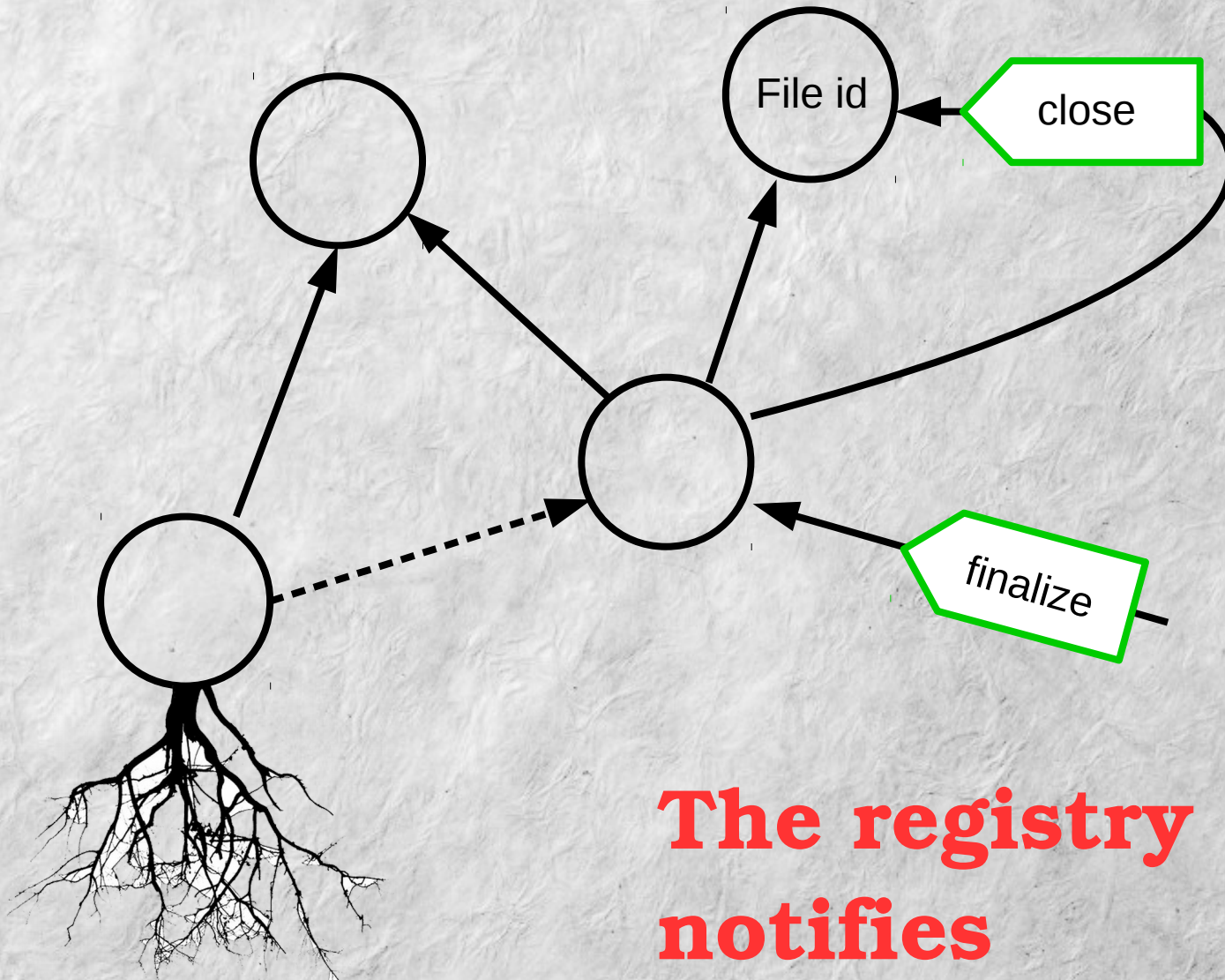
Finalization



There is a **registry** of
“Objects to be notified when about to be
collected”

WeakRegistry default add: theInterestedGuy

Object Finalization



The registry notifies

But **NONE** of it is

MAGIC
The Gathering®





[WARNING]

**The following images
can affect sensitive
people**

A close-up, low-angle shot of a dark, polished wooden door. A hand is pressed against the door, with fingers spread, as if trying to break through or reach out. The lighting is dramatic, highlighting the texture of the wood and the skin of the hand. The background is dark, making the door and hand the central focus.

No matter how weak
your references are

Memory Leaks
will find **you**

The Weak Pharo Story (finally)



nce upon a time, there was

Announcements, an event delivery library,
that the princess named *Engineer* used to
notify **myObject** from **anEvent**

```
announcer  
  when: anEvent  
  send: #message  
  to: myObject
```



The Weak Pharo Story (II)

But *Engineer* did love **myObject** so much that it did not want to retain it for ever. It did not want announcer to hold **myObject** strongly. She wanted a *weak announcer*.

```
announcer weak  
  when: anEvent  
  send: #message  
  to: myObject
```

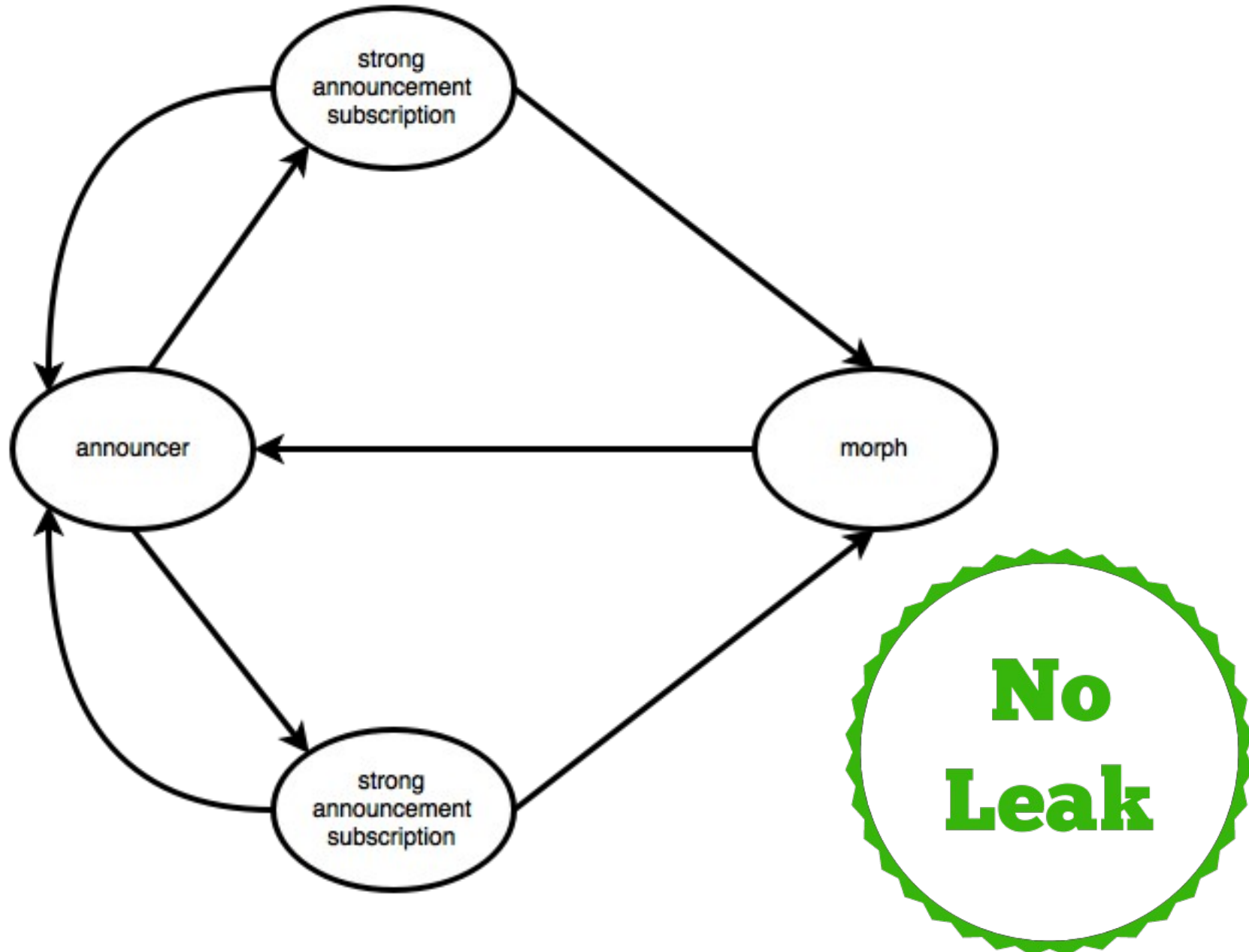
The Weak Pharo Story (III)

However, *Engineer* did not know this may *curse **myObject*** to be *alive for the eternity*. And never be collected and see his friends die. And create **OutOfMemory** errors on the land of objects to torment the rest of the objects.

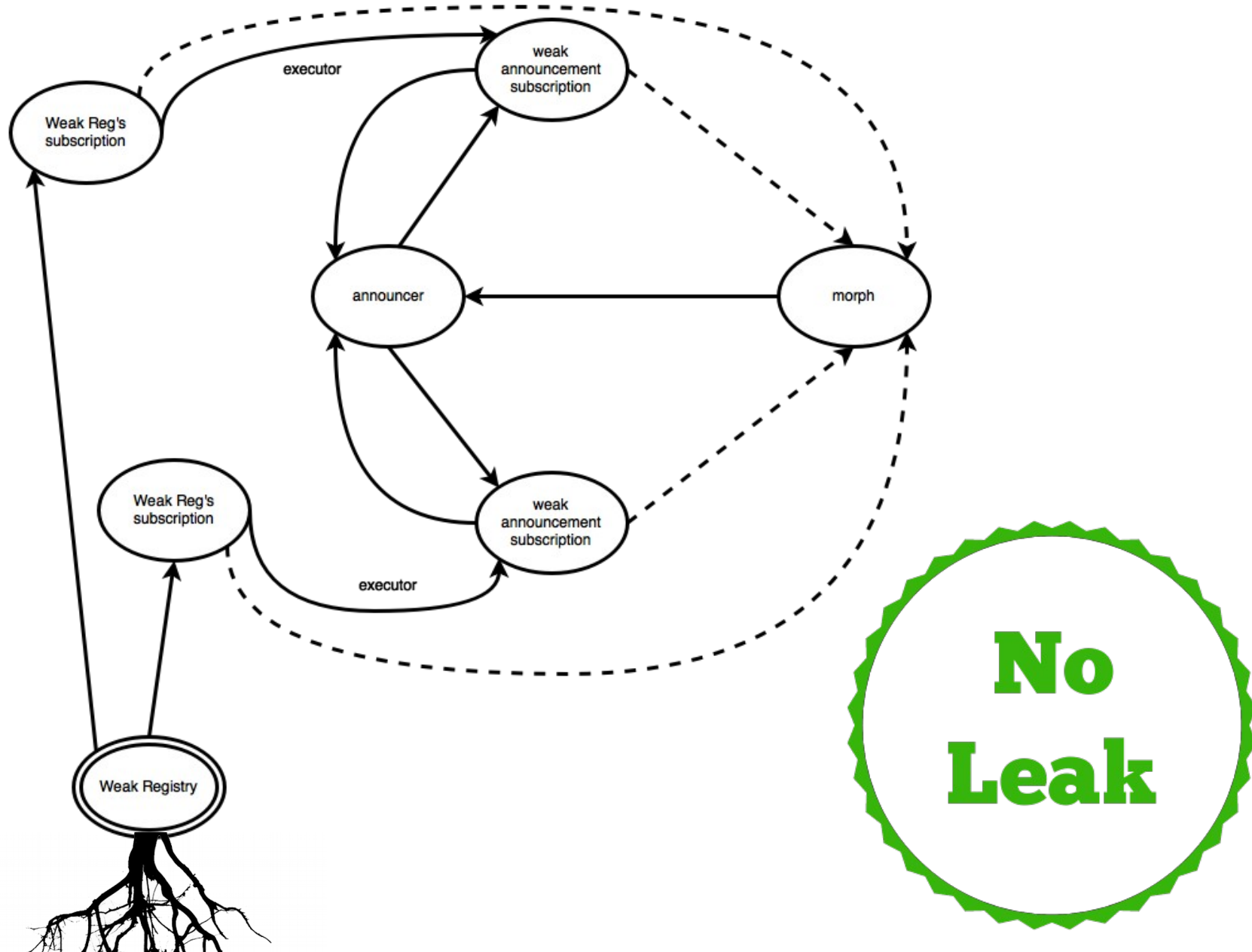
he end



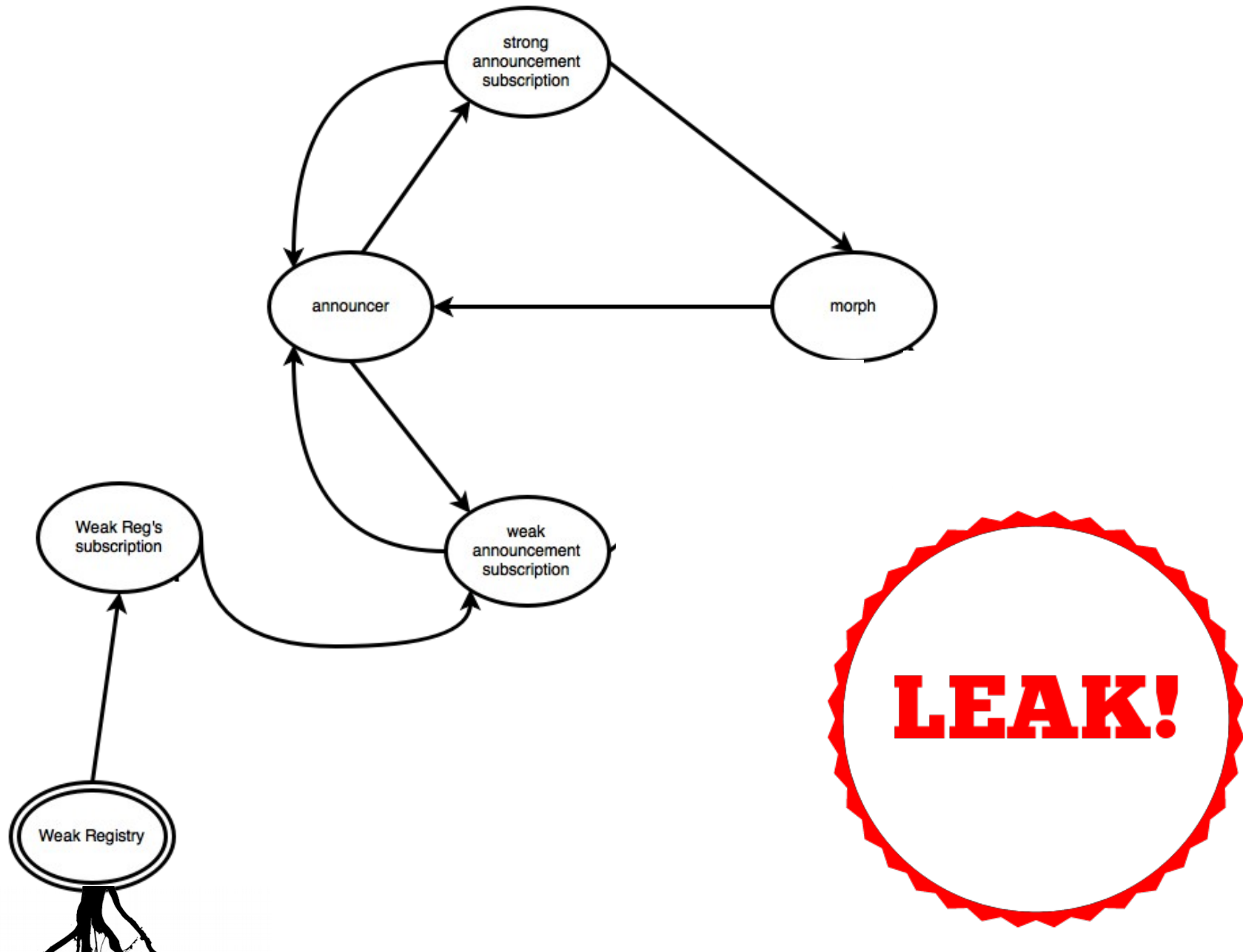
Case 1: The Strong Announcer



Case 2: The Weak Announcer



Case 3: The Hybrid Announcer



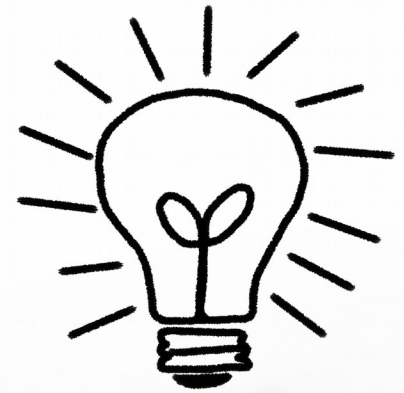
Autopsy

- Weak references do not simply avoid leaks!
- Finalization itself can create leaks!



So... solutions?

- 1) How do we detect leaks?
- 2) How do we prevent some?

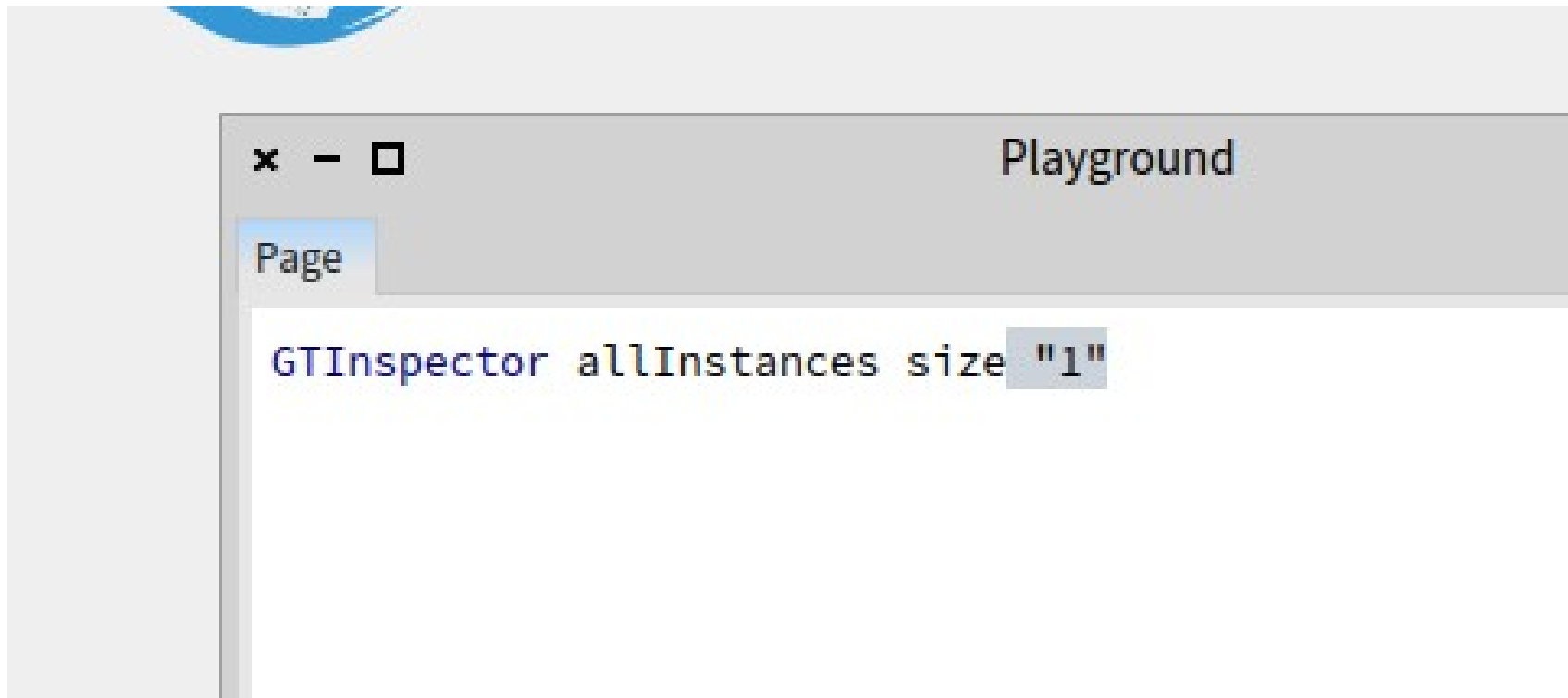


#1 - Detecting Leaks

Gotta catch 'em all!™



Memory leaks investigation



Why ???

anObject pointersTo

- Very inefficient

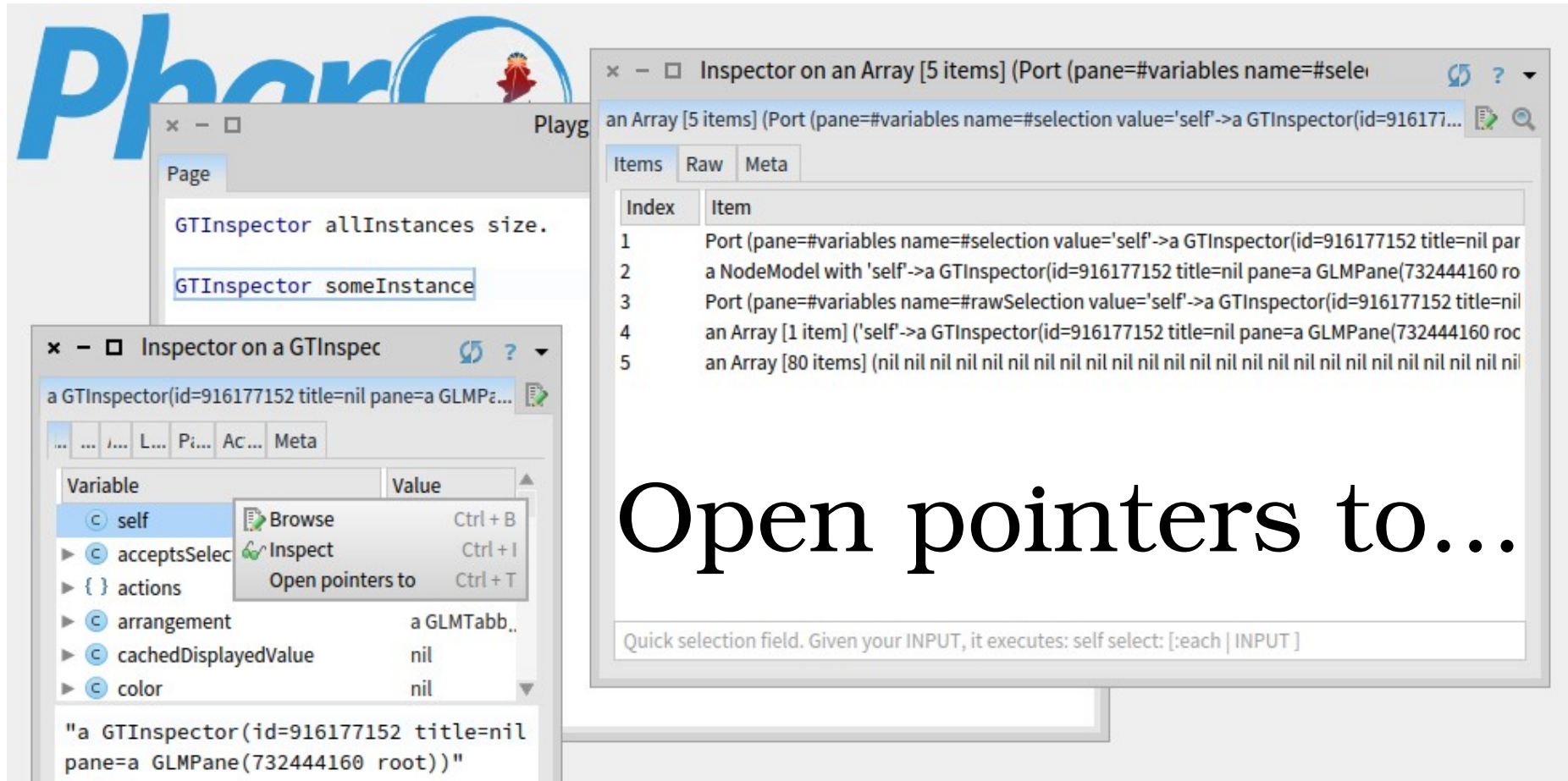
```
SystemNavigation default allObjectsDo: [:e |  
    (e pointersTo: self) ifTrue: [  
        pointers add: e ]].
```



Easy to get lost

anObject pointersTo first
pointersTo first pointersTo
second pointersTo last...

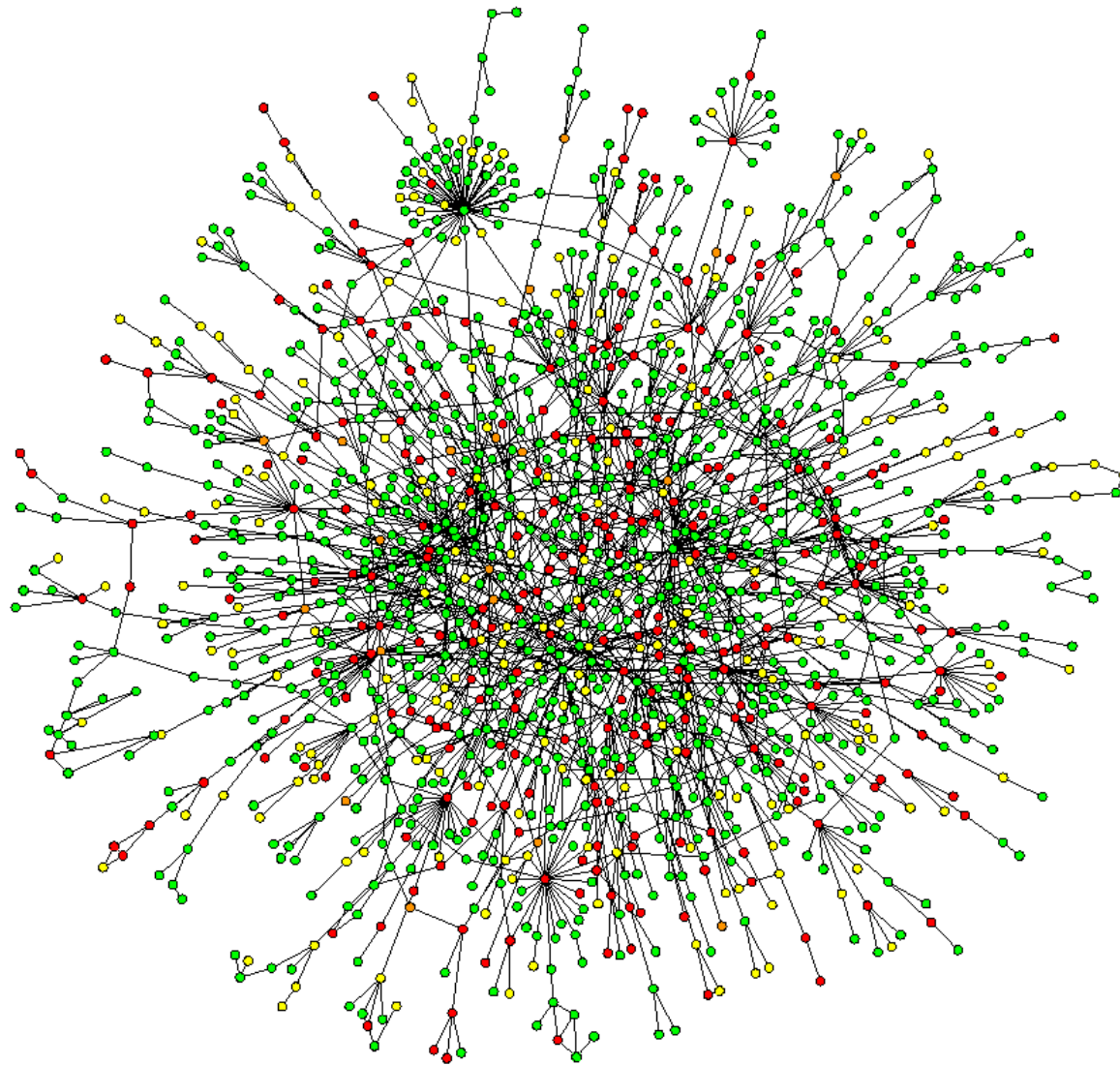




Open pointers to...

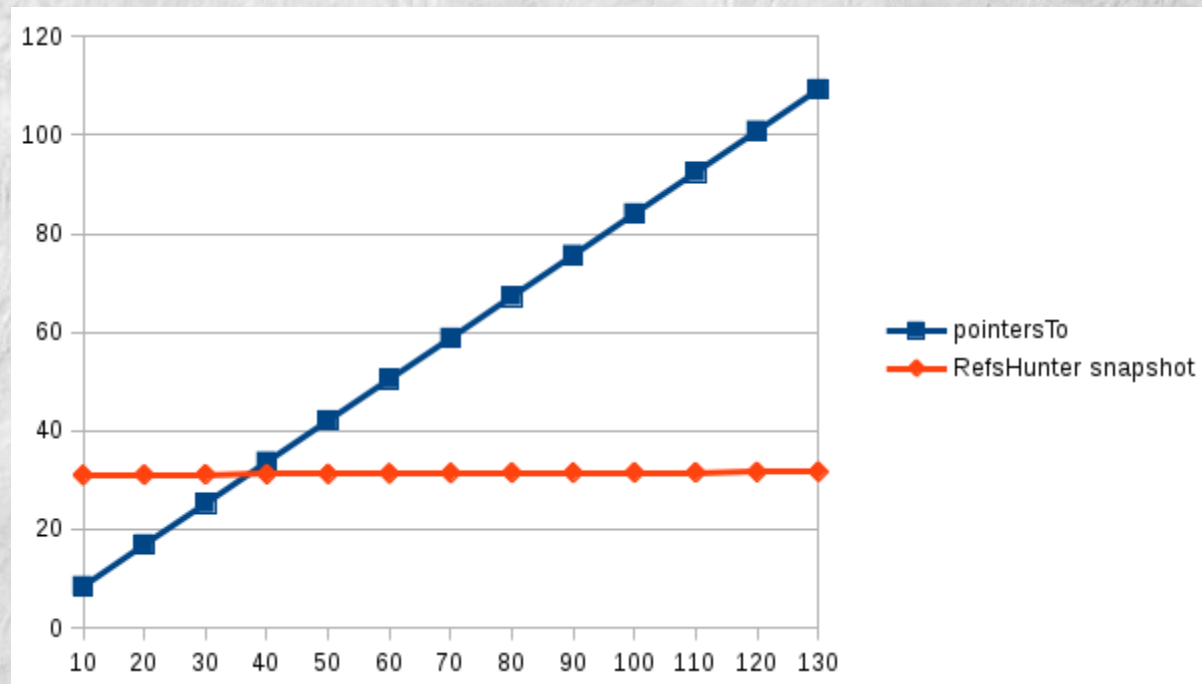
- Easy to use for simple cases
- Uses #pointersTo
- References from tools ► more mess

Hell of announcements and weak references



RefsHunter

- Temporary snapshot of the object memory



RefsHunter

- Shows the shortest path from one object to another
- Fast queries

```
rh := RefsHunter snapshot.  
rh wayFrom: (Array>>#asArray)  
to: Smalltalk specialObjectsArray.
```



RefsHunter

- Find references path to global space
- Easy to use
- No GUI
- Memory inefficient
 - more snapshots are not a good idea, really
- Download from the Catalog





Playground

Page

```
GTInspector allInstances size. "->1"  
  
rh := RefsHunter snapshot.  
rh wayFrom: GTInspector someInstance  
to: Smalltalk specialObjectsArray.
```

Inspector on an OrderedCollection [111 items] (a GTInspector(id=659781888 title=nil pane=)

an OrderedCollection [111 items] (a GTInspector(id=659781888 title=nil pane=)

a HandMorph(1017218304)

Items Raw Meta

Index	Item
1	a GTInspector(id=659781888 title=nil pane=a GLM
2	GTInspector>>compose
3	[:browser browser fixedSizePanels: self class nu
4	[:browser browser fixedSizePanels: self class nu
5	[:a :each a title: [self printObjectAsAnItem: each
6	a GLMReplacePresentationsStrategy
7	an Array [0 items] ()->nil
8	a GLMPager(id=237737728 title=nil pane=a GLMP:)
9	a GLMPane(424122624 1)
10	a GLMCompositePresentation(id=754934272 title:
11	GLMMorphicTabbedRenderer>>render:
12	[:each tabs addLazyPage: [self renderObject: e
13	[:each tabs addLazyPage: [self renderObject: e
14	[self renderObject: each]
15	a LazyTabPage(762999552)
16	an Array [10 items] (a LazyTabPage(762999552) ni
17	an OrderedCollection [1 item] (a LazyTabPage(76:
18	a LazyTabGroupMorph(818890752)
19	a PanelMorph(891969280)
20	a GLMTabSelectorBrick(411610368)
21	(Pharo3TabPanelBorder width: 1 color: (Color r: 0
22	a MorphExtension (20364800) [sticky] [other: (ro
23	a PanelMorph(486595840)
24	a RubScrolledTextMorph(778340096)
25	an Array [14 items] (a RubEditingArea(599357952)
26	a MouseOverHandler
27	a HandMorph(1017218304)

50 / 111

Quick selection field. Given your INPUT, it executes: self select

Raw Extens... Morph Meta

Variable	Value
▶ {} lastEventBuffer	an Array [8 items] l..
▶ Σ lastKeyScanCode	31
▶ C lastMouseEvent	[(433@444) mouse..
▶ C lastSystemEvent	nil
▶ C mouseClickedState	a MouseClickState..
▶ C mouseFocus	nil
▶ C mouseListeners	nil
▶ C mouseOverHandler	a MouseOverHand..
▶ owner	a WorldMorph(562..
▶ Σ recentModifiers	0
▶ C savedPatch	nil
▶ {} submorphs	an Array [0 items] ()
▶ targetOffset	(102.0@221.0)

"a HandMorph(1017218304)"
self

Avoid memory leaks

- Memory leak tests
 - Time consuming for basic Pharo image



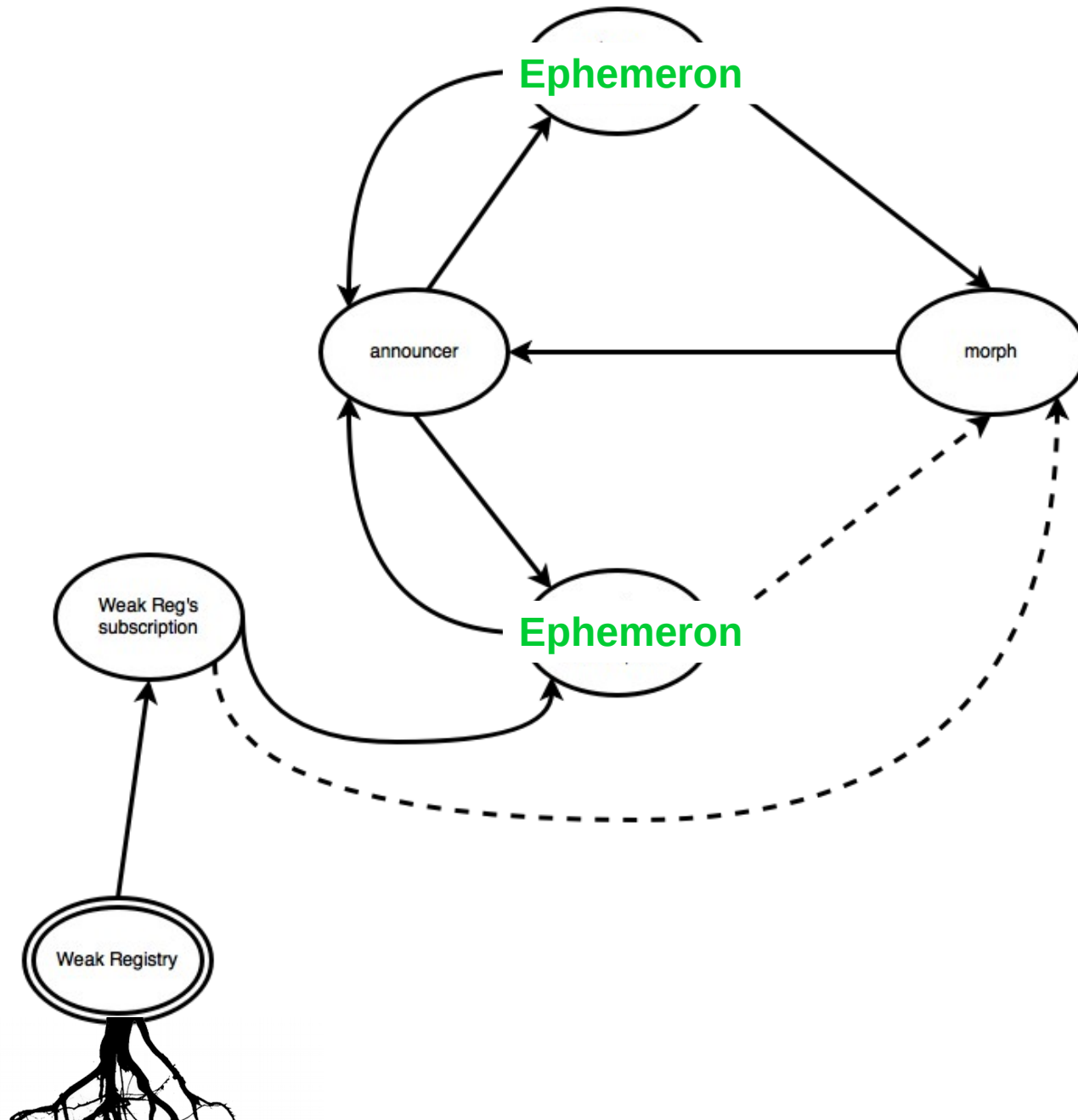
#2 – Avoiding Leaks

Ephemeron Finalization

- Ephemerons are special objects used for finalization
- They do not create leaks by themselves (as the WeakRegistry did)
- Soon in Pharo 6.0



Ephemeronons in Case 3



Lessons learned

- Announcements are sometimes overused
- Crazy leaking objects in the image
 - some tools opened in past during manual integration referenced by active hand click state
- Not every leak last forever
 - it takes 30 seconds to garbage collect closed Nautilus
- We need better tools support



Conclusions

- Weak references are nice
- But they are not magical

You can still create memory leaks with them

- Ephemerons will fix it partially

But you still need to know what you're doing a bit...

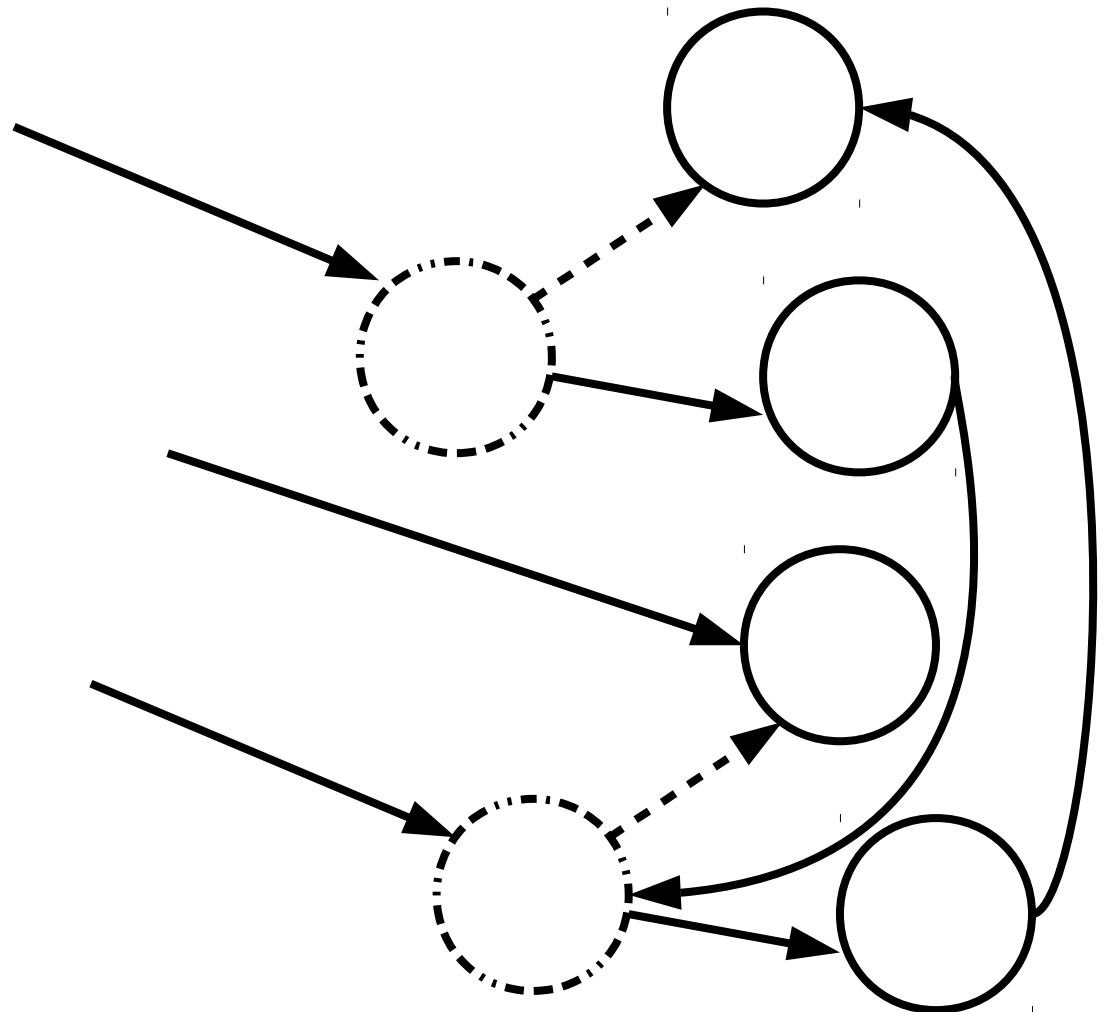
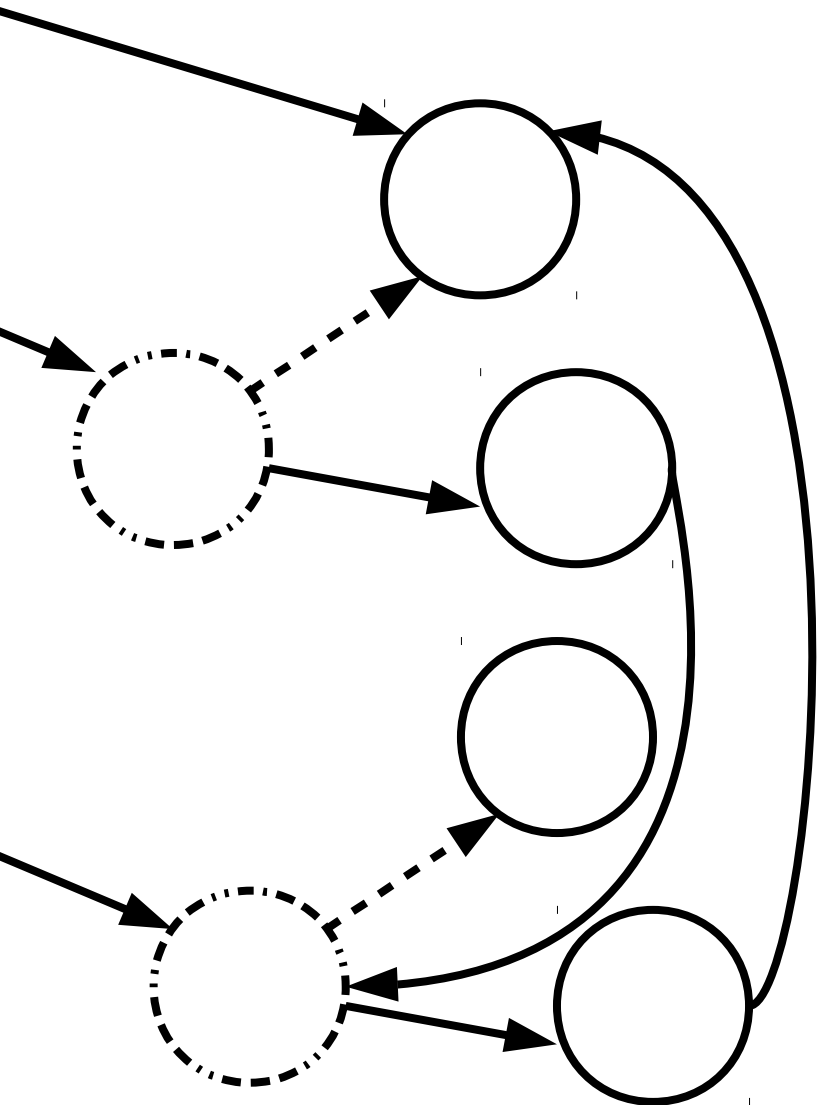
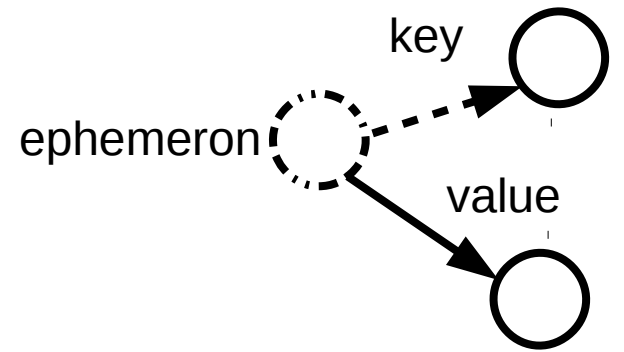


The End

or questions?



3.4 Exercises:



3.4 Ephemeron – Operational View

- When the GC passes...
- 1) It does **not** traverse ephemeron:
It queues them
- 2) Then
 - [traverses ephemeron whose key is referenced]
 - whileTrue: [
 there are ephemeron with keys referenced]

3.4 Exercises:

