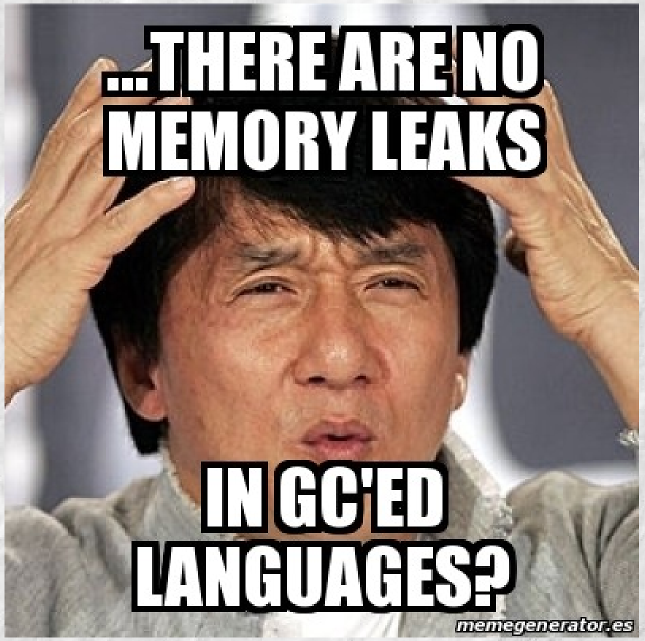
A Weak Pharo Story

Pavel Krivanek & Guille Polito

Smalltalk is a **GC'ed** language

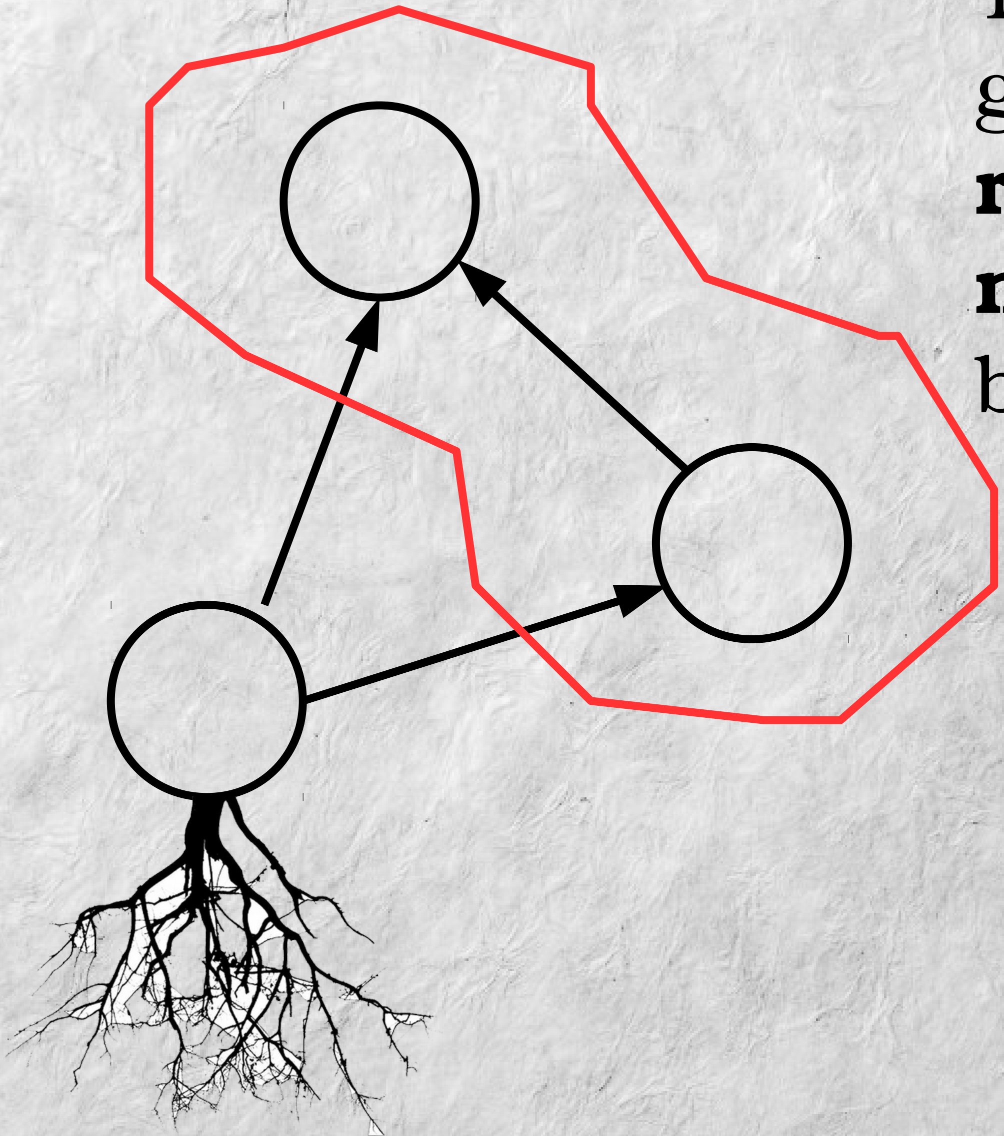


Two kind of leaks

- Leak **application objects**
  e.g., your domain objects, collections...

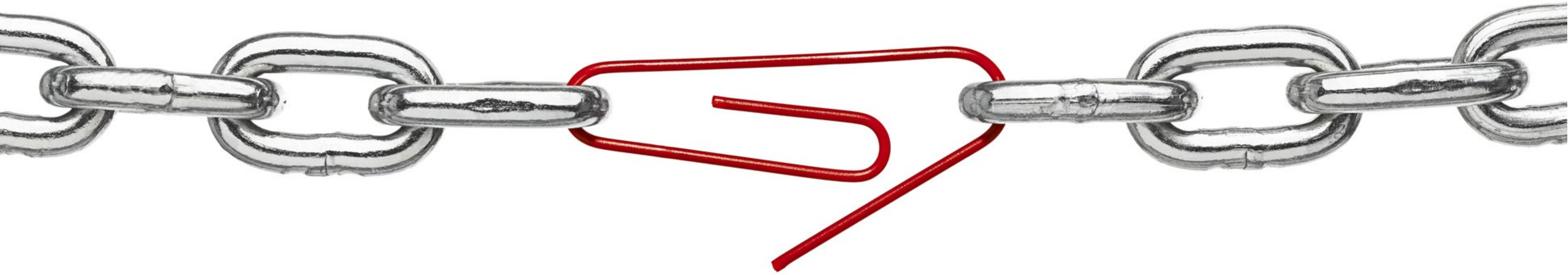
- Leak **external objects**
  e.g., sockets, files, memory allocated in C heap

Root objects hold yours!

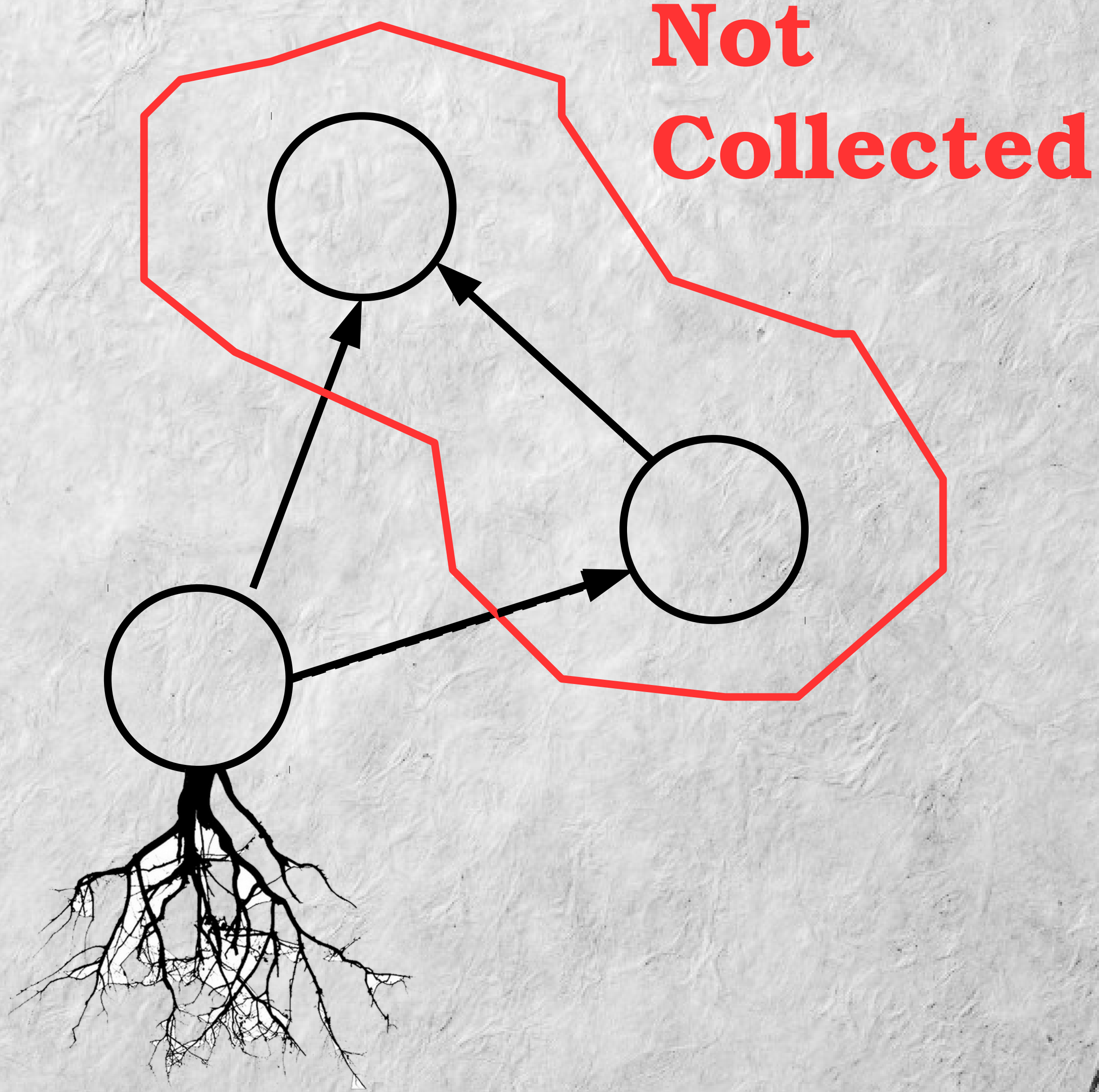


These two guys **in the red area** are **never** going to be **collected**

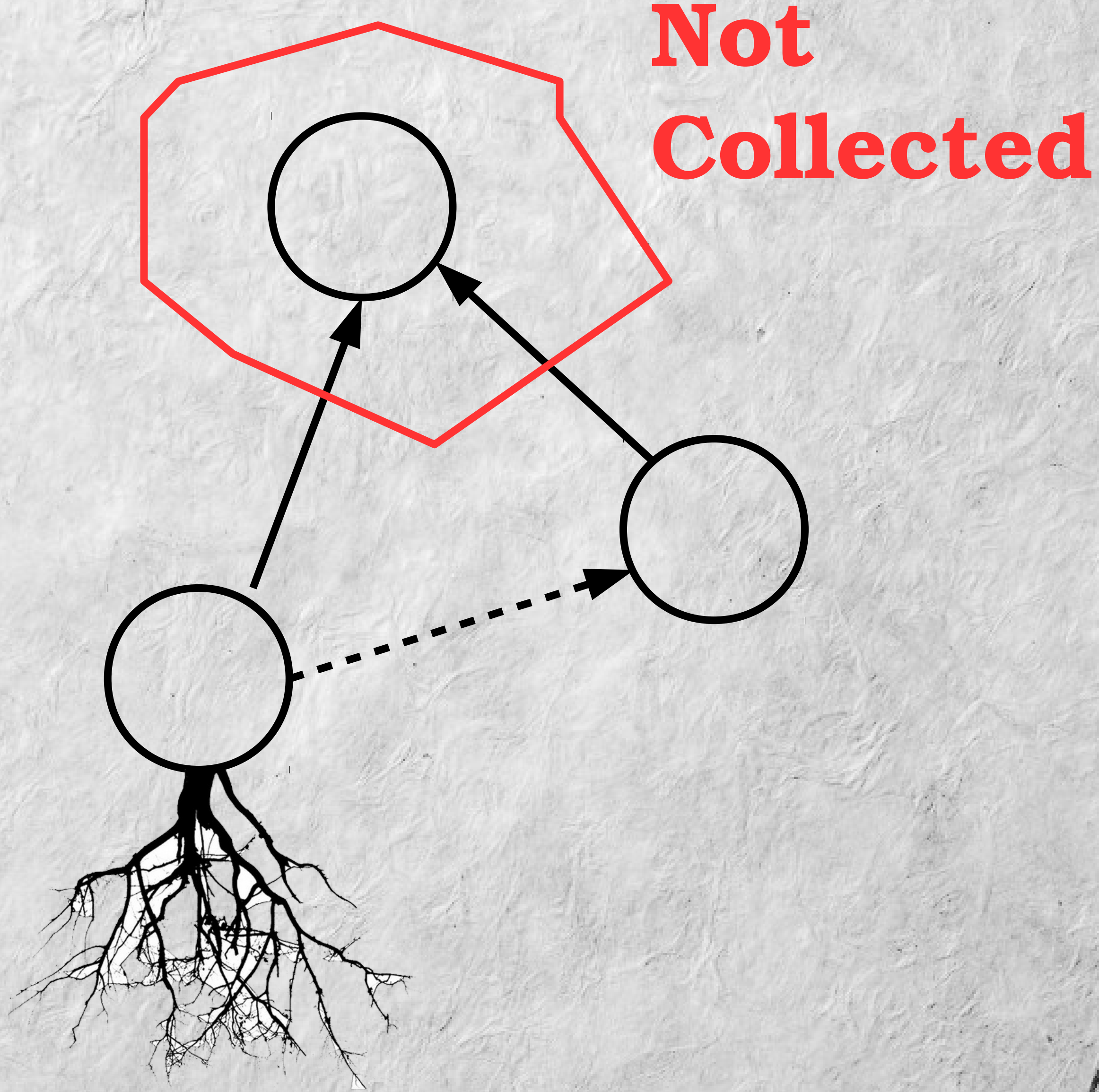
But… we have *Weak References*



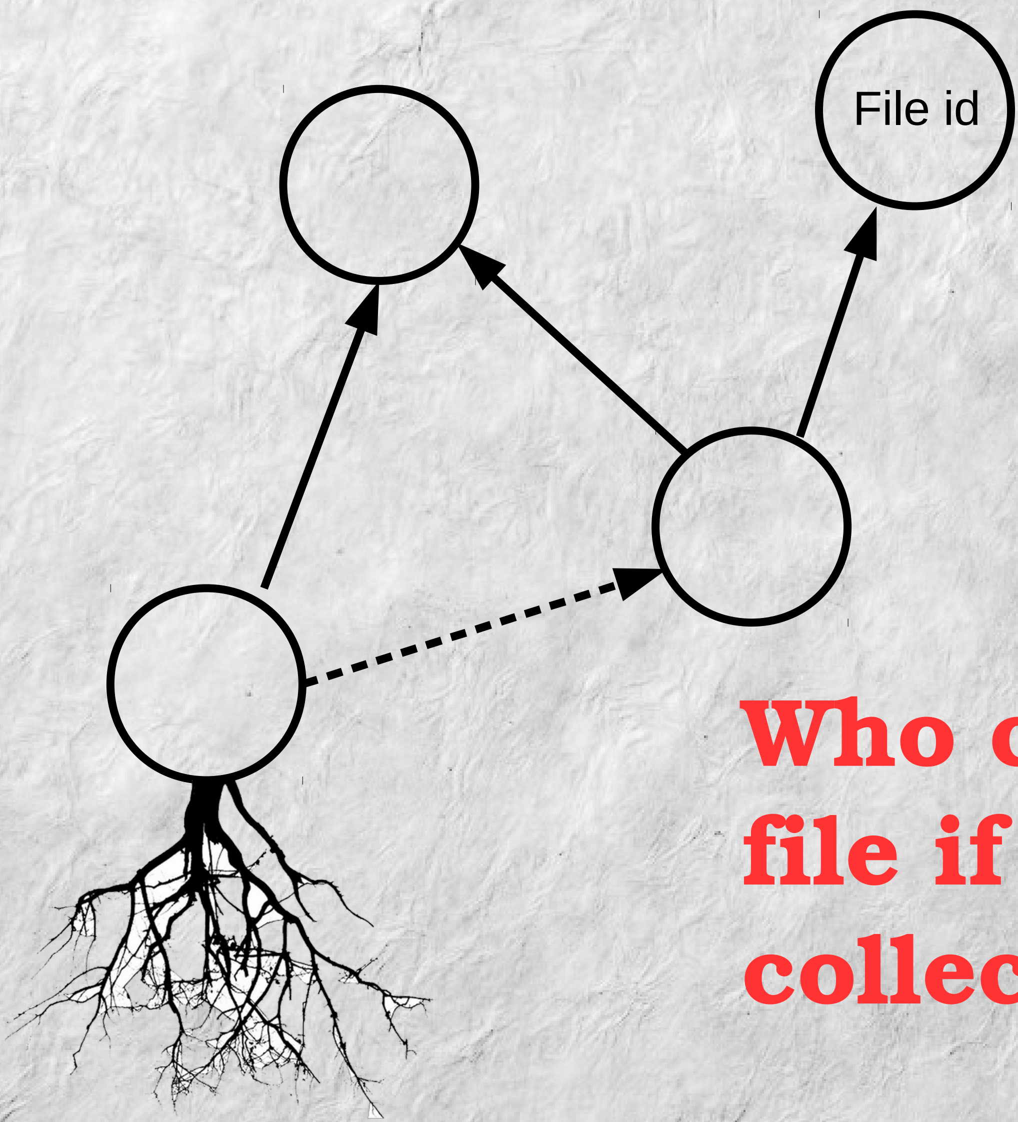
Weak References In One Slide



Weak References In One Slide



What about external objects?



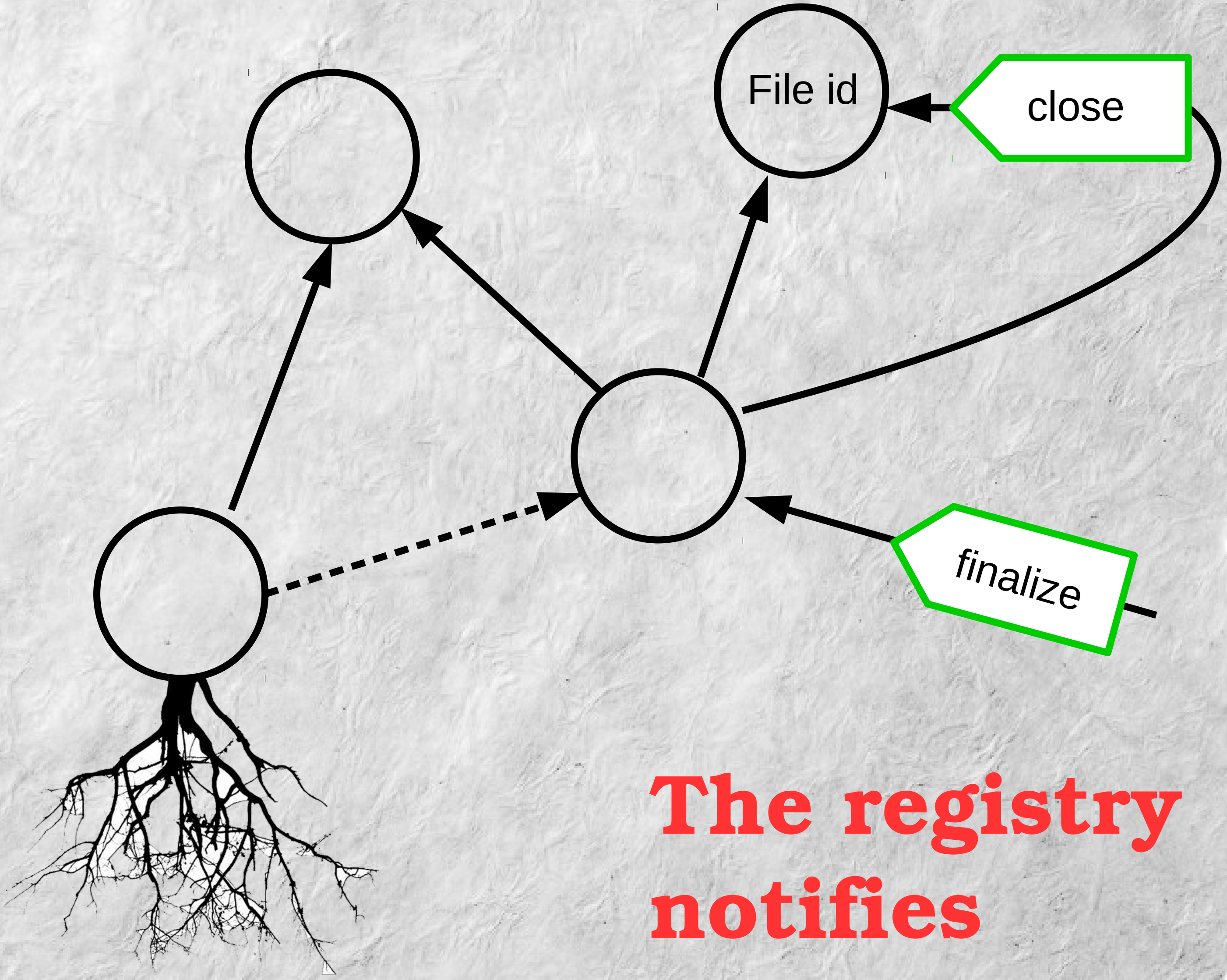
Who closes the file if it gets collected?

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

No matter how weak your references are

Memory Leaks will find you

The Weak Pharo Story (finally)

Once 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*.

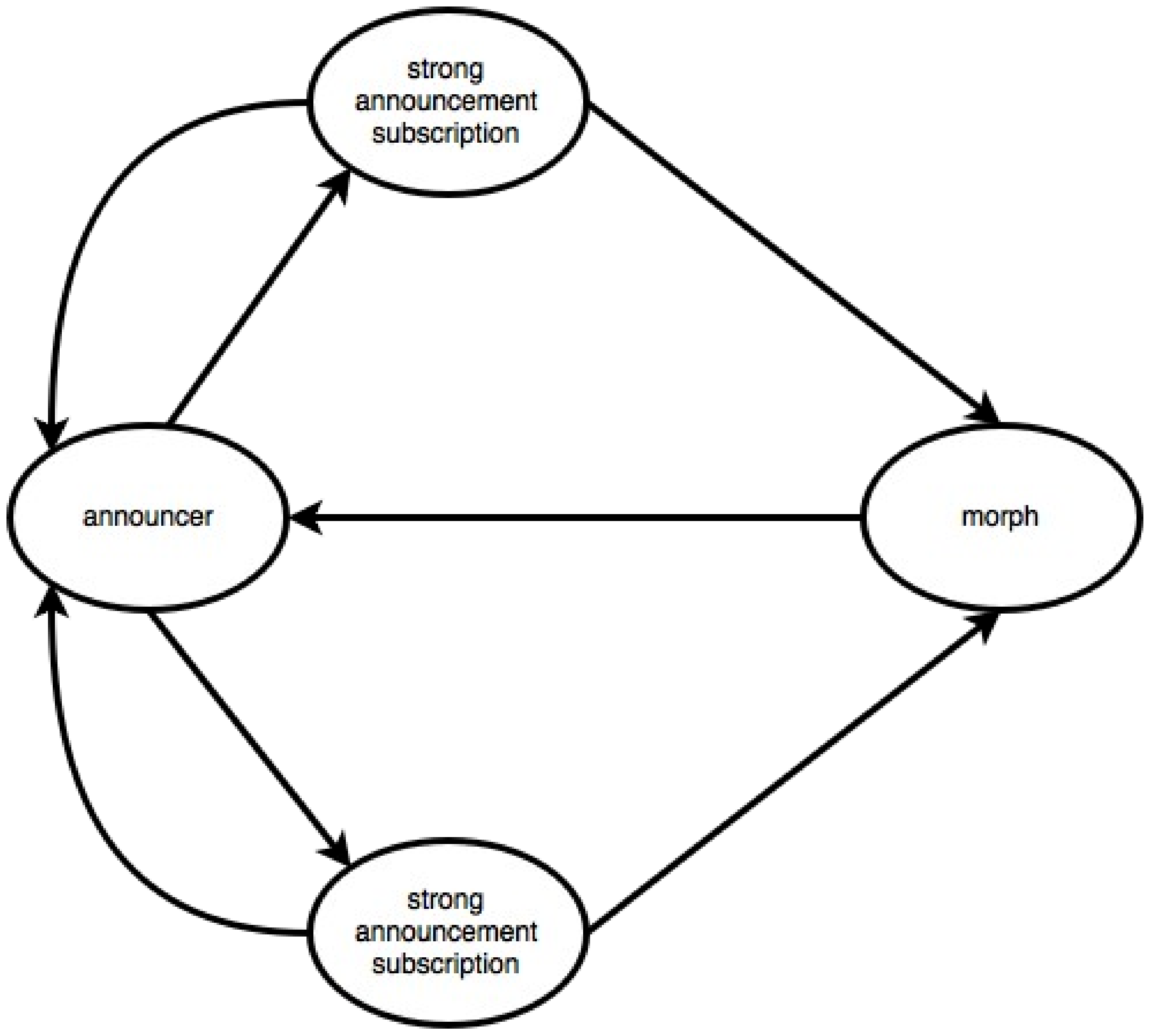
```
announcer weak
        when: anEvent
        send: #message
        to: myObject
```

The Weak Pharo Story (III)

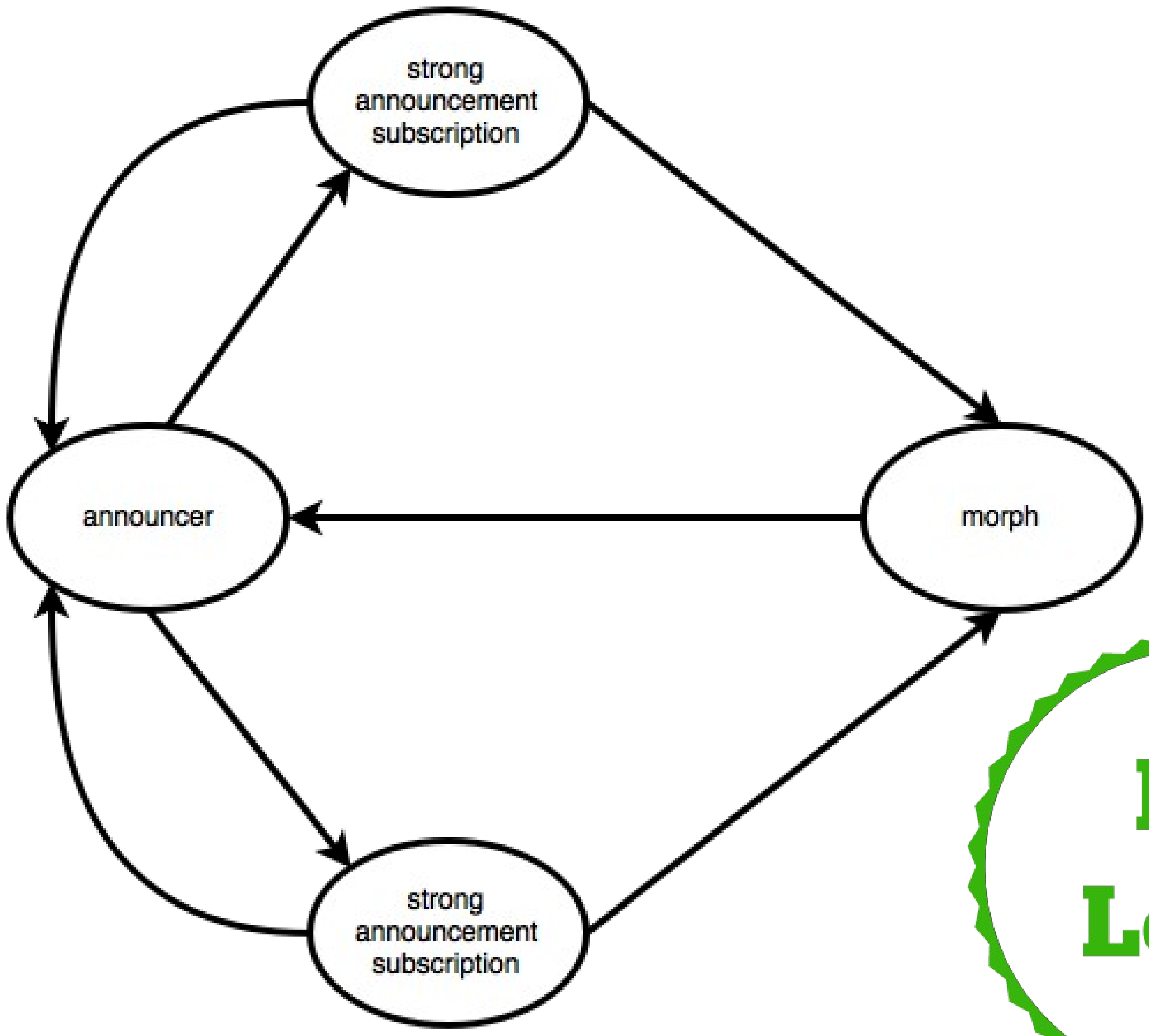
However, *Engineer* did not know this may *curse* **myObject** to be *alive for the eternity*. And never be collected and see his friends die. And create **OutOfMemory** errors on the land of objects to torment the rest of the objects.

The end

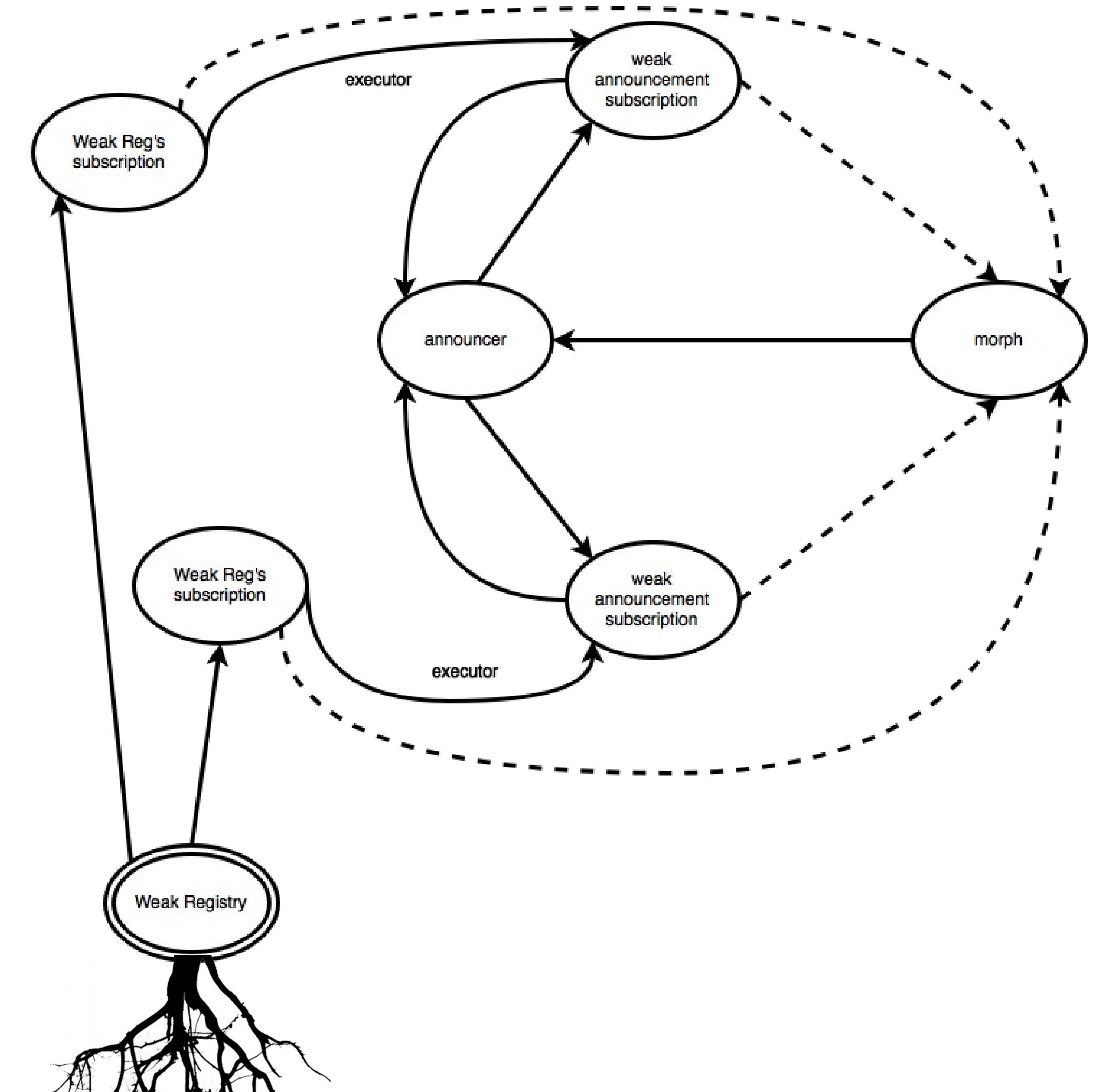
Case 1: The Strong Announcer


strong announcement subscription
announcer
morph
strong announcement subscription

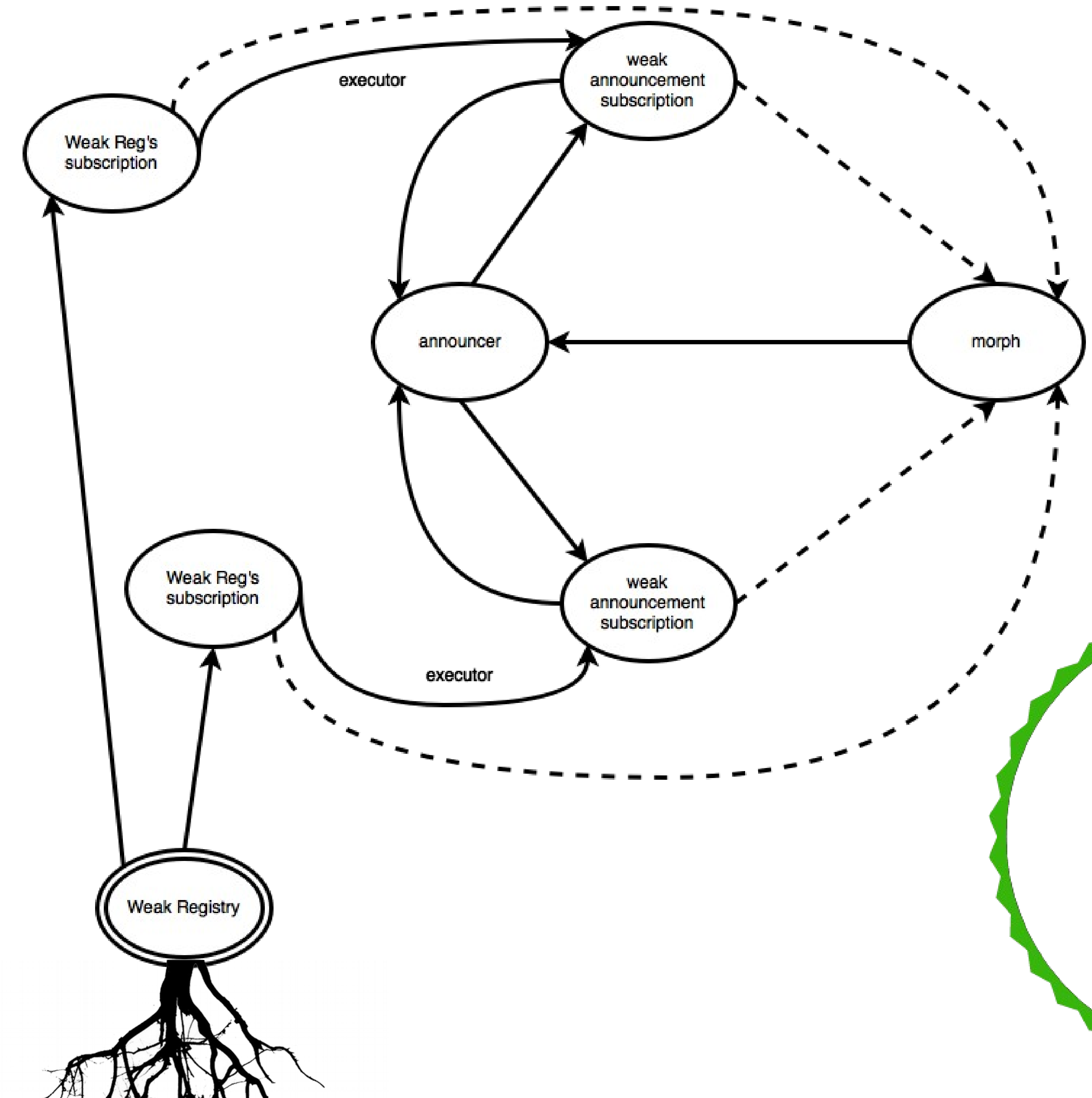
Case 1: The Strong Announcer


strong announcement subscription
announcer
morph
strong announcement subscription

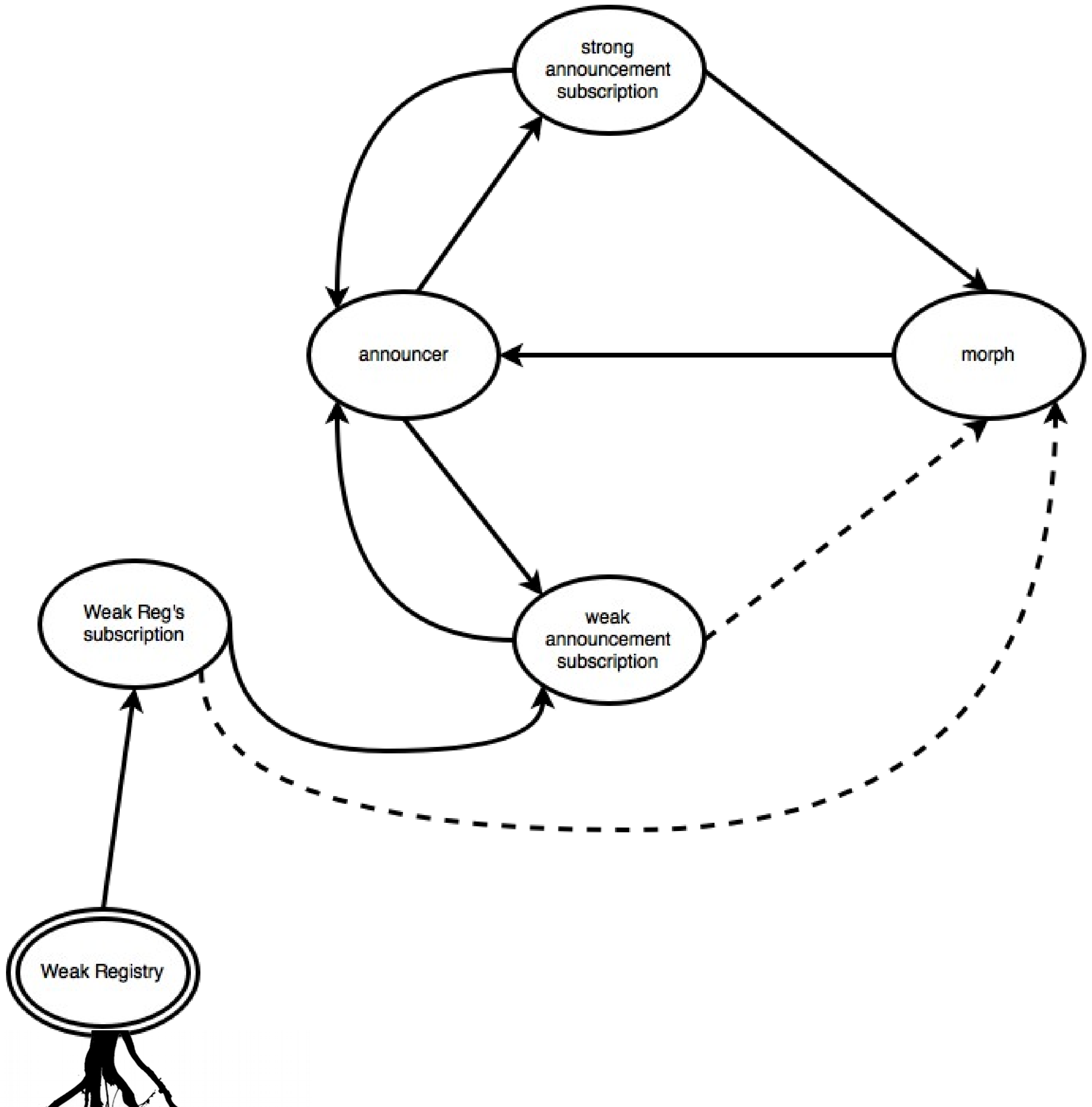
Case 2: The Weak Announcer



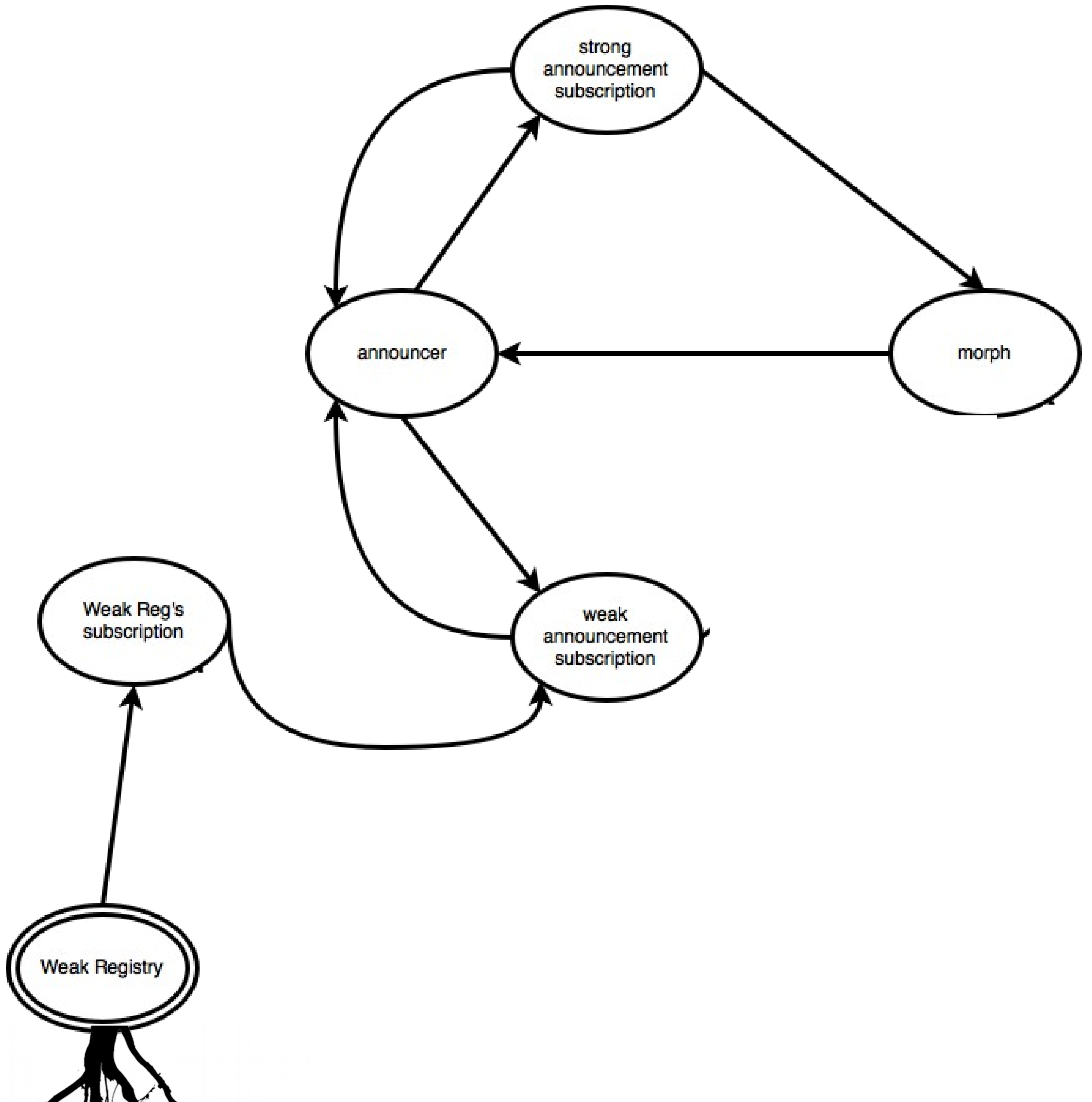
Case 2: The Weak Announcer



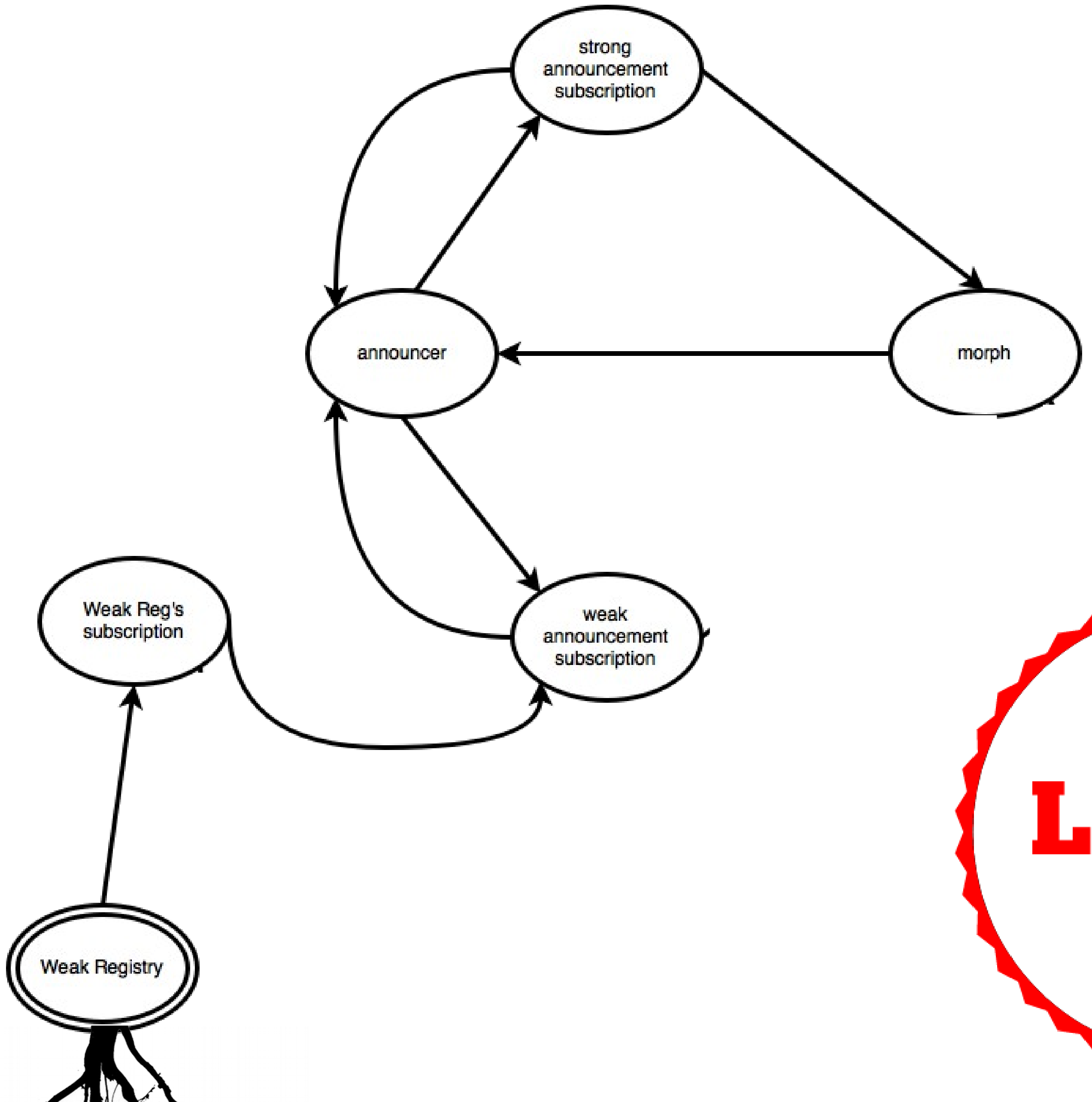
Case 3: The Hybrid Announcer


strong announcement subscription
announcer
morph
Weak Reg's subscription
weak announcement subscription
Weak Registry

Case 3: The Hybrid Announcer


strong announcement subscription
announcer
morph
weak announcement subscription
Weak Reg's subscription
Weak Registry

Case 3: The Hybrid Announcer


strong announcement subscription
announcer
morph
Weak Reg's subscription
weak announcement subscription
Weak Registry

Autopsy

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

So… solutions?

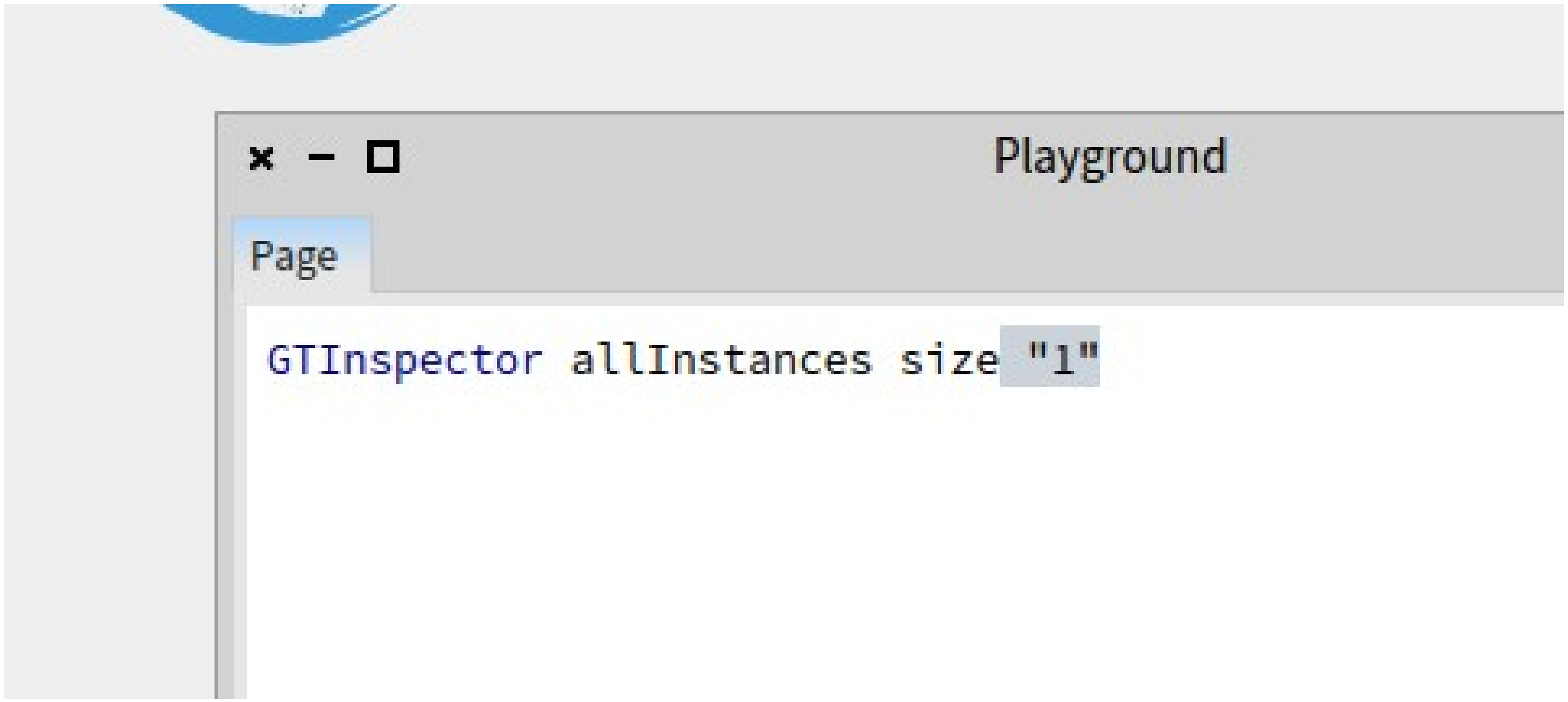
1) How do we detect leaks?

2) How do we prevent some?

#1 - Detecting Leaks

Gotta catch 'em all!™

Memory leaks investigation



```
GTInspector allInstances size "1"
```

Why ???

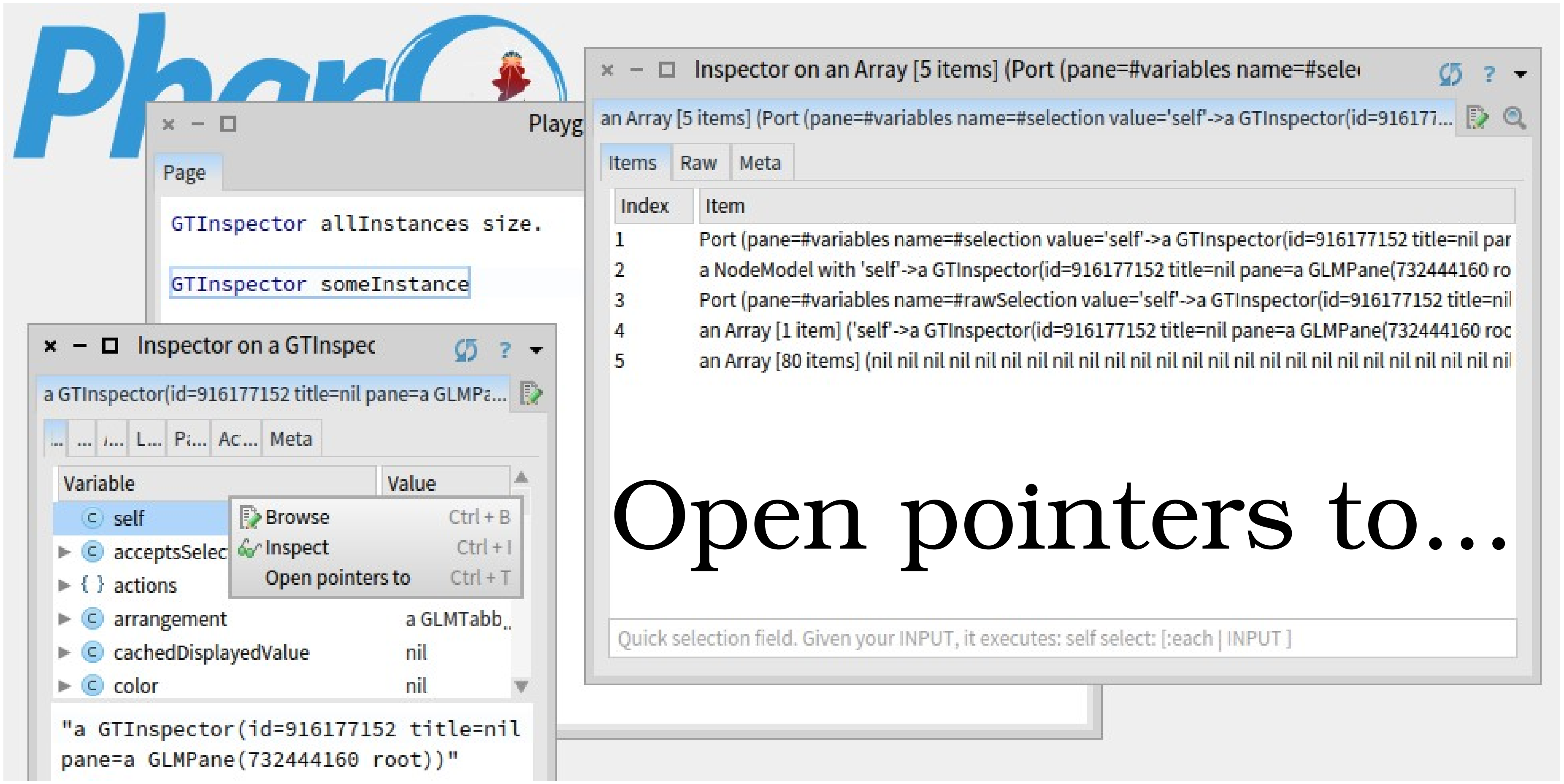
anObject pointersTo

- Very inefficient

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

Easy to get lost

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


Playg
Page
Inspector on a GTInspec
Browse Ctrl + B
Inspect Ctrl + I
Open pointers to Ctrl + T
Items
Raw
Meta
Quick selection field. Given your INPUT, it executes: self select: [:each | INPUT]

```
GTInspector allInstances size.

GTInspector someInstance
```

Inspector on an Array [5 items] (Port (pane=#variables name=#sele

an Array [5 items] (Port (pane=#variables name=#selection value='self'->a GTInspector(id=916177...

Index	Item
1	Port (pane=#variables name=#selection value='self'->a GTInspector(id=916177152 title=nil par
2	a NodeModel with 'self'->a GTInspector(id=916177152 title=nil pane=a GLMPane(732444160 ro
3	Port (pane=#variables name=#rawSelection value='self'->a GTInspector(id=916177152 title=nil
4	an Array [1 item] ('self'->a GTInspector(id=916177152 title=nil pane=a GLMPane(732444160 roc
5	an Array [80 items] (nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil nil

a GTInspector(id=916177152 title=nil pane=a GLMPa...

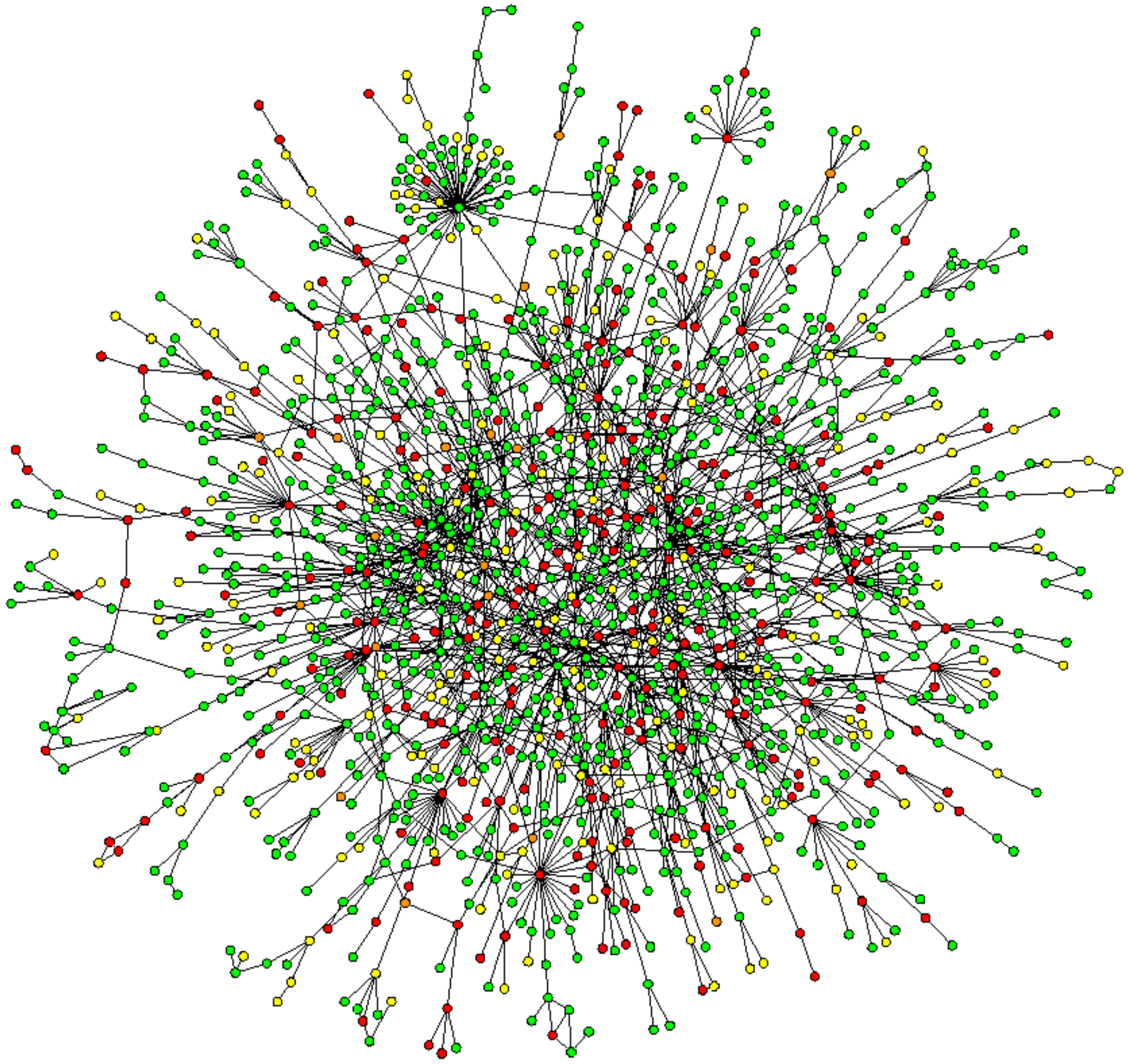
Variable	Value
self	
acceptsSelec	
actions	
arrangement	a GLMTabb
cachedDisplayedValue	nil
color	nil

```
"a GTInspector(id=916177152 title=nil
pane=a GLMPane(732444160 root))"
```

Open pointers to…

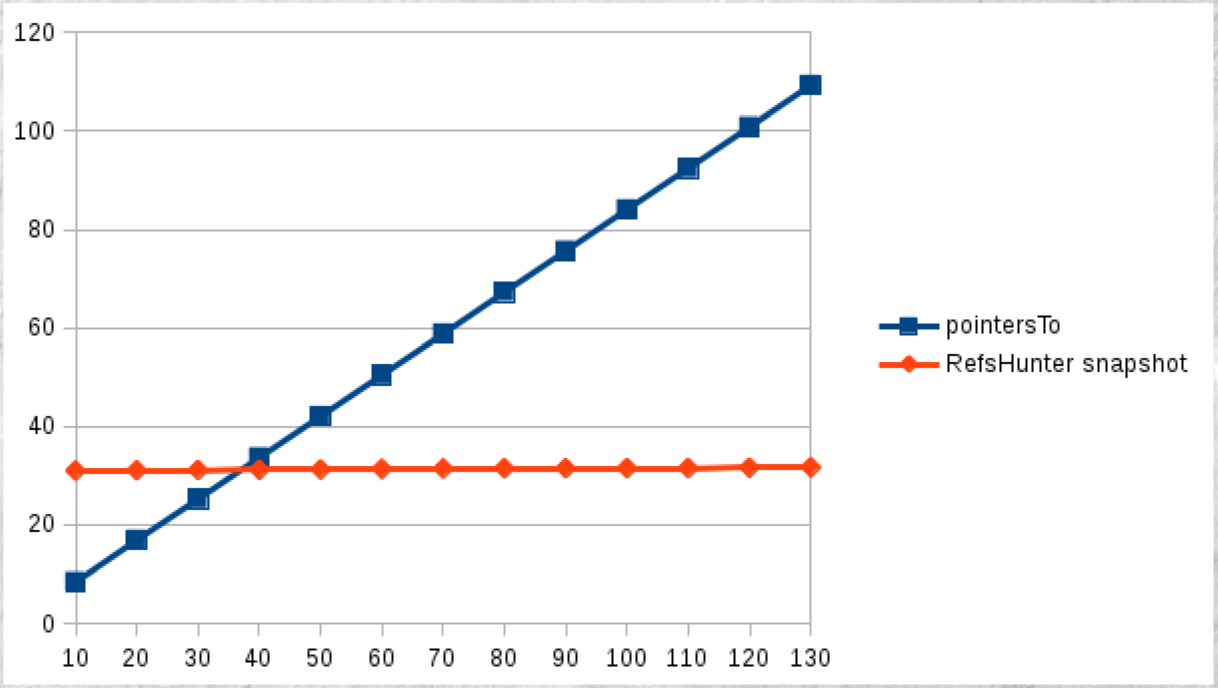
- Easy to use for simple cases
- Uses #pointersTo
- References from tools ▶ more mess

Hell of announcements and weak references



RefsHunter

- Temporary snapshot of the object memory



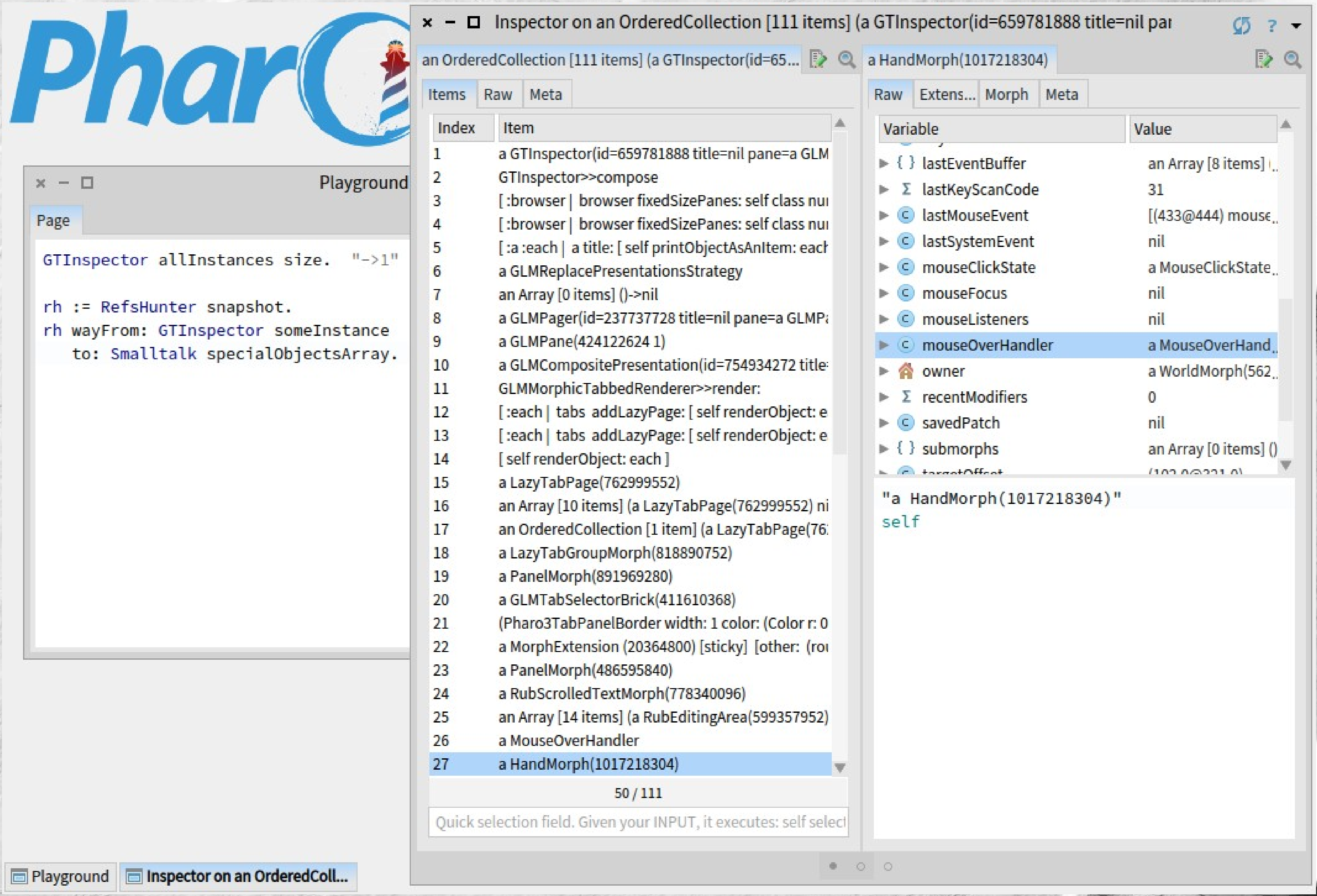
RefsHunter

- 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 inefficent
  - more snapshots are not a good idea, really
- Download from the Catalog


Pharo
Playground
Page
Inspector on an OrderedCollection [111 items] (a GTInspector(id=659781888 title=nil pa
an OrderedCollection [111 items] (a GTInspector(id=65...
Items
Raw
Meta
a HandMorph(1017218304)
Raw
Extens...
Morph
Meta
Playground
Inspector on an OrderedColl...

```
GTInspector allInstances size.  "->1"

rh := RefsHunter snapshot.
rh wayFrom: GTInspector someInstance
   to: Smalltalk specialObjectsArray.
```

Index	Item
1	a GTInspector(id=659781888 title=nil pane=a GLM
2	GTInspector>>compose
3	[:browser | browser fixedSizePanes: self class nu
4	[:browser | browser fixedSizePanes: 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

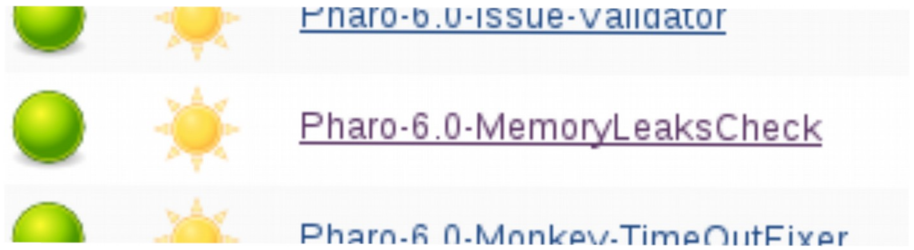
Quick selection field. Given your INPUT, it executes: self select

Variable	Value
lastEventBuffer	an Array [8 items]
lastKeyScanCode	31
lastMouseEvent	[(433@444) mouse
lastSystemEvent	nil
mouseClickState	a MouseClickState
mouseFocus	nil
mouseListeners	nil
mouseOverHandler	a MouseOverHand
owner	a WorldMorph(562
recentModifiers	0
savedPatch	nil
submorphs	an Array [0 items] ()

```
"a HandMorph(1017218304)"
self
```

Avoid memory leaks

- Memory leak tests
  - Time consuming for basic Pharo image



Pharo-6.0-Issue-Validator

Pharo-6.0-MemoryLeaksCheck

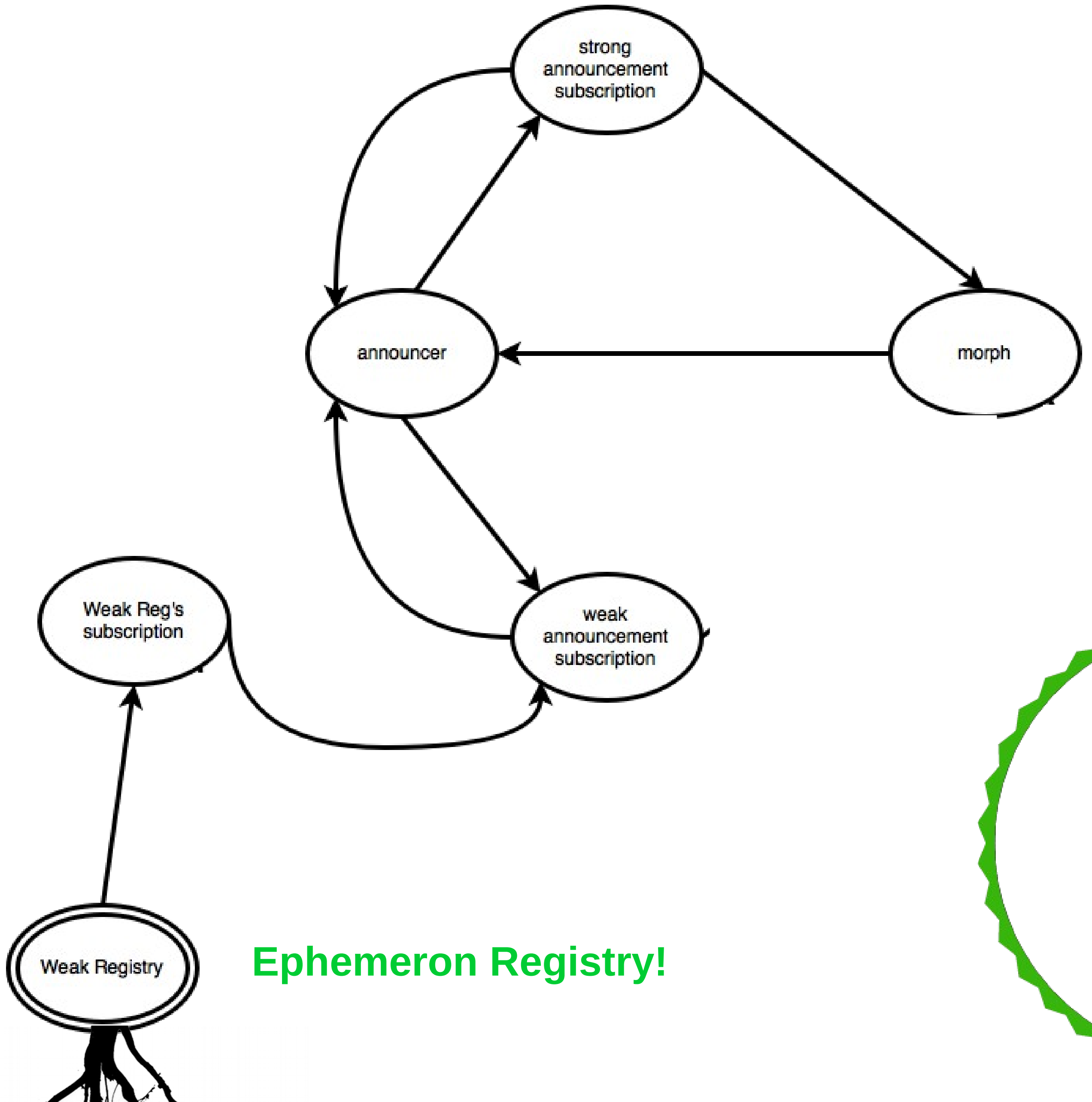
Pharo-6.0-Monkey-TimeOutFixer

#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

Ephemerons in Case 3


strong announcement subscription
announcer
morph
Weak Reg's subscription
weak announcement subscription
Weak Registry

Ephemeron!

Ephemeron Registry!

No Leak

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 lasts forever
  - it takes 30 seconds to garbage collect closed Nautilus
- We need better tools

The End

- 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...

Ephemerons – Operational View

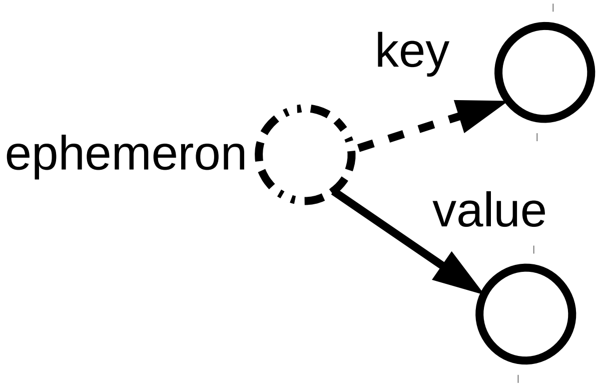
- When the GC passes...
- 1) It does **not** traverse ephemerons:

  It queues them

- 2) Then

```
[ traverses ephemerons whose key is referenced ]
    whileTrue: [
        there are ephemerons with keys referenced ]
```

Exercises:


key
ephemeron
value