The Unix Executable as a Smalltalk Method

Draft talk for Onward! Essays at SPLASH’25, Oct 16-18

Joel Jakubovic

The “Executable File” is secretly a Smalltalk method in disguise.

Programming System = Language + Everything Else

- Language
- Compiler
- Editor
- Libraries
- Debugger
- etc

a.k.a. “programming environment”

Smalltalk System = Smalltalk Lang + Everything Else

Projects Tools Apps Do Extras Windows Help HomeProject Search or evaluate... 00:28:03

ToolBuilder-Morphic: PluggableButtonMorphPlus

ProtoObject
Object
Morph
PluggableButtonMorph
PluggableButtonMorphPlus

-- all --
accessing
action
debug menu
drawing
event handling

action
action:
enabled
enabled:
getColorSel
getColorSel

System Browser: MyWindow

Refactoring-Critics-BlockRules
Refactoring-Critics-ParseTree
Refactoring-Critics-Transform
Refactoring-Spelling
RefactoringTools
ConfigurationOf
RoelTyper-Core
RoelTyper-Pharc
RoelTyper-Squea
RoelTyper-Tests
Ocompletion
Ocompletion-EC
Ocompletion-EC
Ocompletion-EC
OcompletionSqu
Cassowary-Kern
ToolBuilder tests
Cassowary-Tests
Cassowary-Dem

MyWindow

-- all --
as yet unclassified

buildWith:

browse

buildWith: build
window := b
textSpec :=
window child
^ builder bui

Control Pa

Edit Strer
GridX:

Workspace

```
wnd := SystemWindow new. wnd.
wnd setLabel: 'Control Panel'.
t1 := StaticTextMorph new. t1 newContents: 'Edit Strength:'.
wnd addMorph: t1.
wnd openInWorld.
t1 position: (wnd position + (50@50)).
b := PluggableButtonMorphPlus new.
wnd addMorph: b.
b label: 'Preferred'.
b position: (t1 topRight + (10@0)).
b2 := PluggableButtonMorphPlus new.
wnd addMorph: b2.
b2 label: 'Required'.
b2 position: (b topRight + (10@0)).
t2 := StaticTextMorph new. t2 newContents: 'GridX:'.
wnd addMorph: t2.
t2 position: (t1 bottomLeft + (0@10)).
wnd extent: 380@150.

g := GridLayoutTest new.
g setUp.
g test05AdhereToEdge

ToolBuilder build: MyWindow new.
myLocal
Context
```

Kernel-Methods: Context

ProtoObject
Object
InstructionStream
Context
ContextPart
BlockContext

instance class ?

-- all --
*60Deprecated-controlling
*Tools-Inspector
*Tools-debugger access
accessing
closure support
controlling

aboutToReturn:through:
activateMethod:withArgs:receiv
activateMethod:withArgs:receiv
activateReturn:value:
activeHome
activeOuterContext
actualStackSize

browse senders implementors versions inheritance hierarchy variables source

Process inspector. Stepping in the debugger is done by sending messages to contexts to get them to execute their bytecodes. See methods in the instruction decoding protocol.

Contexts, though normal in their variable size, are actually only used in two sizes, small and large, which are determined by the temporary space required by the method being executed.

Contexts must only be created using the method newForMethod:. Note that it is impossible to determine the real object size of a Context except by asking for the frameSize of its method. Any fields above the stack pointer (stackp) are truly invisible -- even (and especially!) to the garbage collector. Any store into stackp other than by the primitive method stackp: is potentially fatal.

(*) efficient virtual machines create contexts lazily on demand, avoiding the overhead of creating them on every message send and of copying receiver and arguments from sender context to caller context. This optimization is invisible to the Smalltalk system.

class comment for Context - eem 4/4/2017 17:45

Smalltalk System = Smalltalk Lang + Everything Else

Projects Tools Apps Do Extras Windows Help HomeProject Search or evaluate... 00:28:03

ToolBuilder-Morphic: PluggableButtonMorphPlus

ProtoObject
Object
Morph
PluggableButtonMorph
PluggableButtonMorphPlus

-- all --
accessing
action
debug menu
drawing
event handling

action
action:
enabled
enabled:

System Browser: MyWindow

Refactoring-Critics-BlockRules
Refactoring-Critics-ParseTree
Refactoring-Critics-Transform
Refactoring-Spelling
RefactoringTools
ConfigurationOf
RoelTyper-Core
RoelTyper-Pharo
RoelTyper-Squea
RoelTyper-Tests
Ocompletion
Ocompletion-EC
Ocompletion-EC
Ocompletion-EC
OcompletionSqu
Cassowary-Kern
ToolBuilder tests
Cassowary-Tests
Cassowary-Dem

MyWindow

-- all --
as yet unclassified

buildWith:

Kernel-Methods: Context

ProtoObject
Object
InstructionStream
Context
ContextPart
BlockContext

-- all --
*60Deprecated-controlling
*Tools-Inspector
*Tools-debugger access
accessing
closure support
controlling

aboutToReturn:through:
activateMethod:withArgs:receiv
activateMethod:withArgs:receiv
activateReturn:value:
activeHome
activeOuterContext
actualStackSize

instance class ?

browse senders implementors versions inheritance hierarchy variables source

Process inspector. Stepping in the debugger is done by sending messages to contexts to get them to execute their bytecodes. See methods in the instruction decoding protocol.

Contexts, though normal in their variable size, are actually only used in two sizes, small and large, which are determined by the temporary space required by the method being executed.

Contexts must only be created using the method newForMethod:. Note that it is impossible to determine the real object size of a Context except by asking for the frameSize of its method. Any fields above the stack pointer (stackp) are truly invisible -- even (and especially!) to the garbage collector. Any store into stackp other than by the primitive method stackp: is potentially fatal.

(*) efficient virtual machines create contexts lazily on demand, avoiding the overhead of creating them on every message send and of copying receiver and arguments from sender context to caller context. This optimization is invisible to the Smalltalk system.

class comment for Context - eem 4/4/2017 17:45

Workspace

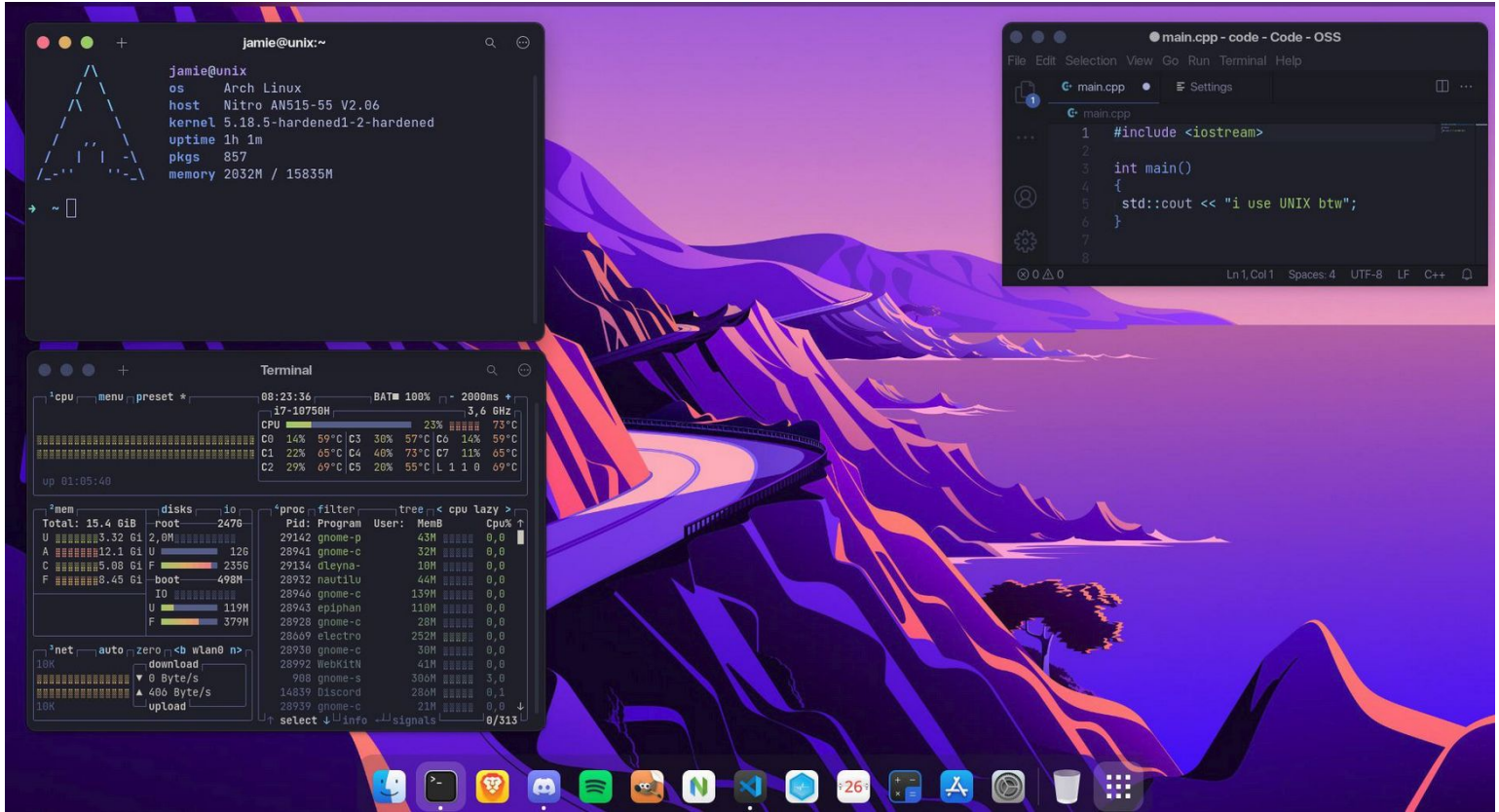
```
vnd := SystemWindow new. wnd.
vnd setLabel: 'Control Panel'.
1 := StaticTextMorph new. t1 newContents: 'Edit Strength:'.
vnd addMorph: t1.
vnd openInWorld.
1 position: (wnd position + (50@50)).
 := PluggableButtonMorphPlus new.
vnd addMorph: b.
 label: 'Preferred'.
 position: (t1 topRight + (10@0)).
2 := PluggableButtonMorphPlus new.
vnd addMorph: b2.
2 label: 'Required'.
2 position: (b topRight + (10@0)).
2 := StaticTextMorph new. t2 newContents: 'GridX:'.
vnd addMorph: t2.
2 position: (t1 bottomLeft + (0@10)).
vnd extent: 380@150.

 := GridLayoutTest new.
 setUp.
 test05AdhereToEdge

oolBuilder build: MyWindow new.
nyLocal
ontext
```

Language

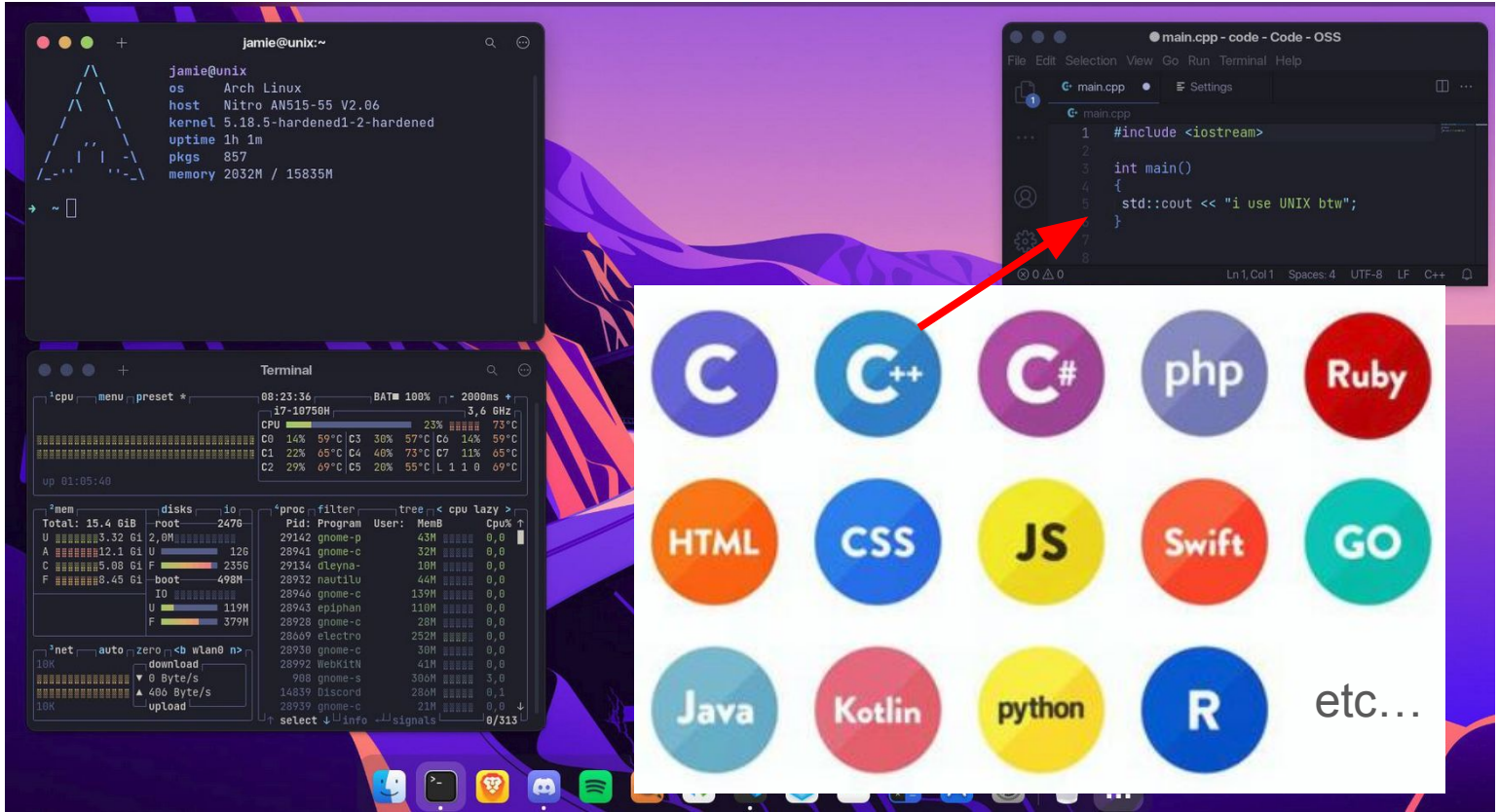
Control Pa
Edit Stren
GridX:

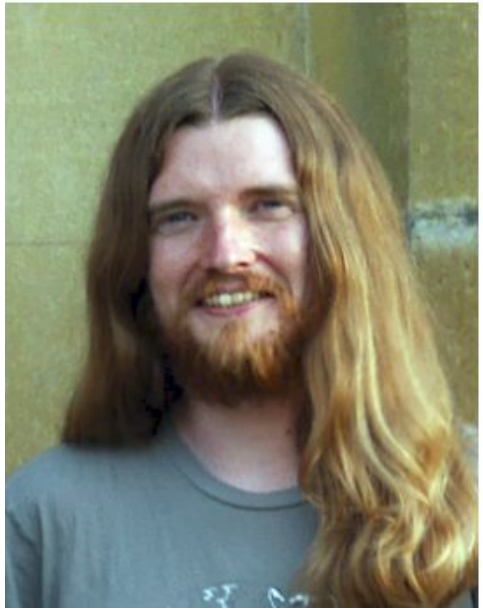
Everything else

Unix System = Many Programming Systems



Unix System = Many Programming Systems





Unix, Plan 9 and the Lurking Smalltalk

Stephen Kell

Kell, S. (2018). Unix, Plan 9 and the Lurking Smalltalk. In: De Mol, L., Primiero, G. (eds) Reflections on Programming Systems. Philosophical Studies Series, vol 133. Springer, Cham.
https://doi.org/10.1007/978-3-319-97226-8_6

Abstract High-level programming languages and their virtual machines have long aspired to erase operating systems from view. Starting from Dan Ingalls' Smalltalk-inspired position that "an operating system is a collection of things that don't fit inside a language; there shouldn't be one", I contrast the ambitions and trajectories of Smalltalk with those of Unix and its descendents, exploring why Ingalls's vision appears not (yet) to have materialised. Firstly, I trace the trajectory of Unix's "file" abstraction into Plan 9 and beyond, noting how its logical extrapolation suggests a

Smalltalk: Visionary, Influential, Designed … yet Niche

Kell: “Smalltalk, by contrast [to Unix], is easier to miss in modern systems. As a language, today it finds only niche interest. Its key programmatic concepts, namely classes and late-bound “messaging”, have had an enormous influence on popular languages; this is clearest in highly dynamic class-based languages such as Python and Ruby, but is easily discernible in Java and C++, among many others. The rich user interface it presented to the programmer has also influenced countless modern “integrated” development environments. Despite this considerable influence, something seems to have been lost: anecdotally, enthusiasts are quick to point out that none of these contemporary languages or environments matches the simplicity, uniformity or immediacy of a Smalltalk system.”

Unix: Evolved, Viral, Convenient … but Lacking

Kell: “Unix is, infamously, a survivor—even satirised as the world’s first computer virus. Its design remains ubiquitous: not only in its direct-descendent commodity operating systems (e.g. GNU/Linux), but as a key component of others (Apple’s Mac OS) and a clear influence on the remainder.”

This suggests a **“Generalised Unix”** (Linux, Mac, Windows)**:**
Software tends to be organised as a collection of **large-ish files and processes**, each of which contains significant **sub-structure**, which can be **encoded differently** in each case

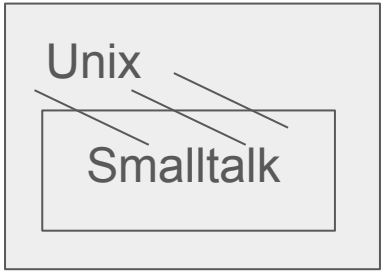
(Kell’s “fragmentation” of file + runtime data formats)

Kell: Help Unix Complete its Evolution!

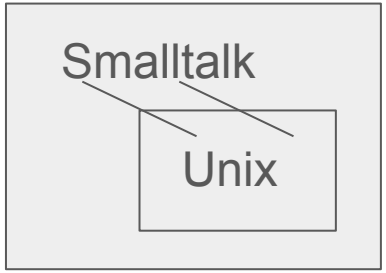
- Highly recommended reading. Summary:
- Kell's "Smalltalk Wishlist": Programmatic+Descriptive Availability, Interposable Bindings
- They sort-of already exist in Unix …
- … but fragmented, half-baked, falling frustratingly short of Smalltalk.
- Because Unix evolved…
- Which was key to its viral success over Smalltalk.
- Kell wants to further this evolution, following Plan 9 ("sequel" to Unix).
- Crucially: do so maintaining plurality of languages/abstractions/views.
- "Oh but you now have program this particular way": not allowed!
- Sorry Stephen…

Preliminary Points

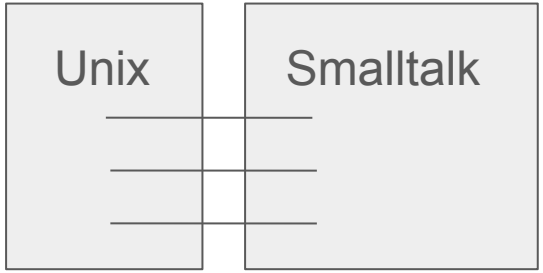
- Ignore superficial differences e.g. names. Searching for similar “shape”, similar structure, similar behaviour.
- Compare Unix and Smalltalk as “equals”. How each looks on its own terms, not how it appears when emulated in the other system.



❌ confusing

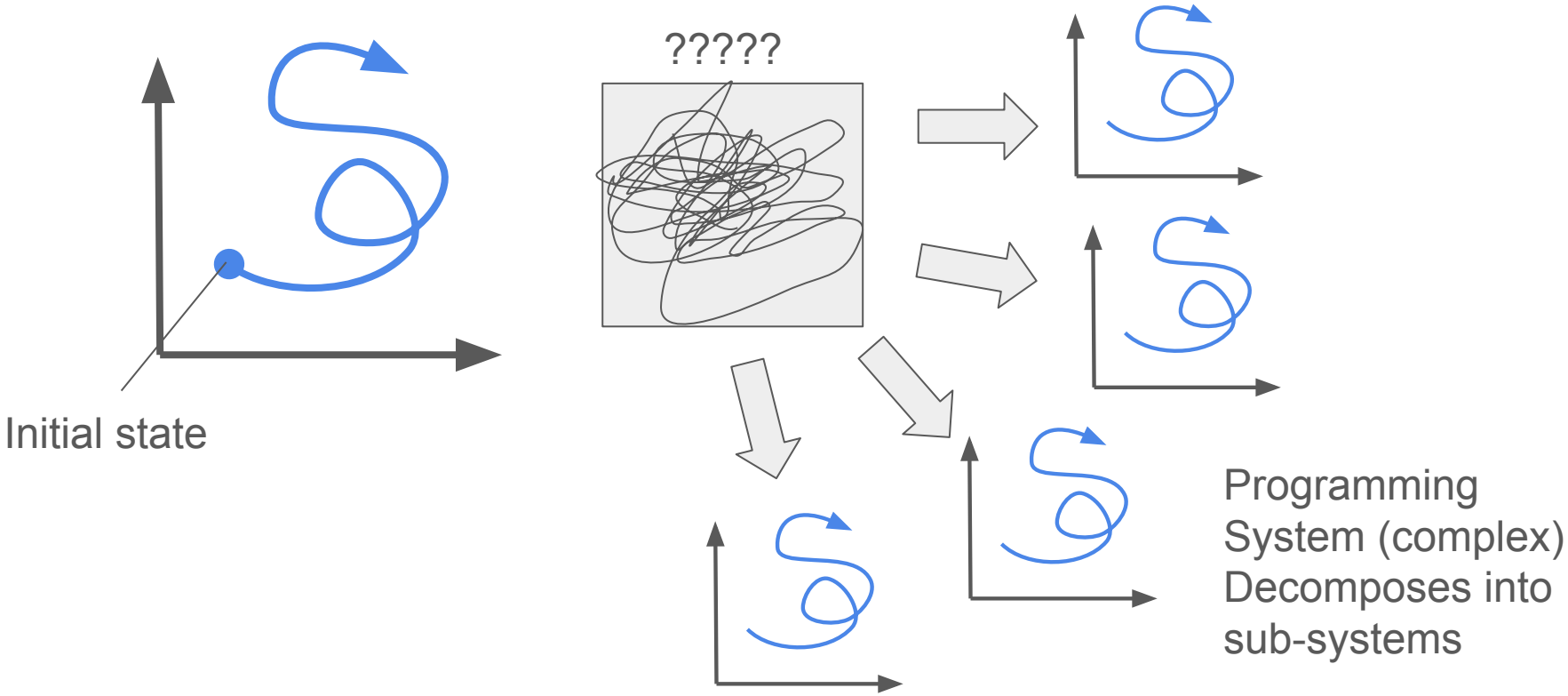


❌ confusing

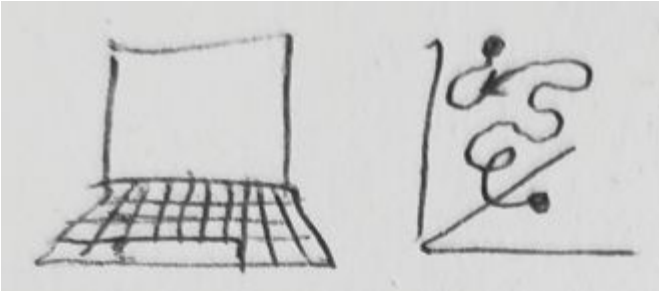


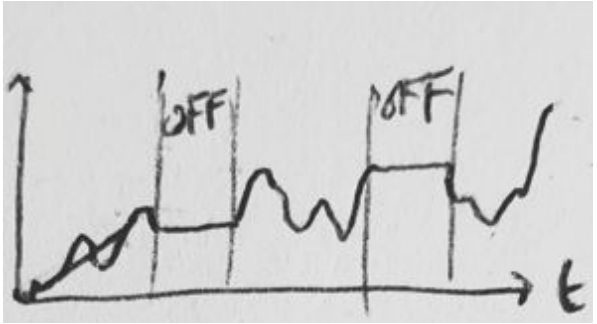
✅ like-for-like

Programming Systems are Dynamical Systems



Interesting State = Persistent State





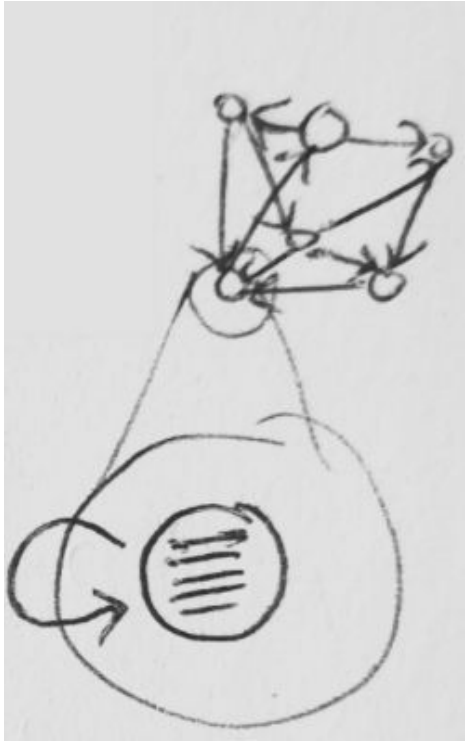
Unix vs. Smalltalk: persistent sub-system evolution

Unix

Unix vs. Smalltalk: persistent sub-system evolution

Unix

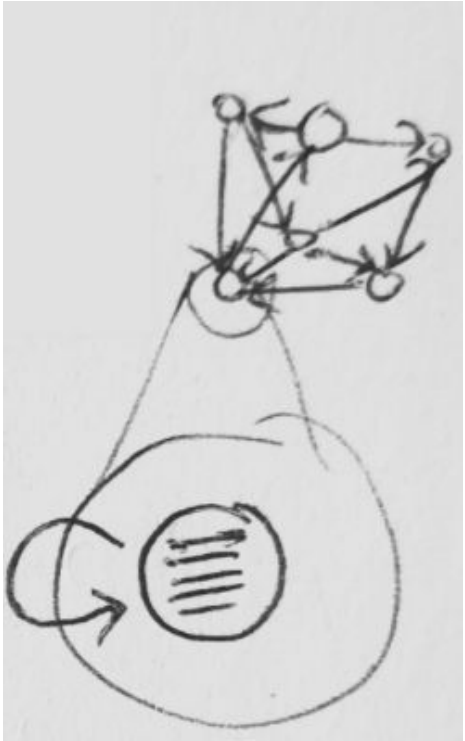
Smalltalk



Unix vs. Smalltalk: persistent sub-system evolution

Unix

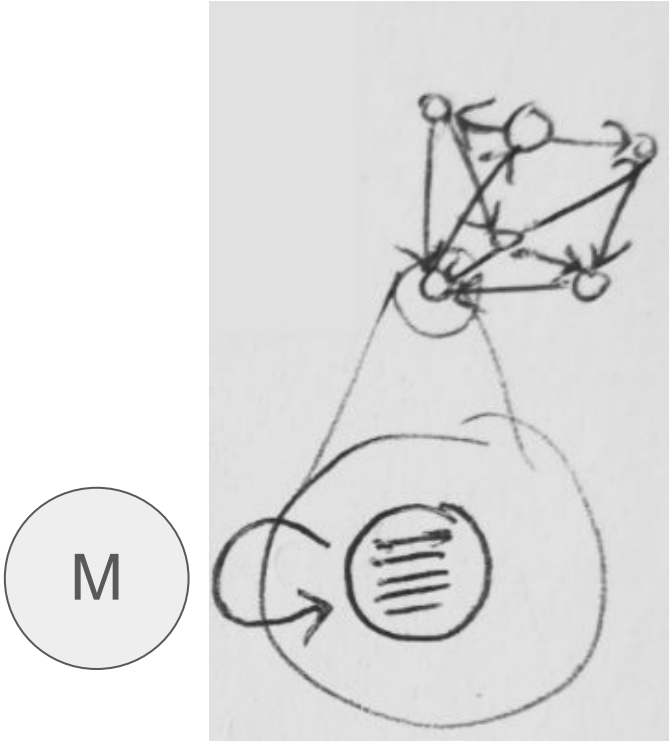
Smalltalk



Unix vs. Smalltalk: persistent sub-system evolution

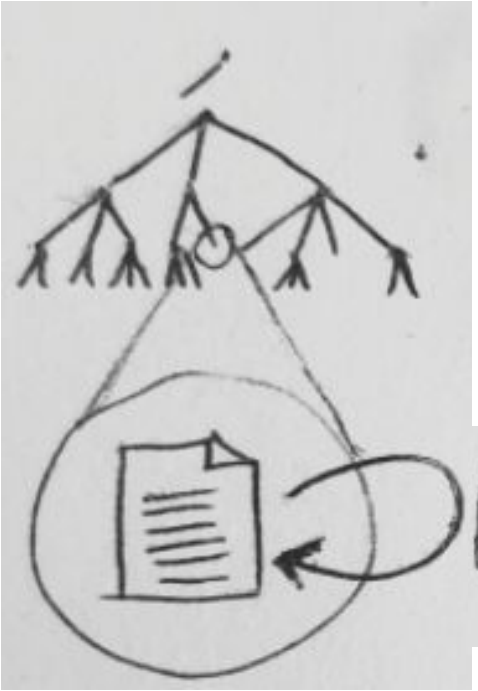
Unix

Smalltalk

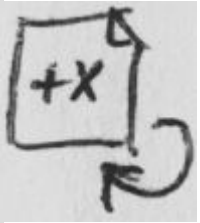


Unix vs. Smalltalk: persistent sub-system evolution

Unix

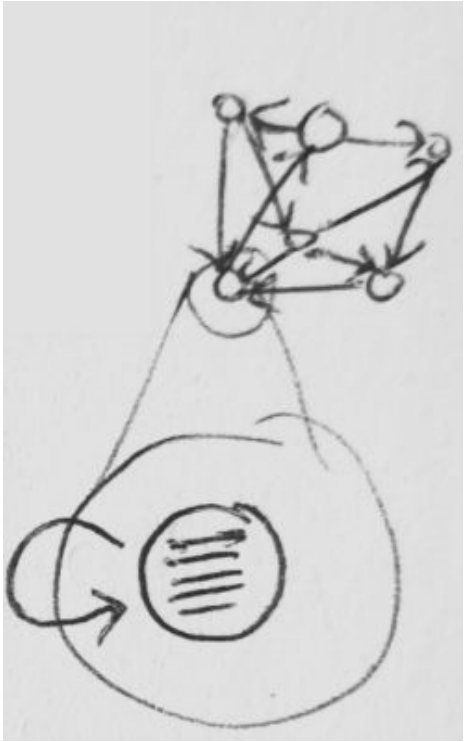


Kell: “It now seems reasonable to declare “file” and “object” as synonymous. Both are equally universal, more-or-less semantics-free, and deliberately so.”



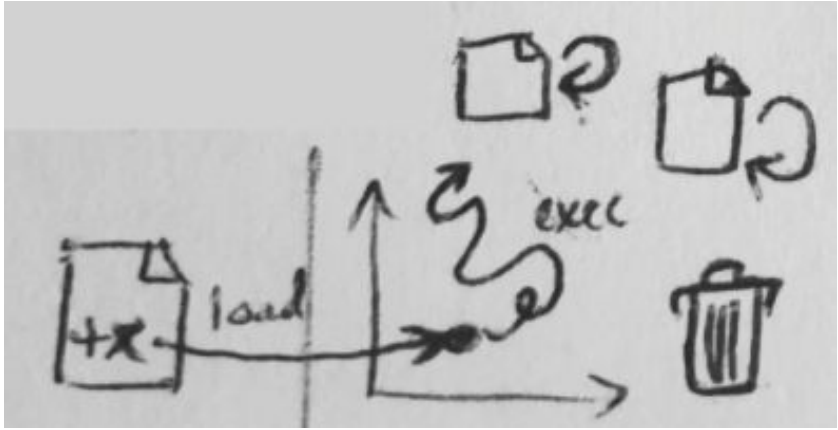
M

Smalltalk



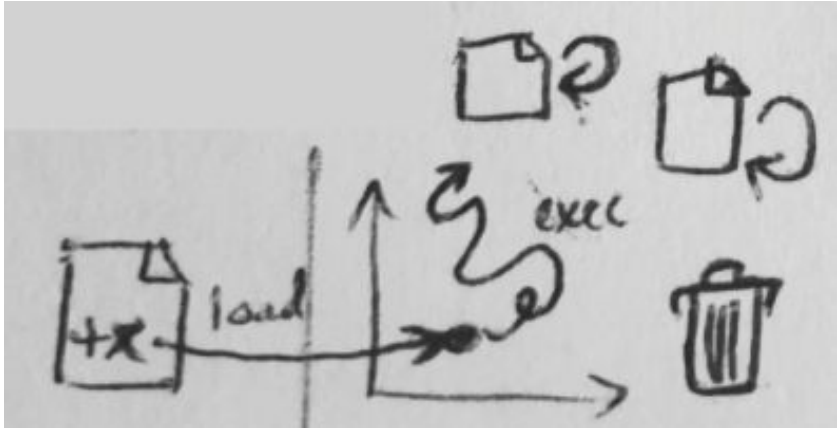
Unix vs. Smalltalk: transient sub-system evolution

Unix



Unix vs. Smalltalk: transient sub-system evolution

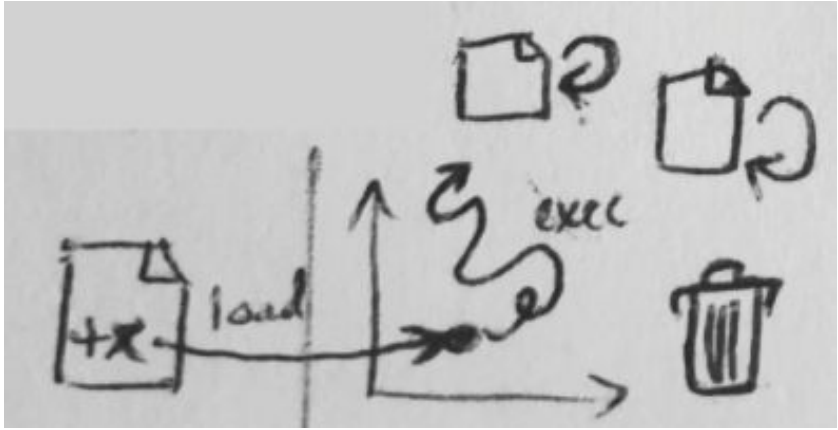
Unix



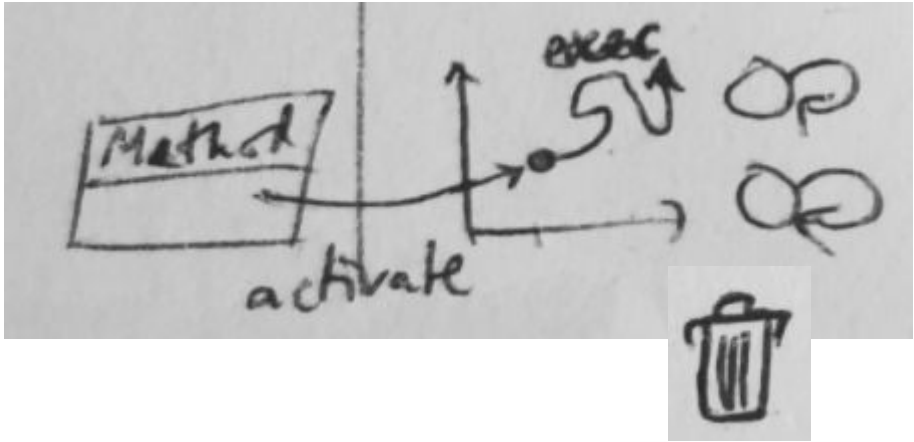
Private address space ✅
(require debug APIs)

Unix vs. Smalltalk: transient sub-system evolution

Unix



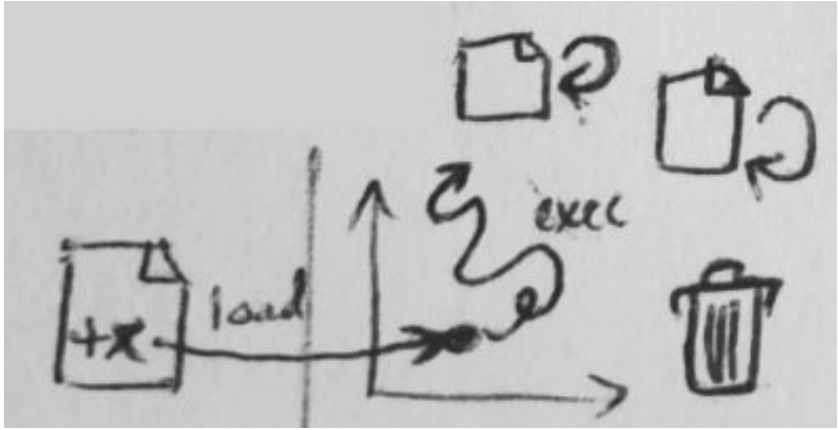
Smalltalk



Private address space ✅
(require debug APIs)

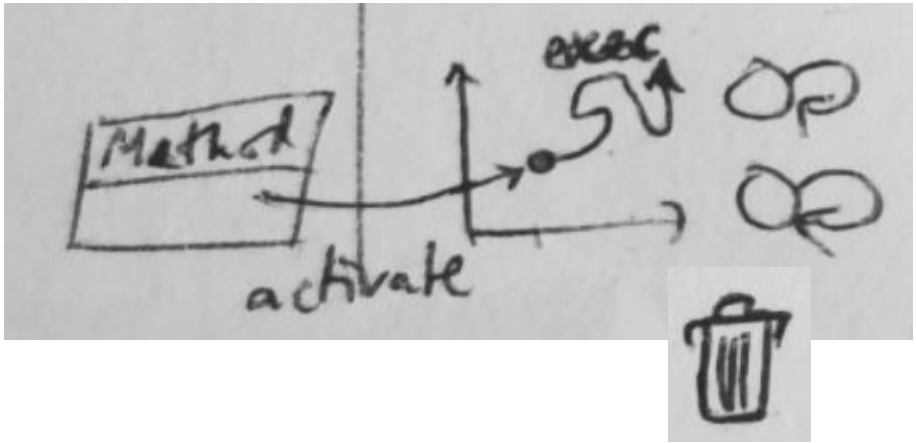
Unix vs. Smalltalk: transient sub-system evolution

Unix



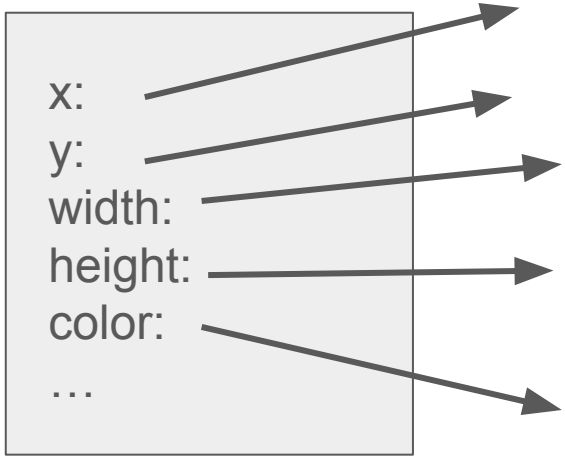
Private address space ✅
(require debug APIs)

Smalltalk



Normally hidden from user✅
(require reflection APIs)

Unix vs. Smalltalk: fragmentation vs. uniformity



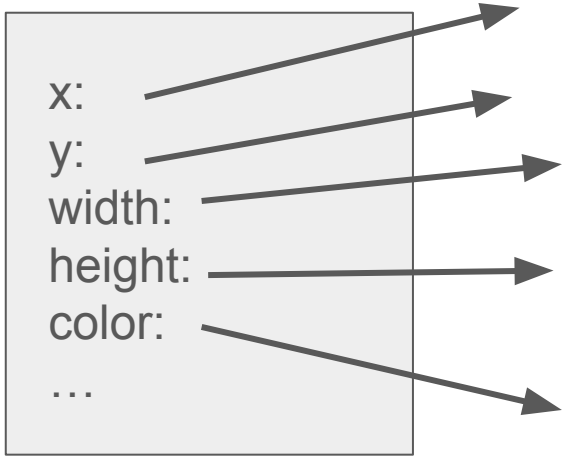
Reflection: enumerate slots

Unix vs. Smalltalk: fragmentation vs. uniformity

Unix file

```
0x0000: 4D
0x0001: 5A
0x0002: 00
0x0003: 00
0x0004: 32
0x0005: 7C
…
```

Smalltalk object



Reflection: enumerate slots

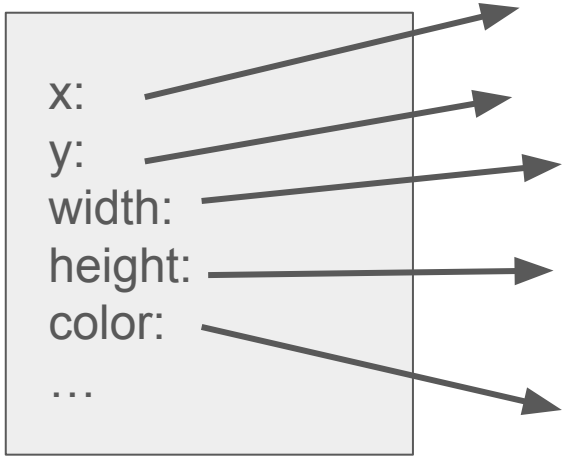
Unix vs. Smalltalk: fragmentation vs. uniformity

Unix file

```
0x0000: 4D
0x0001: 5A
0x0002: 00
0x0003: 00
0x0004: 32
0x0005: 7C
…
```

```
x=10
y=20

{“x”: 10, “y”: 20}

<entry name=”x”
value=”10” />
…
```

Smalltalk object



Reflection: enumerate slots

Unix vs. Smalltalk: fragmentation vs. uniformity

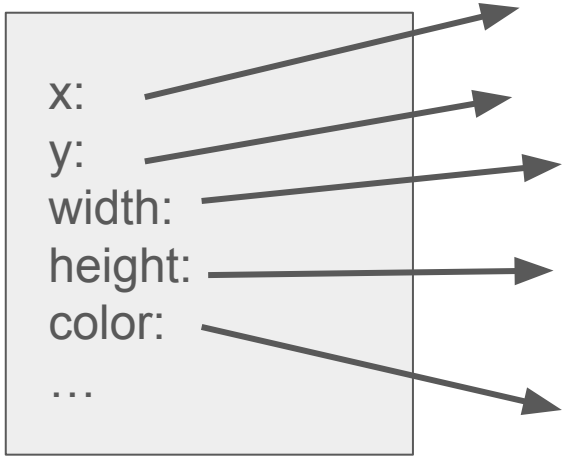
Unix file

```
0x0000: 4D
0x0001: 5A
0x0002: 00
0x0003: 00
0x0004: 32
0x0005: 7C
…
```

```
x=10
y=20

{“x”: 10, “y”: 20}

<entry name=”x”
value=”10” />
…
```

Now add in all the binary encodings…!

Smalltalk object

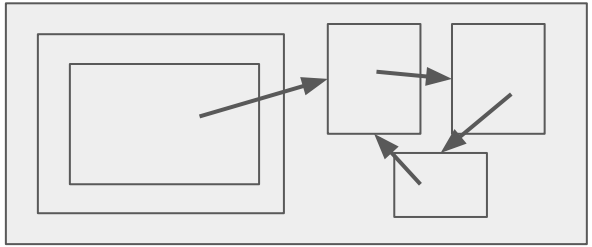


Reflection: enumerate slots

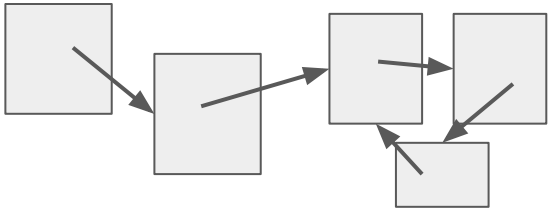
Kell's Fragmentation & "Large Objects"

Kell: "Unix processes happily "accommodate" diverse implementations of language-level abstractions, albeit in the weakest possible sense: by being oblivious to them. (…) Each language implementation must adopt its own mechanisms for object binding and identity, i.e. conventions for representing and storing object addresses. (…) Another way of looking at this is that the operating system concerns itself with *large objects* only."

Unix file/process



Smalltalk objects/methods

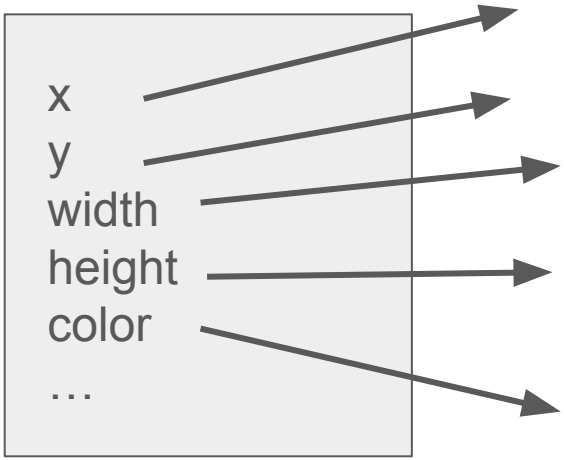


Directories = Objects?

Kell: “This unifying filesystem abstraction includes names and other metadata for all such entities, along with enumerable directory structures. Although primitive, this is clearly a metasystem. For instance, enumeration of files in a directory corresponds closely to enumeration of slots in an object, as expressible using the Smalltalk meta-object protocol.”

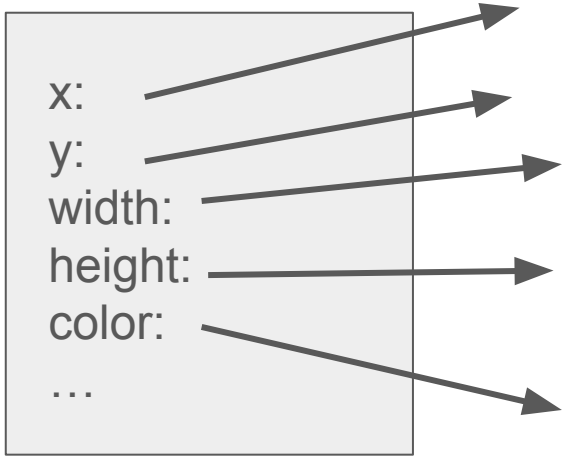
Directories = Objects?

Unix directory



readdir(): enumerate entries
Graph structure ✅ (symlinks)
No fragmentation!

Smalltalk object



Reflection: enumerate slots

ProcFS

```
sagar@LHB:~$ tree /proc
/proc
├── 1
│   ├── arch_status
│   ├── attr
│   │   ├── apparmor
│   │   │   ├── current
│   │   │   ├── exec
│   │   │   └── prev
│   │   ├── current
│   │   ├── exec
│   │   ├── fscreate
│   │   ├── keycreate
│   │   ├── prev
│   │   ├── smack
│   │   │   └── current
│   │   └── sockcreate
│   ├── autogroup
│   ├── auxv
│   ├── cgroup
│   ├── clear_refs
│   ├── cmdline
│   ├── comm
│   ├── coredump_filter
```

Exception that proves the rule…
Most structured data in Unix is not “exploded” into directories and files.

(That’s what Plan 9 does instead.)

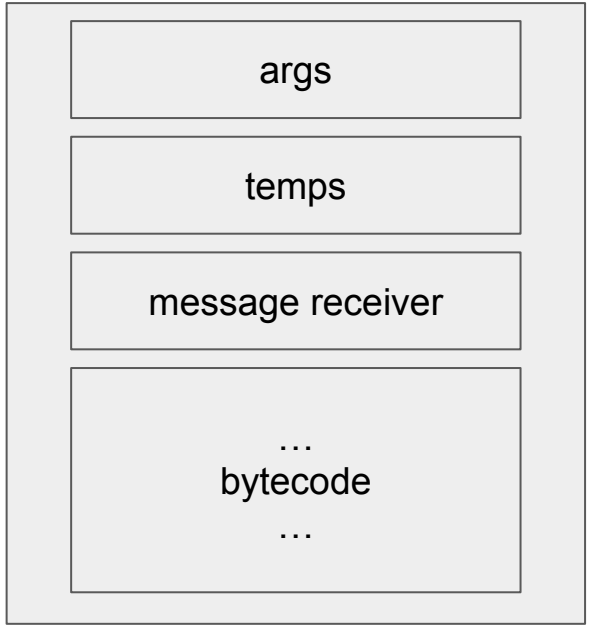
Generalised Unix vs. Generalised Smalltalk

“Generalised Unix”: organising software as a collection of large-ish files and processes with significant + heterogenous internal structure.

“Generalised Smalltalk”: organising software as a graph of millions of small objects and method (activations) with uniform structure.

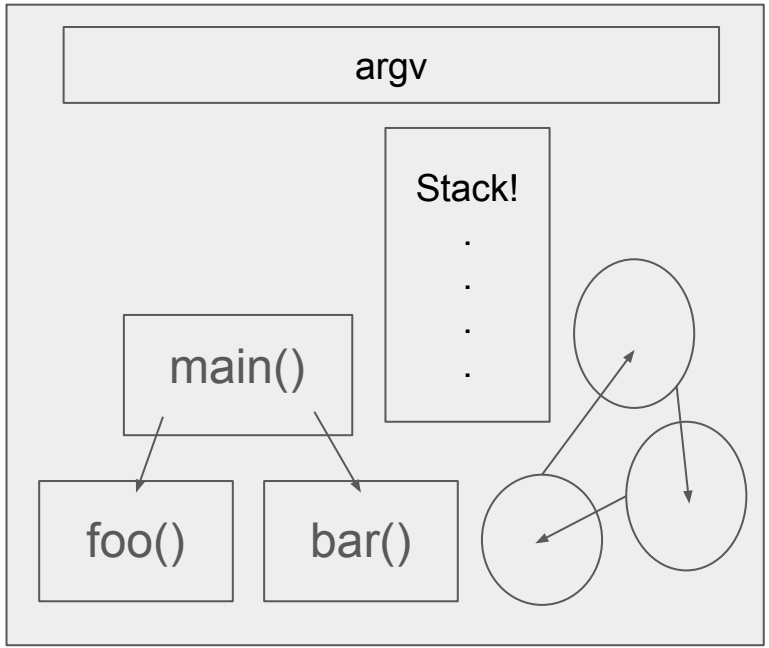
Large Processes, Small Methods

Smalltalk method activation

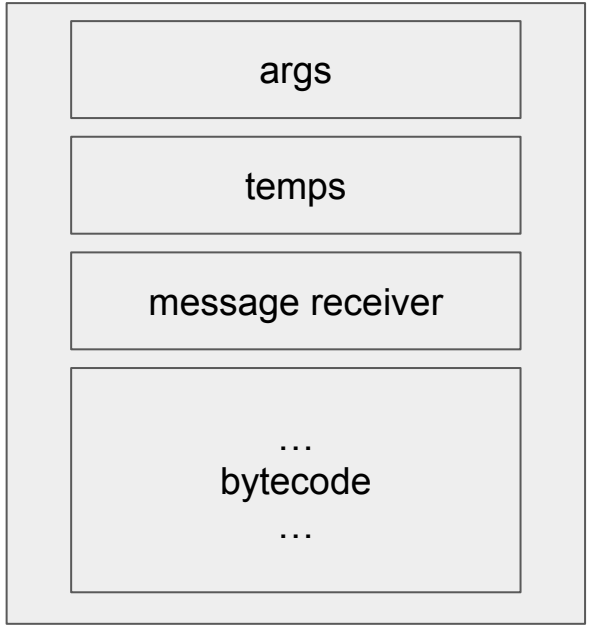


Large Processes, Small Methods

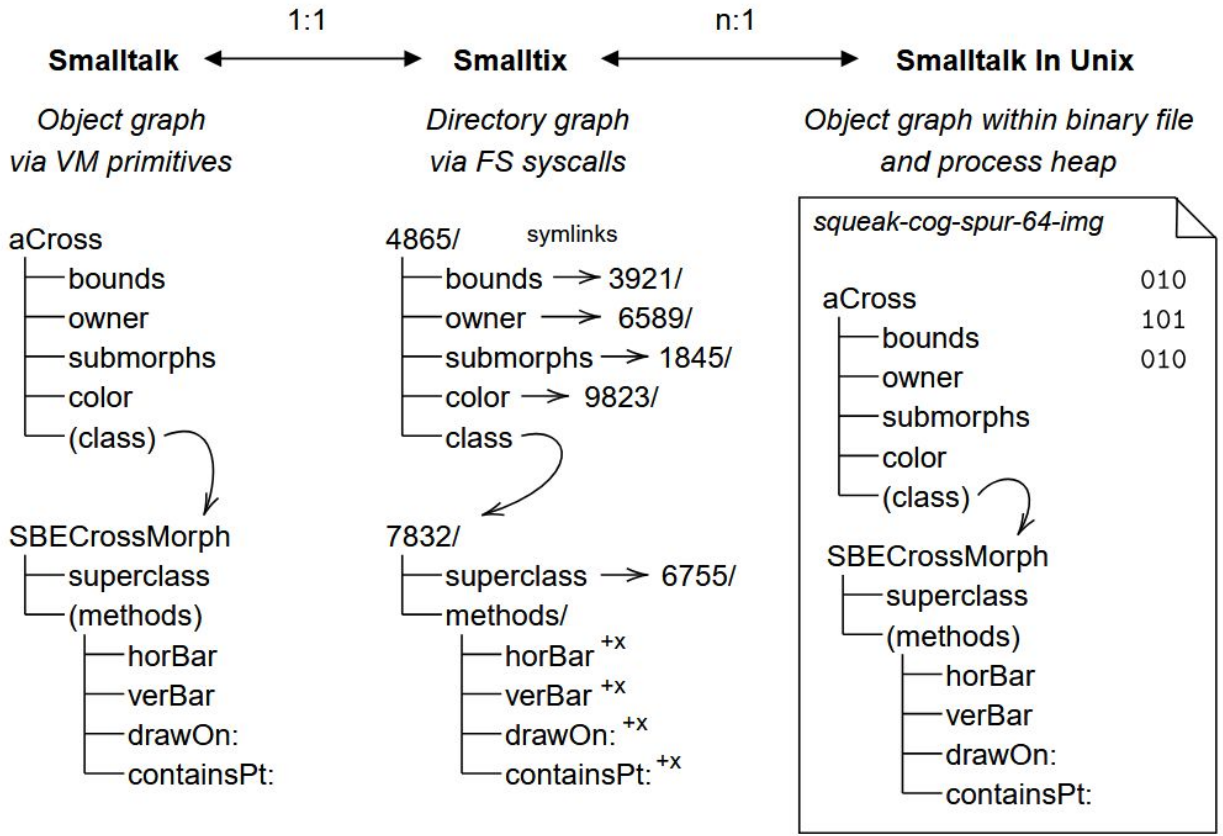
Unix process



Smalltalk method activation

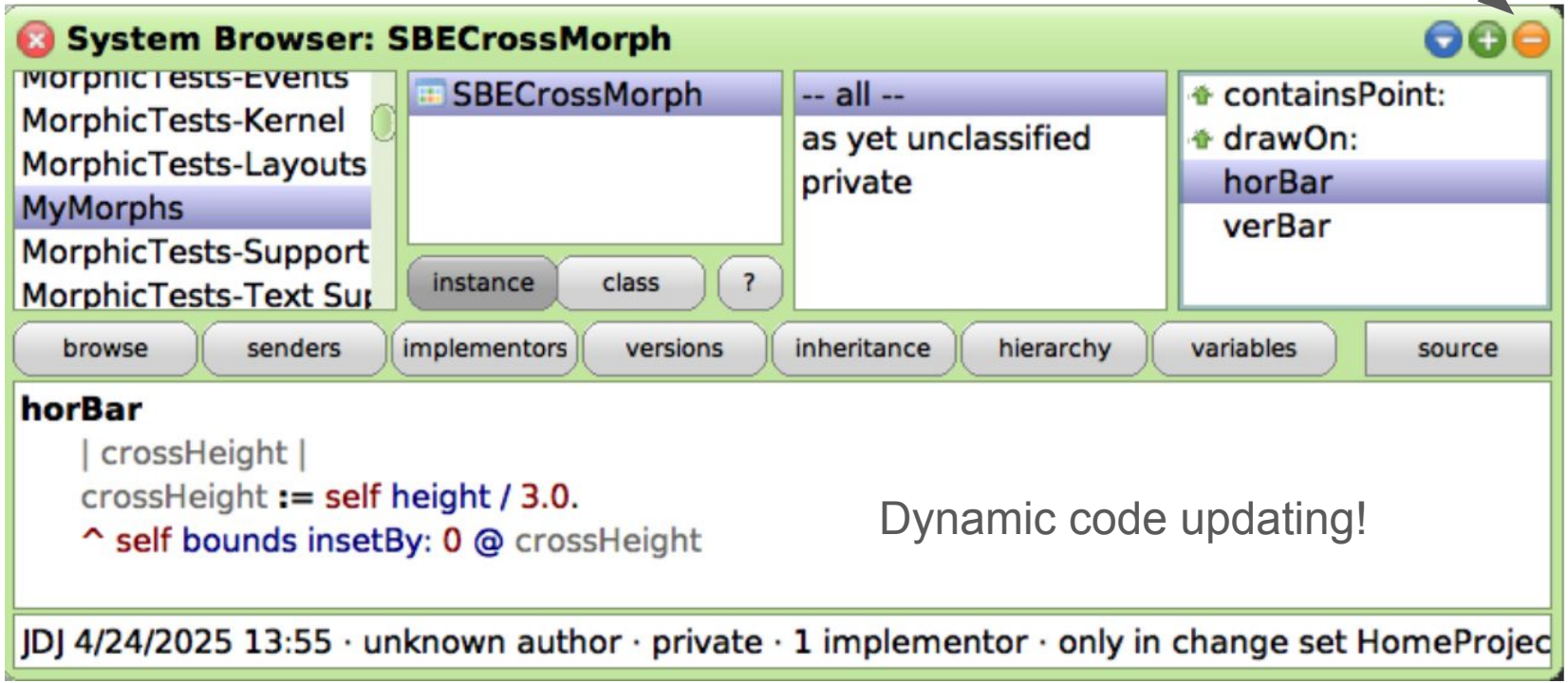


Smalltix: Huge Numbers of Tiny Files and Processes



Four Favourite Features of Smalltalk

Persistence! Uniformity! GUI Openness!



Dynamic code updating!

Four Favourite Features of Smalltalk (3 already in Unix!)

Persistence!

```
fwrite()
fflush()
```

Uniformity!

```
/proc/123
 |- cmdline
 |- cpu
 |- fd
 |- maps
 |- mem
  …
```

GUI Openness!*
(*only via Smalltix)

```
main_window/
 |- info_pane/
 |- titlebar/
 |- submit_btn/
 |- close_btn/
 |- scrollbar/
  …
```

Dynamic code updating!

```
vim application.c
make application
./application
```

Improvising Unix into a Smalltalk VM

Hijack the filesystem as object storage.

Hijack the program loader as a “method activator”.

Improvising Unix into a Smalltalk VM

Hijack the filesystem as object storage.

Hijack the program loader as a “method activator”.

WATCH AND TREMBLE… (demo)

Squeak By Example “SBECrossMorph”

```
7   verticalBar := self bounds insetBy: crossWidth@0.
8   aCanvas fillRectangle: horizontalBar color: self color.
9   aCanvas fillRectangle: verticalBar color: self color.
```

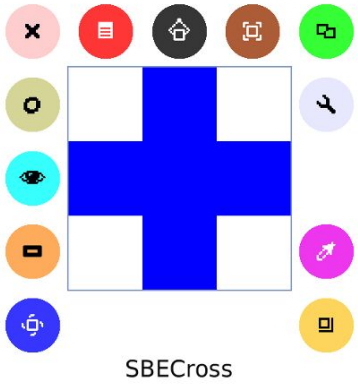


Figure 12.3: A SBECrossMorph with its halo; you can resize it as you wish.

Sending the bounds message to a morph answers its bounding box, which is an instance of Rectangle. Rectangles understand many messages

EXPLORER

- SMALLTIX
  - objects
    - aCross
      - bounds
      - class
    - boundsRect
      - class
      - corner
      - origin
    - brPoint
      - class
      - x
      - y
    - Morph
      - methods
        - bounds
        - height
        - superclass

OUTLINE

TIMELINE

README.md | send | horBar

objects > SBECrossMorph > methods > horBar

```
# | crossHeight |
# crossHeight := (self height / 3.0) rounded.
# ^ self bounds insetBy: 0 @ crossHeight.
#
self=$1
tmp1=$(./send $self height)
tmp2=$(./send $tmp1 / float/3.0)
crossHeight=$(./send $tmp2 rounded)
tmp4=$(./send int/0 @ $crossHeight)
tmp5=$(./send $self bounds)
./send $tmp5 insetBy- $tmp4
```

drj@JoelZ13: ~/repos/smallti

```
drj@JoelZ13:~/repos/smalltix/objects$ aRect=$(./send aCross horBar)
```

EXPLORER

SMALL...

- objects
  - Morph
    - methods
      - bounds
      - height
    - superclass
  - obj1
  - obj2
  - obj3
  - obj4
    - class U
    - corner U
    - origin U
  - Object
    - methods
      - basicNew
    - objCounter M

OUTLINE

TIMELINE

README.md | send | horBar | class U

objects > obj4 > class

```
1   Rectangle
```

drj@JoelZ13: ~/repos/smallti

```
+++++++++ [[ obj4 == \i\n\t ]]
+++++++++ [[ obj4 == \f\l\o\a\t ]]
++++++++++ ./bind obj4 setOrigin-corner-
+++++++++ method=Rectangle/methods/setOrigin-corner-
+++++++++ Rectangle/methods/setOrigin-corner- obj4 obj2 obj3
drj@JoelZ13:~/repos/smalltix/objects$ echo $aRect
obj4
drj@JoelZ13:~/repos/smalltix/objects$
```

Doesn’t strictly have to be Smalltalk…

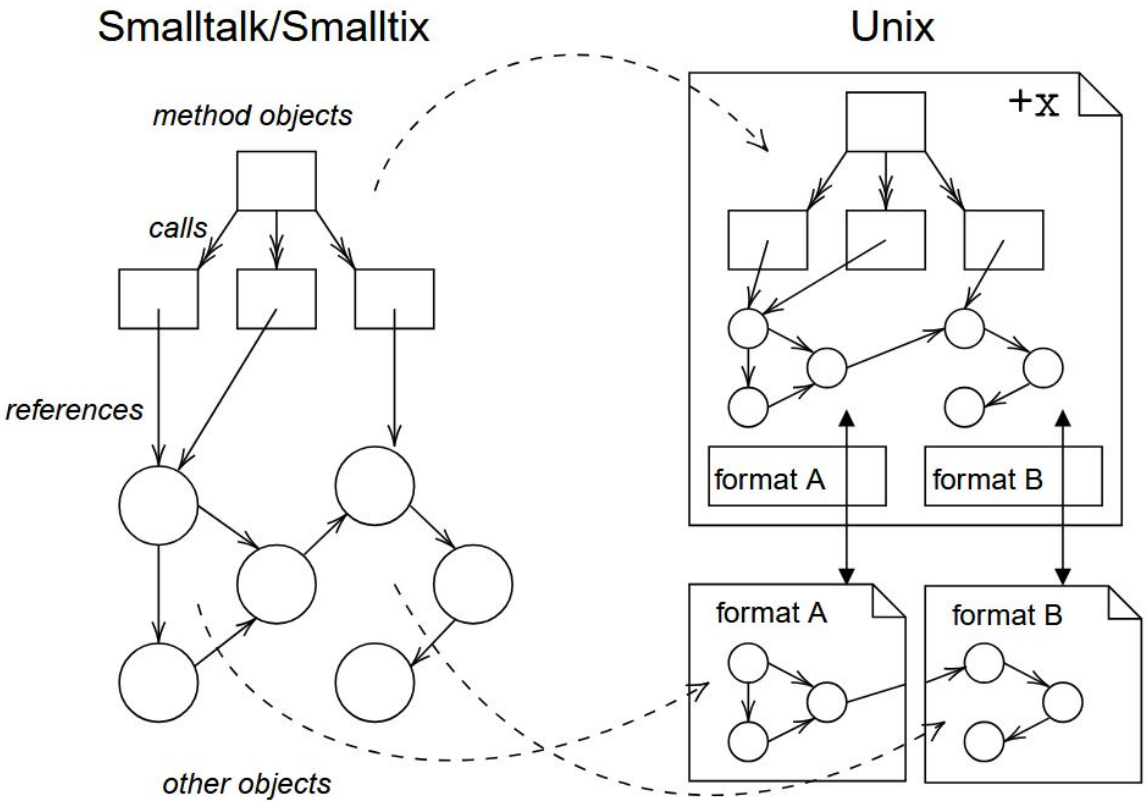
All programming systems (even Smalltalk!) speak “filesystem” under Unix hegemony.

Executable binaries can be compiled from any language.

Executable scripts can be written in any language.

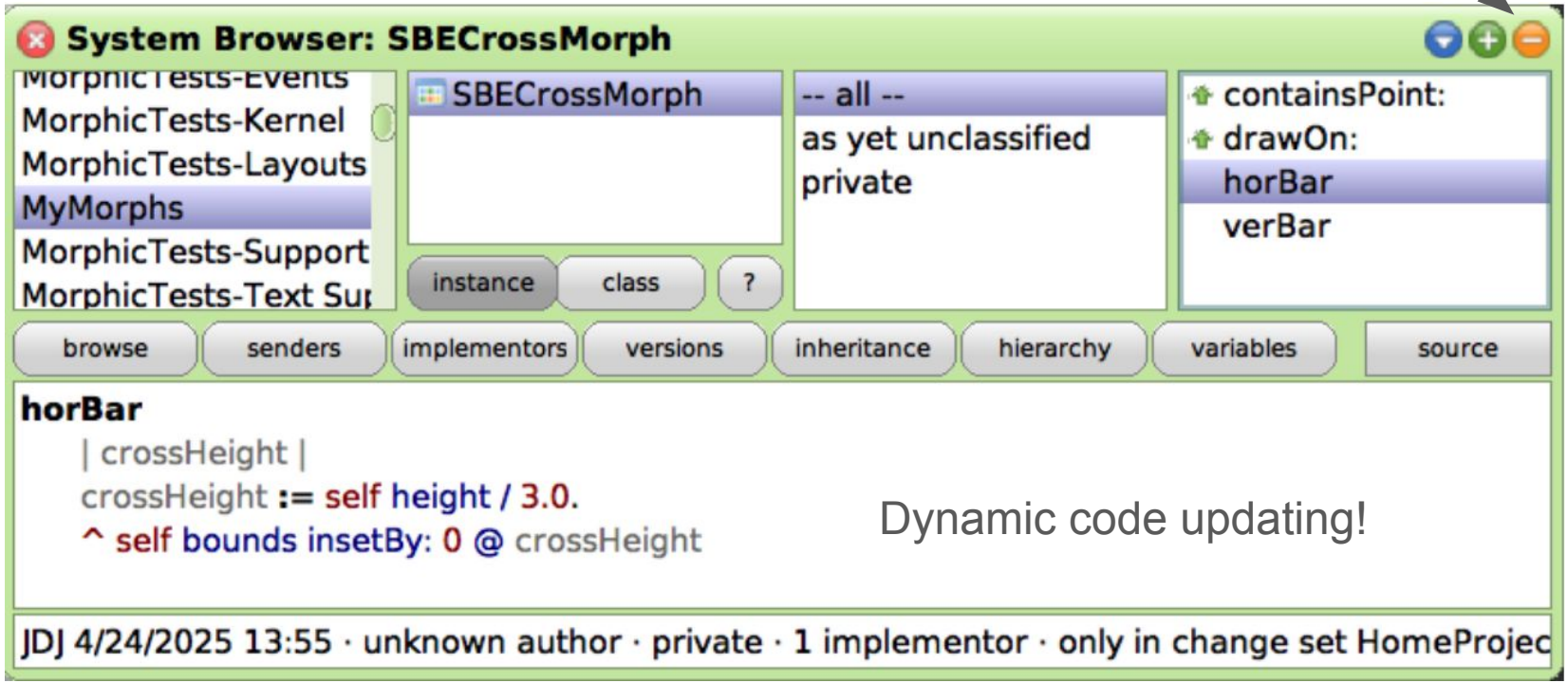
Language-agnostic (for free) down to the “method” level.

Unix Process = Specialised Method Tree Activator?



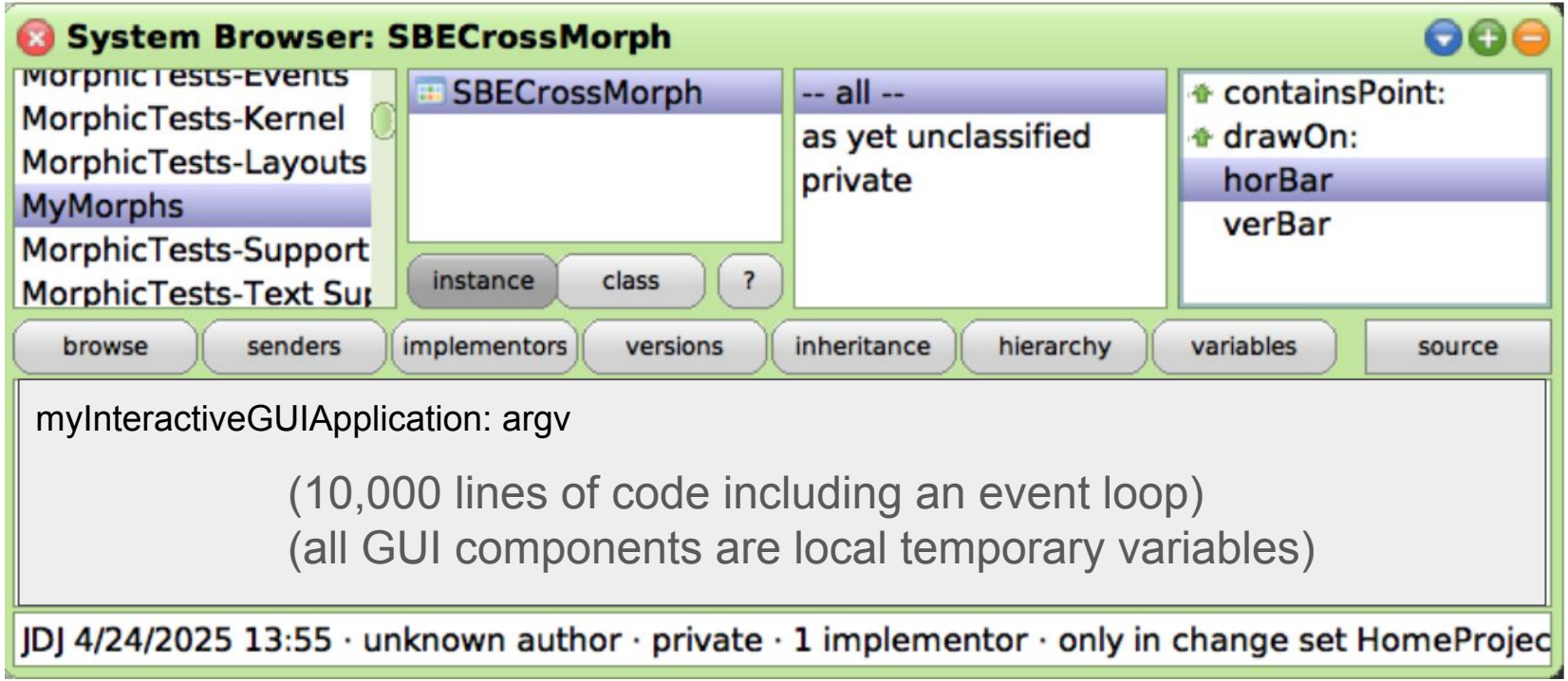
Four Favourite Features of Smalltalk

Persistence! Uniformity! GUI Openness!



Dynamic code updating!

GUIs should not be stuffed into a single method!



Conclusion

- Smalltalk is nice, but we're stuck with Unix.
- Abstract similarity: directories=objects, executables=methods, processes=activations.
- Concrete size discrepancy. Fragmentation. Files/processes heavyweight "large objects", unlike fine-grained Smalltalk objects/activations.
- If we try Smalltix anyway, we get Smalltalk conveniences for free: persistence, uniformity, dynamic software updating, GUI openness.
- Promising practical experiments!
- Interesting research paths in optimising away the scary overhead!