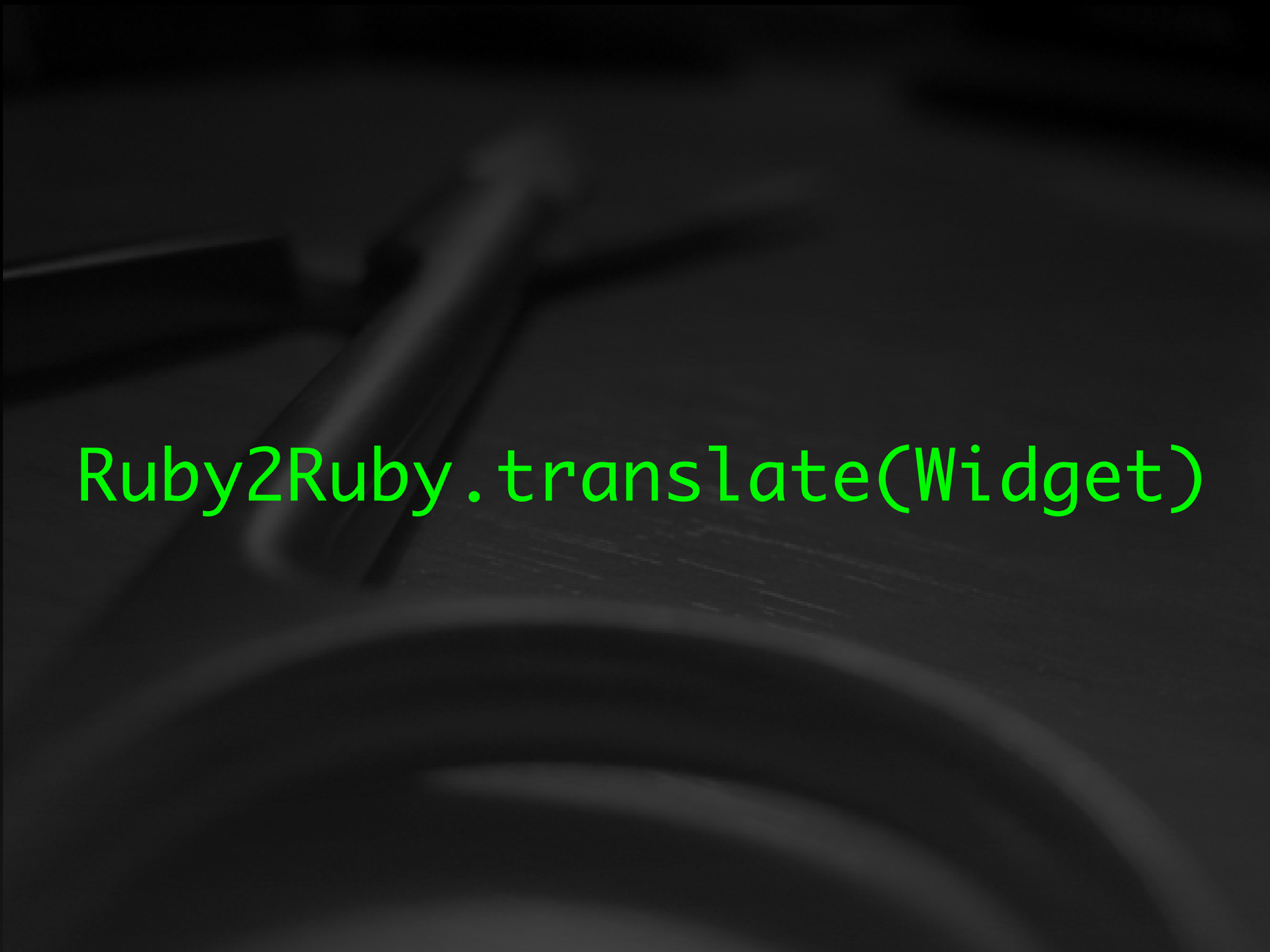
Code Generation

The Safety Scissors Of Metaprogramming

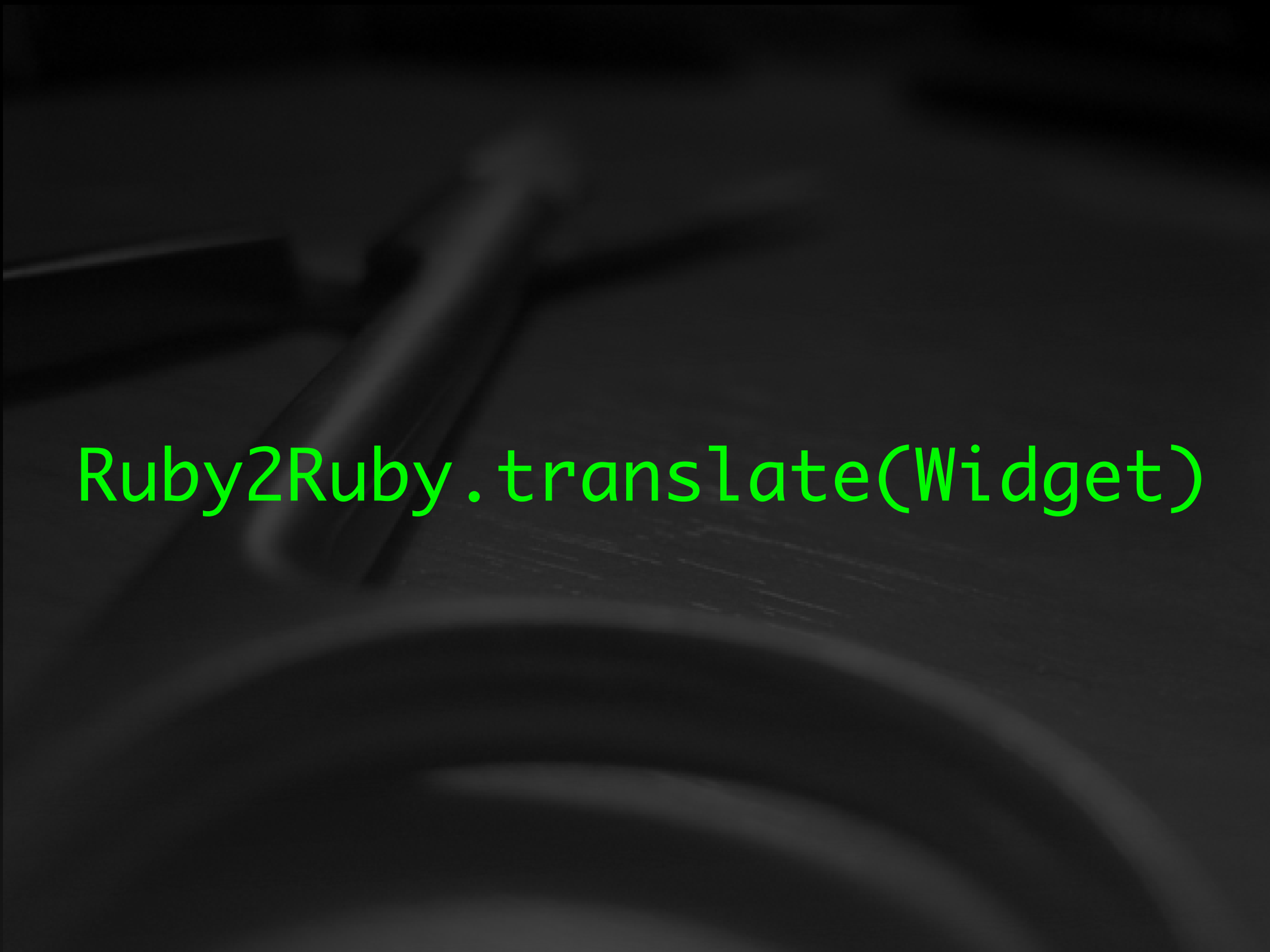

Ruby2Ruby

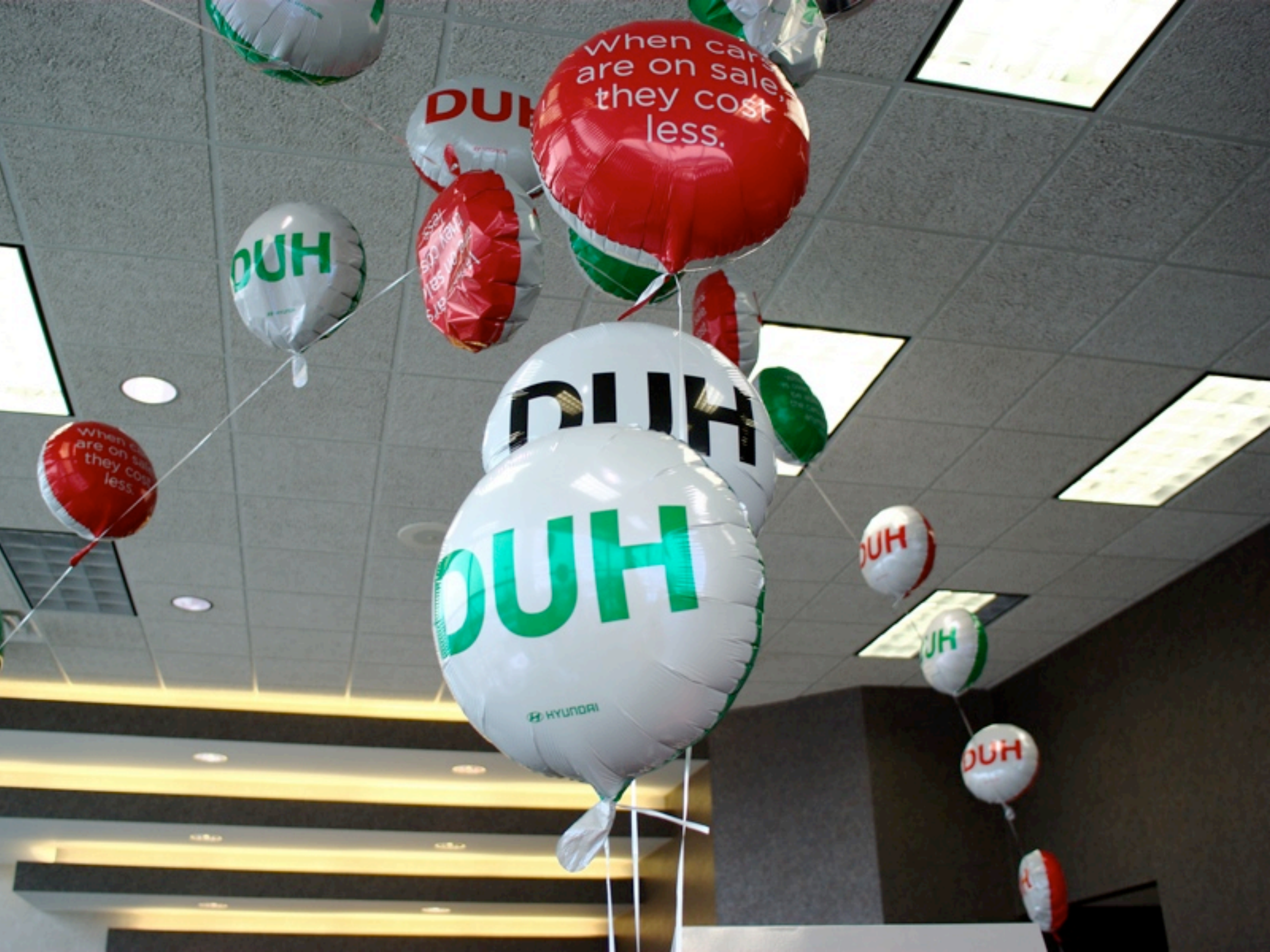
```
(lambda do
  puts “hello world”
end).to_ruby
```

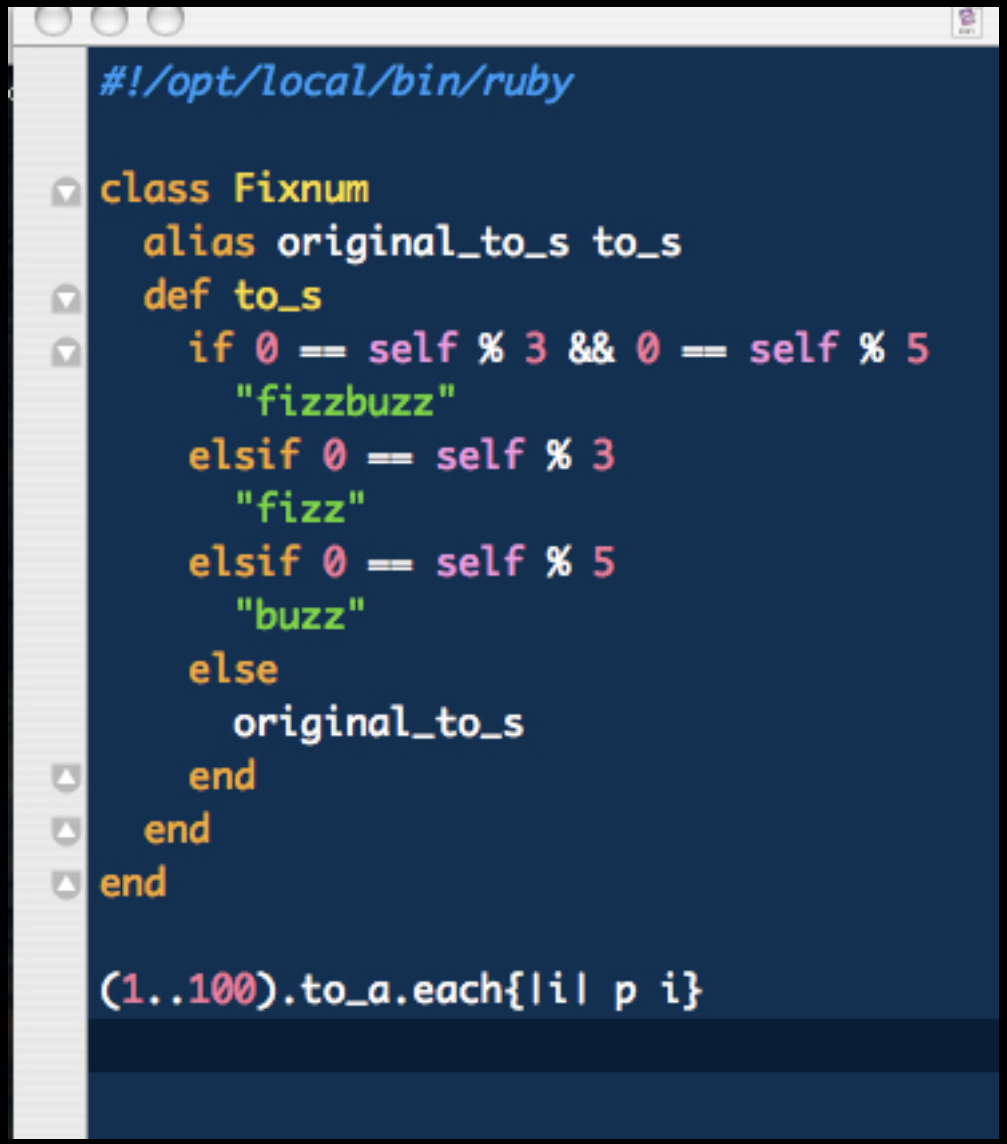
```
class Widget < ActiveRecord::Base
end
```


Ruby2Ruby.translate(Widget)

```
class Widget < ActiveRecord::Base
  has_one :foo
end
```

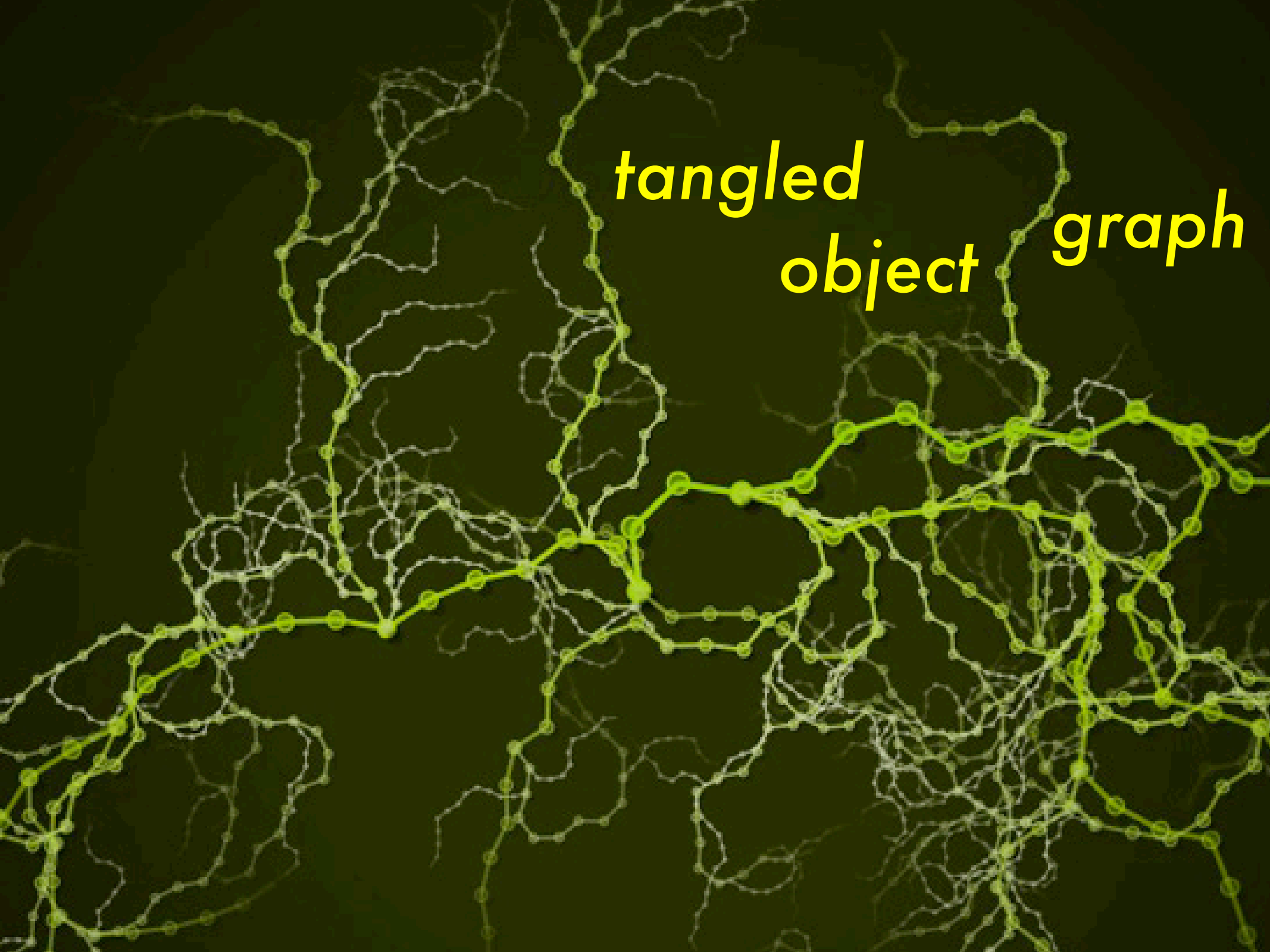

Ruby2Ruby.translate(Widget)


when cars are on sale, they cost less.
DUH
DUH
DUH
DUH
DUH
DUH
HYUNDAI


#!/opt/local/bin/ruby
class Fixnum
alias original_to_s to_s
def to_s
if 0 == self % 3 && 0 == self % 5
"fizzbuzz"
elsif 0 == self % 3
"fizz"
elsif 0 == self % 5
"buzz"
else
original_to_s
end
end
end
(1..100).to_a.each{|i| p i}


overkill


monkeypatching


tangled
object
graph

stop monkey-patching!

nothing can ever protect you

from bad programming

nothing can ever protect you

from bad programming

VIRTUOUS CODE

Monkeypatching is Destroying Ruby

(The title of this post is intended to be deliberately provocative, as well as being a nod to Steven Colbert's "The People Destroying America" segments. It's provocative because I want to get people talking about this issue. I don't actually think that monkey patching is "destroying" Ruby, but I do think the proliferation of the technique has real and troubling implications for Ruby's future.)

"Monkey patching", for anyone who doesn't know, refers to the practice of extending or modifying existing code by changing classes at run-time. It is a powerful technique that has become popular in the Ruby community at least in part because the Ruby language makes it so easy. Any class can be re-opened at any time and amended in any way.

I believe the term first arose in the Python community, as a derogatory term for a practice which that community tended to frown on. The Ruby community, on the other hand, has embraced the term and the practice with enthusiasm. I'm starting to think that the Pythonistas' attitude may have been justified.

Here's what crystalized it for me. The other day I wrote a small Rails plugin (NullDB). It was inspired largely by another plugin, UnitRecord. UnitRecord is by Dan Manges, a talented Rails developer whom I have a lot of respect for.

UnitRecord is implemented almost entirely as a set of monkey patches. When invoked, it dynamically modifies several standard Ruby and Rails classes, including `ActiveRecord::Base`, `Test::Unit::TestCase`. As a result of this implementation, it is tightly coupled to the inner workings of ActiveRecord. A small change to Rails and it could cease to work, and such a failure would be difficult to debug. Indeed, one of the reasons I decided to write NullDB was because of just such a failure.

In writing NullDB, I discovered that I could achieve the same functionality without resorting to monkey patching. Instead of modifying existing classes, it implements the Rails Database Adapter API. The finished library is shorter than UnitRecord, is composed entirely of implemetations of public APIs, and contains zero monkey patches. The code is also easier to understand, in my opinion, because of the lack of metaprogramming.

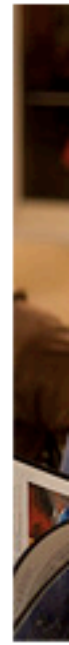
alias_method_chain :alias_method_chain, :awesome (or, how I learned to stop worrying and made Python nation and anyone else afraid of monkey-patching my bitch)

February 27th, 2008

There's a crazy idea out there that monkey-patching is bad. alias_method_chain is perhaps the worst offender. It allows you to easily decorate an object with behavior at runtime. The problem occurs when maintaining code that is all alias_method_chain'ed out - such as ActiveRecord::Base and the gazillion plugins that monkey patch it. It's tough to figure out what method you're actually aliasing away, and god help you if you rely on ordering.

Some may criticize, but I prefer to Do Cool Shit™

I present to you a deliciously self-referential way to keep your alias_method_chain headaches at bay:



I liv
Fra
star
nov
also
and

tests

specs

tests

Y U NEED

specs

TESTING

I FIND YOUR LACK OF TESTS DISTURBING.

tests rawk

debuggers suck

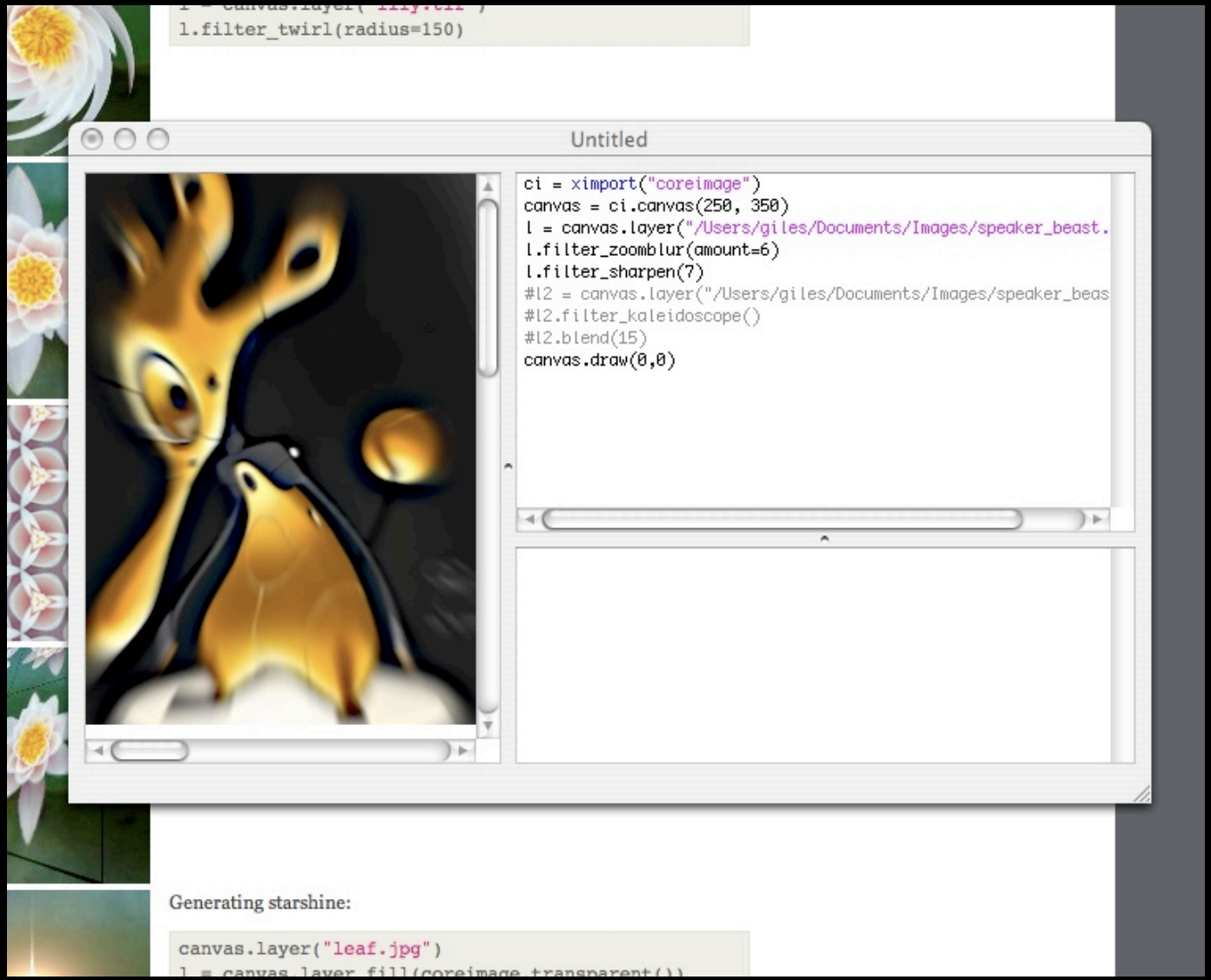
debuggers

suck

monkey-patch
monkey-patching
itself

meta-monkey-patching


rubinius


l.filter_twirl(radius=150)
Untitled
ci = ximport("coreimage")
canvas = ci.canvas(250, 350)
l = canvas.layer("/Users/giles/Documents/Images/speaker_beast.
l.filter_zoomblur(amount=6)
l.filter_sharpen(7)
#l2 = canvas.layer("/Users/giles/Documents/Images/speaker_beas
#l2.filter_kaleidoscope()
#l2.blend(15)
canvas.draw(0,0)
Generating starshine:
canvas.layer("leaf.jpg")

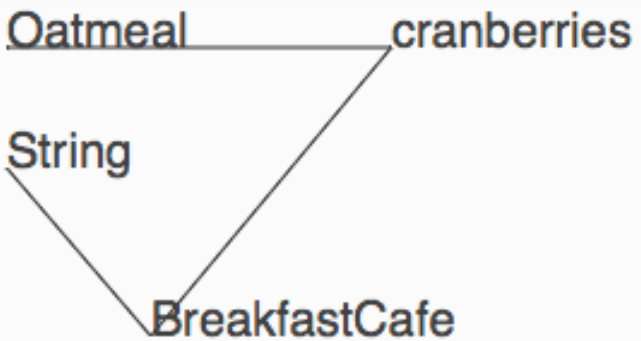
nodebox

code == data

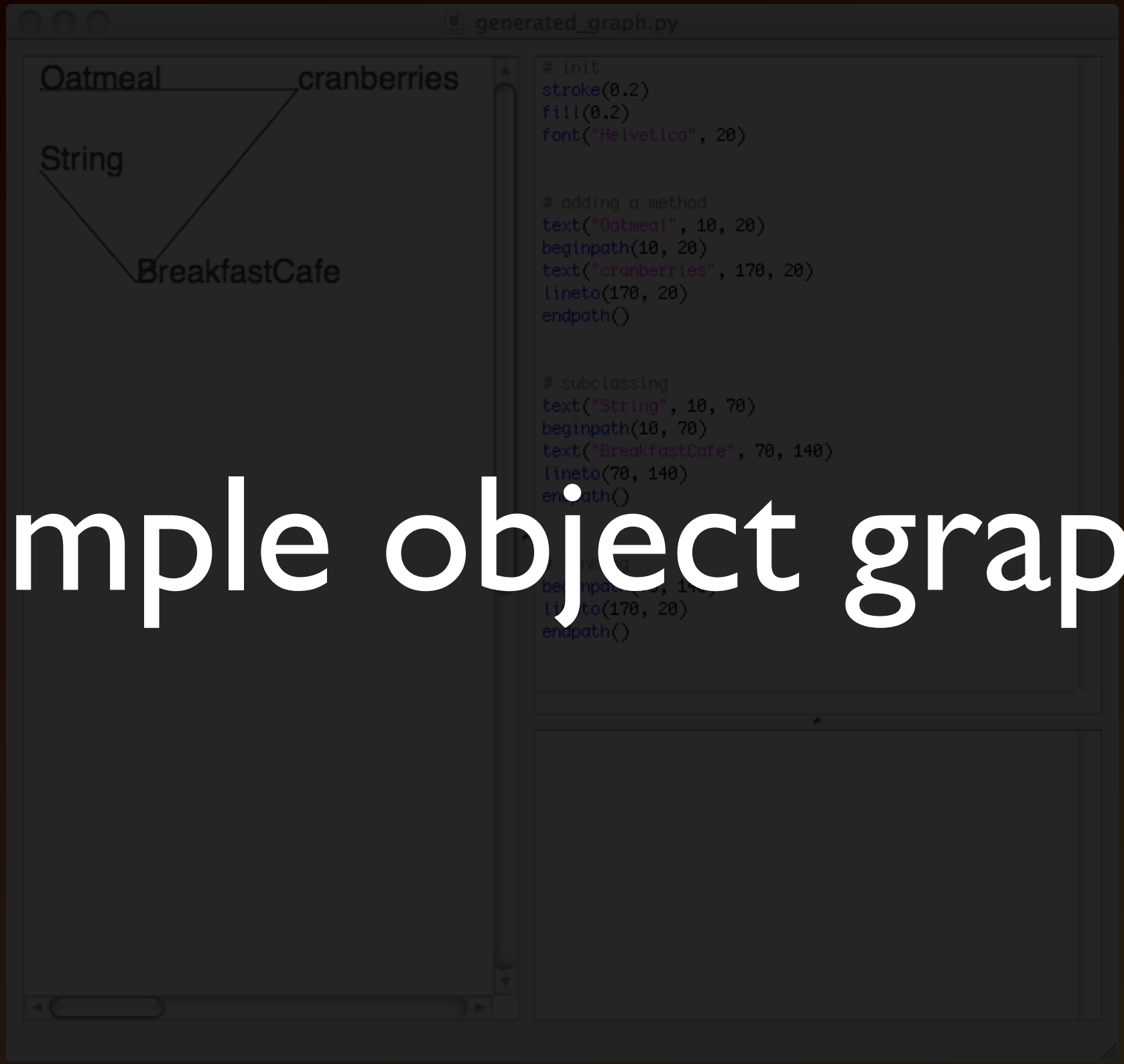
at this point, I just demoed code from my blog.
below is a link to the post:

http://gilesbowkett.blogspot.com/2008/03/simple-metaprogramming-logger-with.html
(code samples are included)

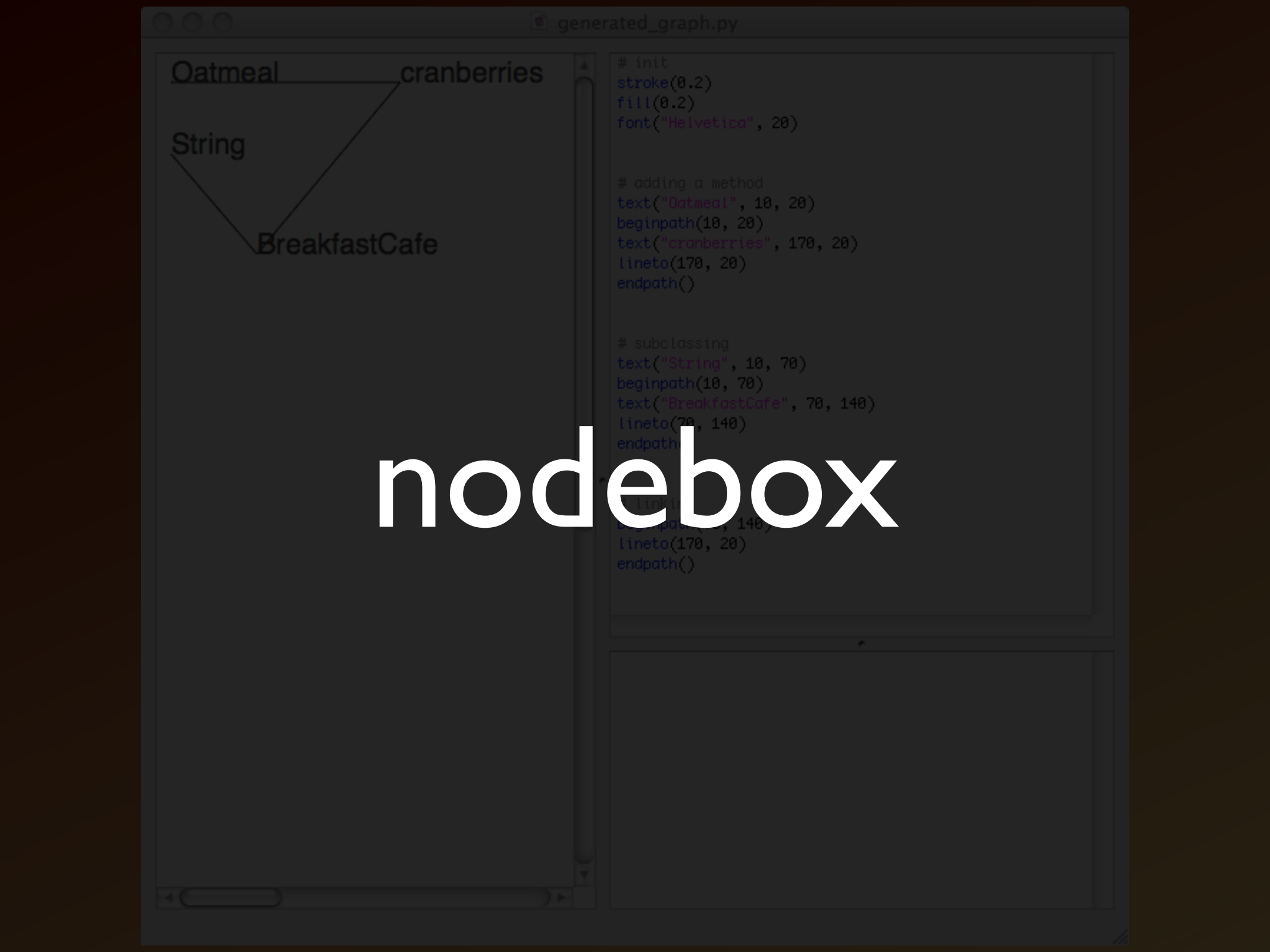
generated_graph.py

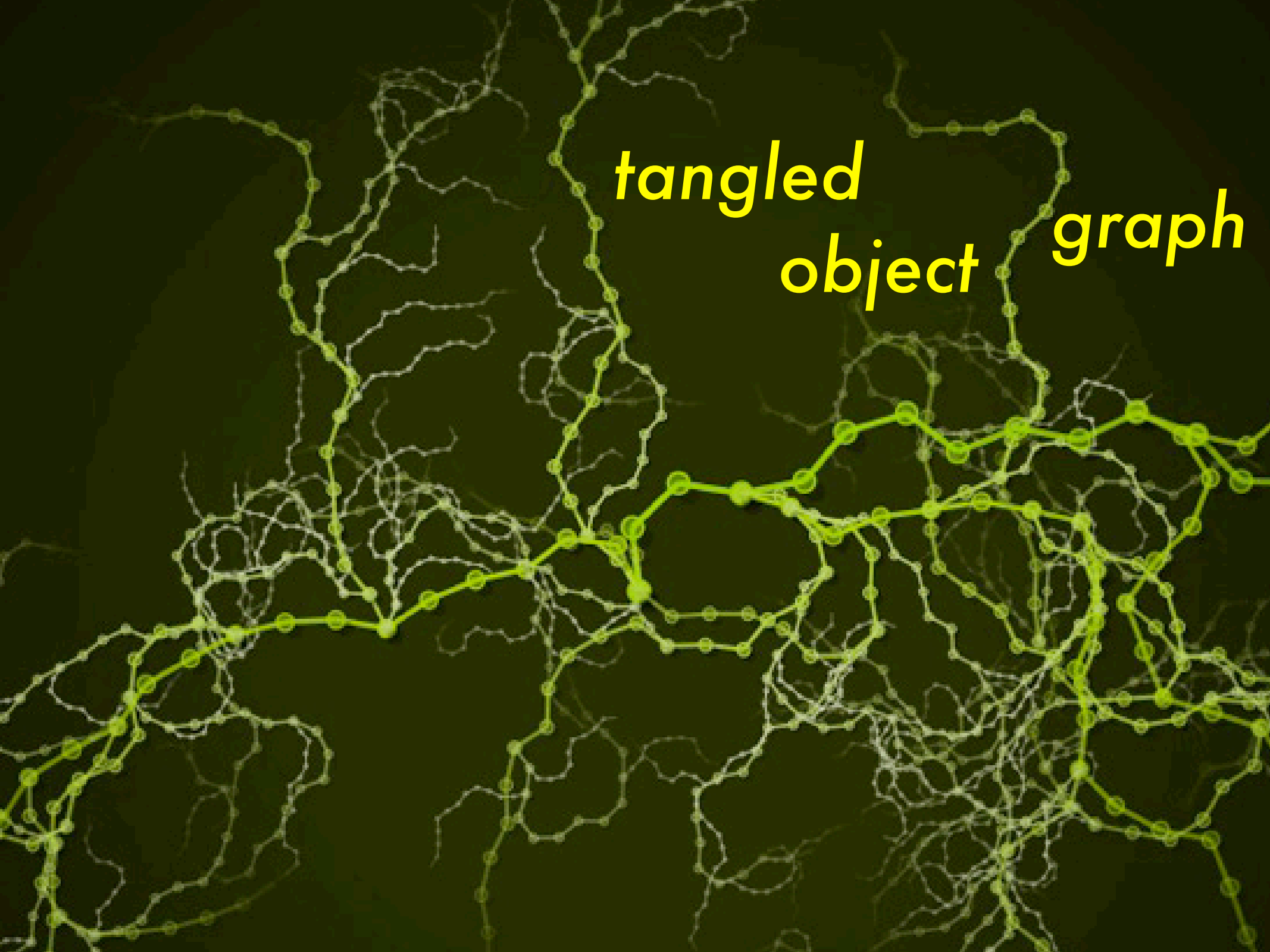


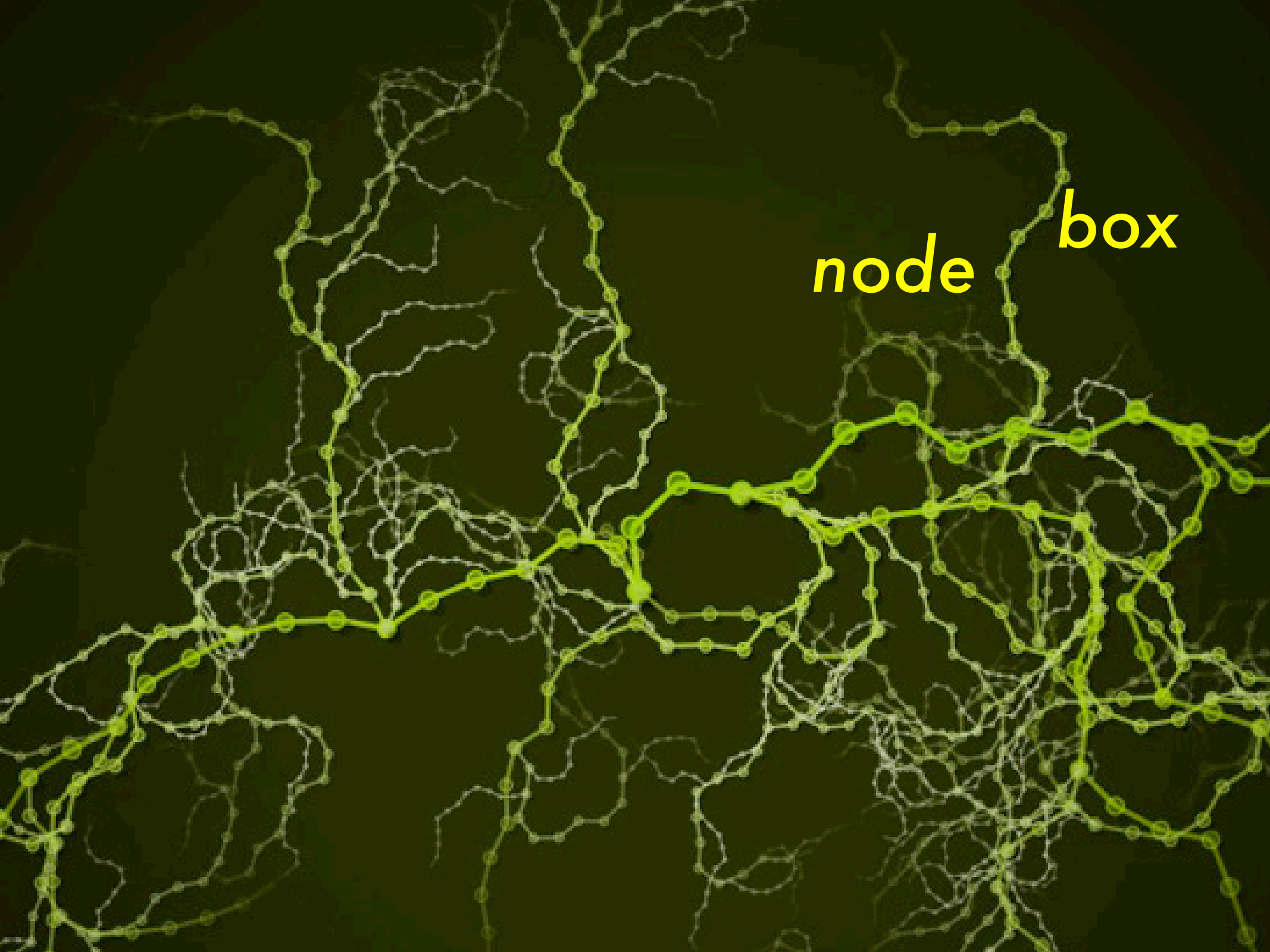

```
# init
stroke(0.2)
fill(0.2)
font("Helvetica", 20)


# adding a method
text("Oatmeal", 10, 20)
beginpath(10, 20)
text("cranberries", 170, 20)
lineto(170, 20)
endpath()


# subclassing
text("String", 10, 70)
beginpath(10, 70)
text("BreakfastCafe", 70, 140)
lineto(70, 140)
endpath()


# linking
beginpath(70, 140)
lineto(170, 20)
endpath()
```



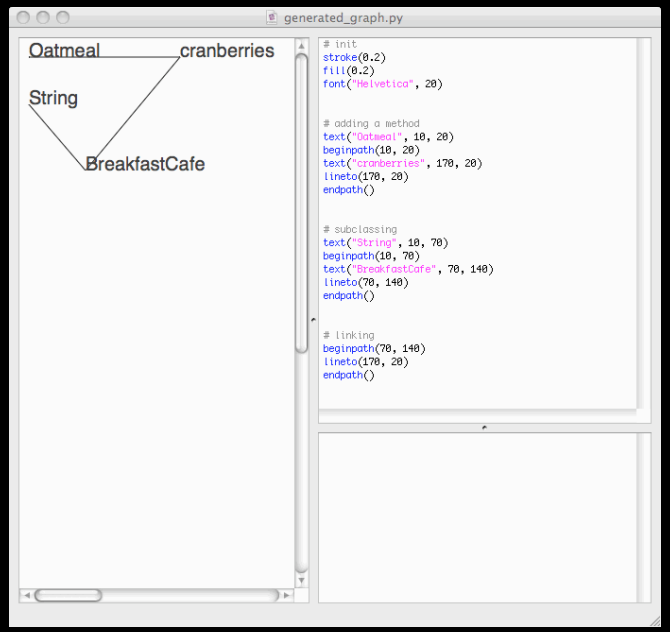
simple object graph

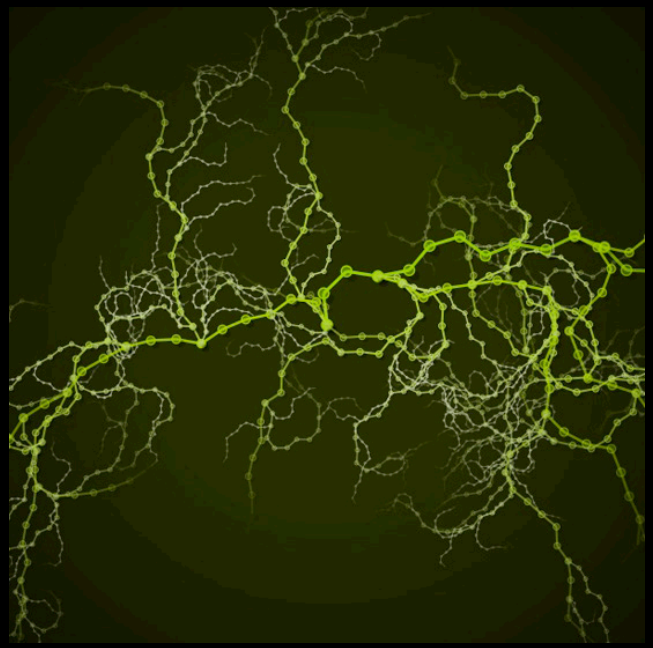

generated_graph.py
Oatmeal
cranberries
String
BreakfastCafe
# init
stroke(0.2)
fill(0.2)
font("Helvetica", 20)
# adding a method
text("Oatmeal", 10, 20)
beginpath(10, 20)
text("cranberries", 170, 20)
lineto(170, 20)
endpath()
# subclassing
text("String", 10, 70)
beginpath(10, 70)
text("BreakfastCafe", 70, 140)
lineto(70, 140)
lineto(170, 20)
endpath()
nodebox

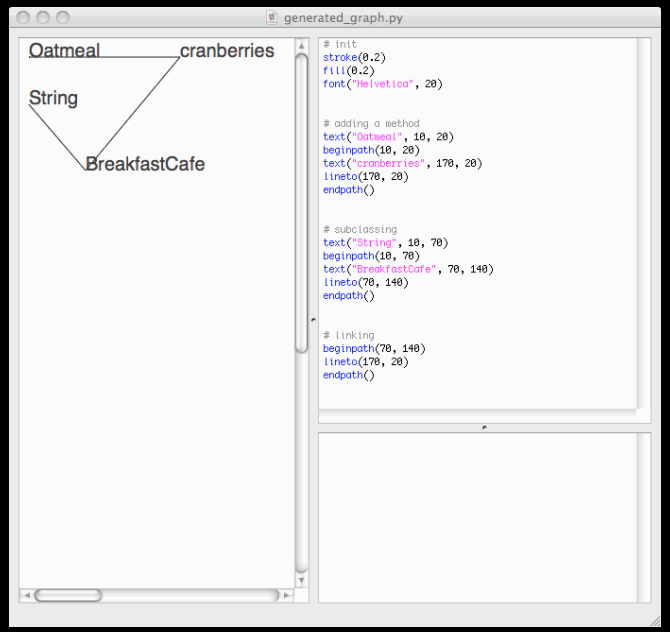

tangled
object
graph

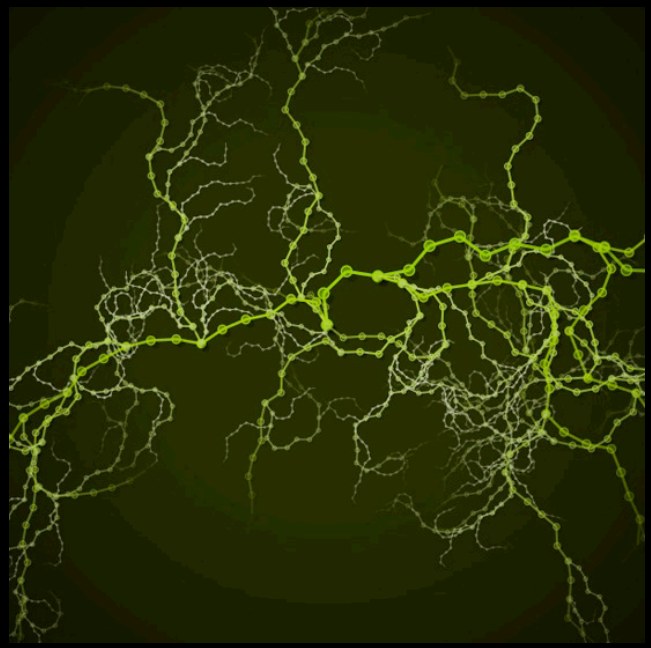

node
box

same tool


generated_graph.py
Oatmeal
cranberries
String
BreakfastCafe
# init
stroke(0.2)
fill(0.2)
font("Helvetica", 20)
# adding a method
text("Oatmeal", 10, 20)
beginpath(10, 20)
text("cranberries", 170, 20)
lineto(170, 20)
endpath()
# subclassing
text("String", 10, 70)
beginpath(10, 70)
text("BreakfastCafe", 70, 140)
lineto(70, 140)
endpath()
# linking
beginpath(70, 140)
lineto(170, 20)
endpath()

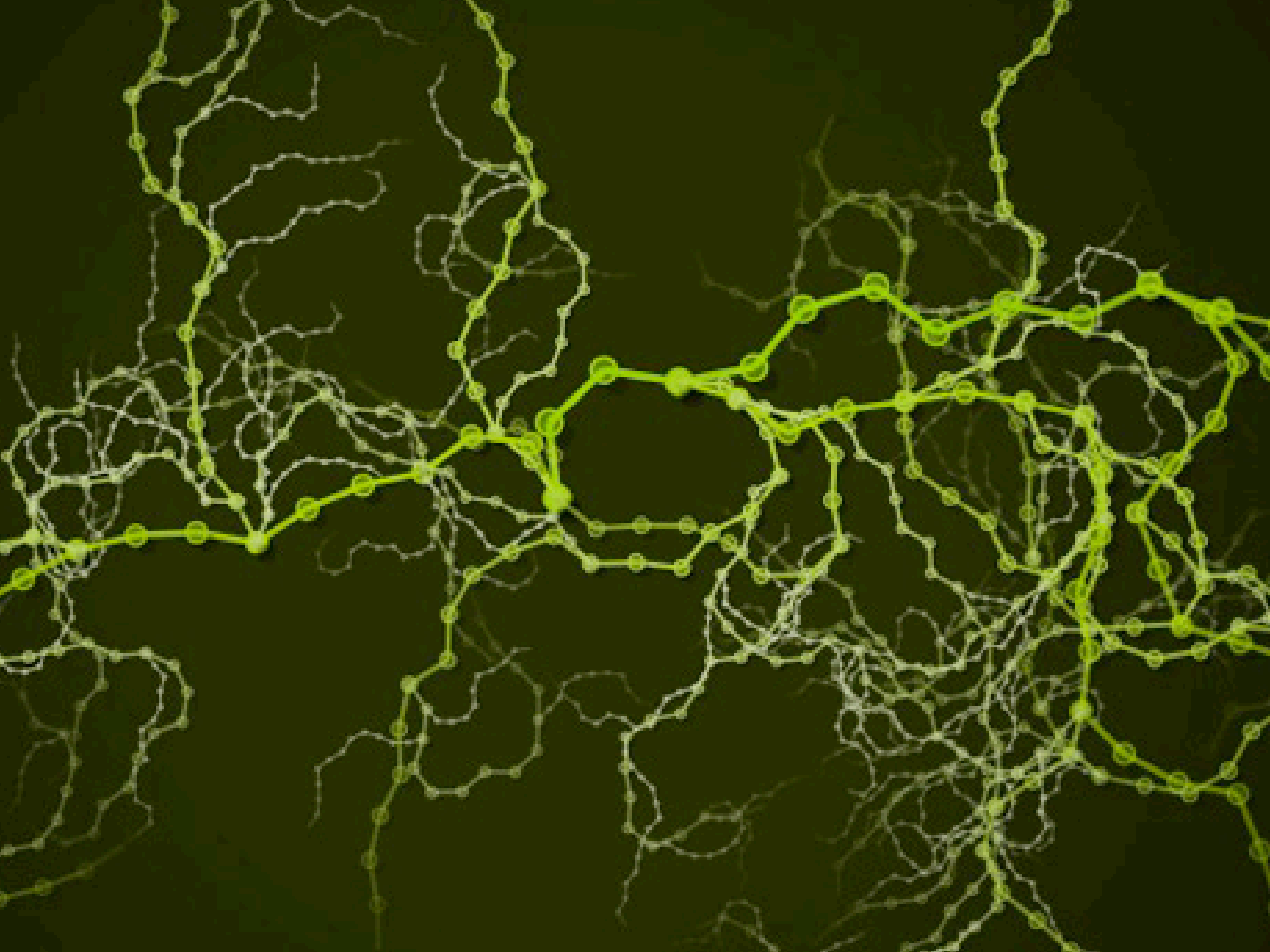


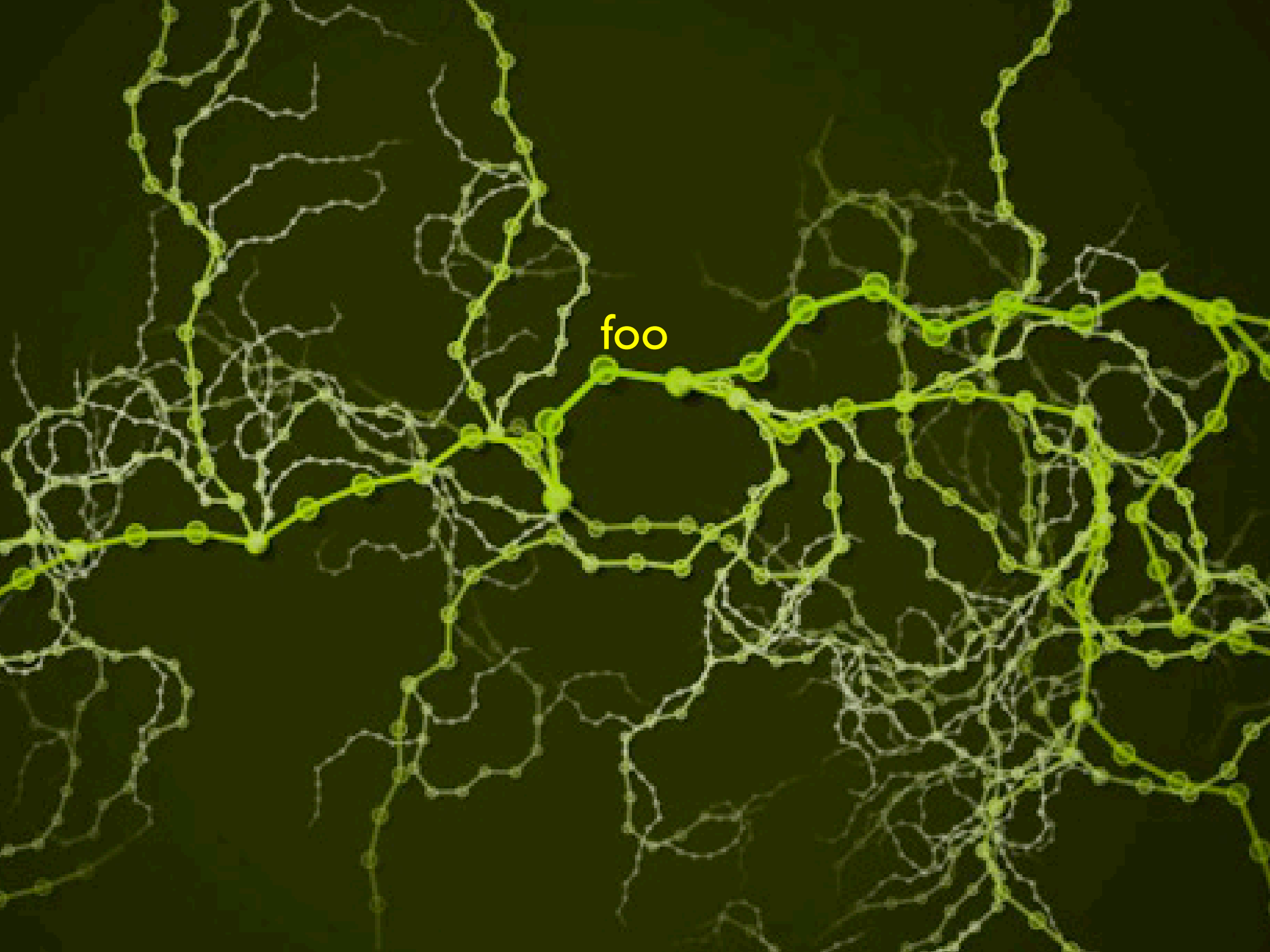


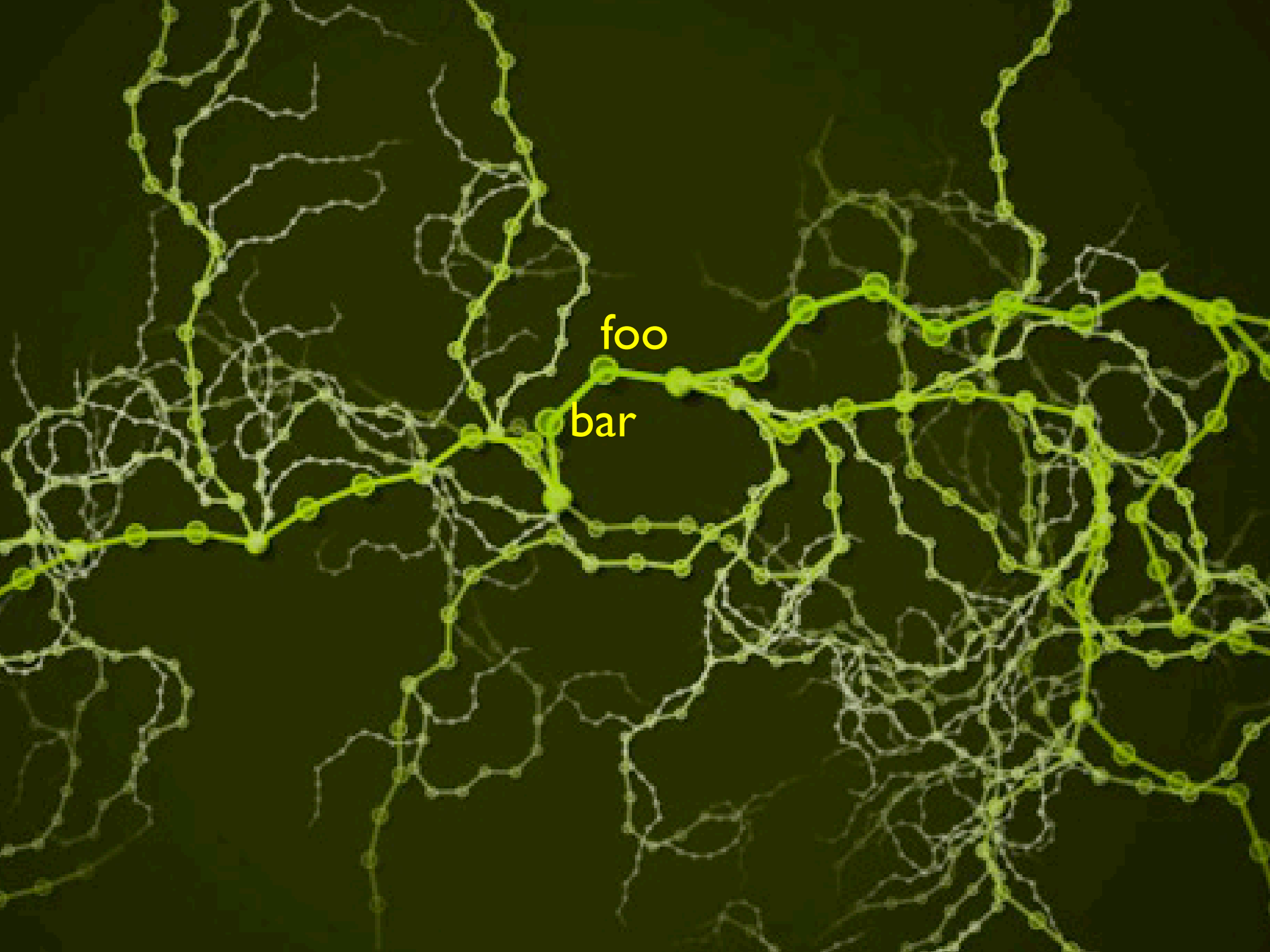


different output

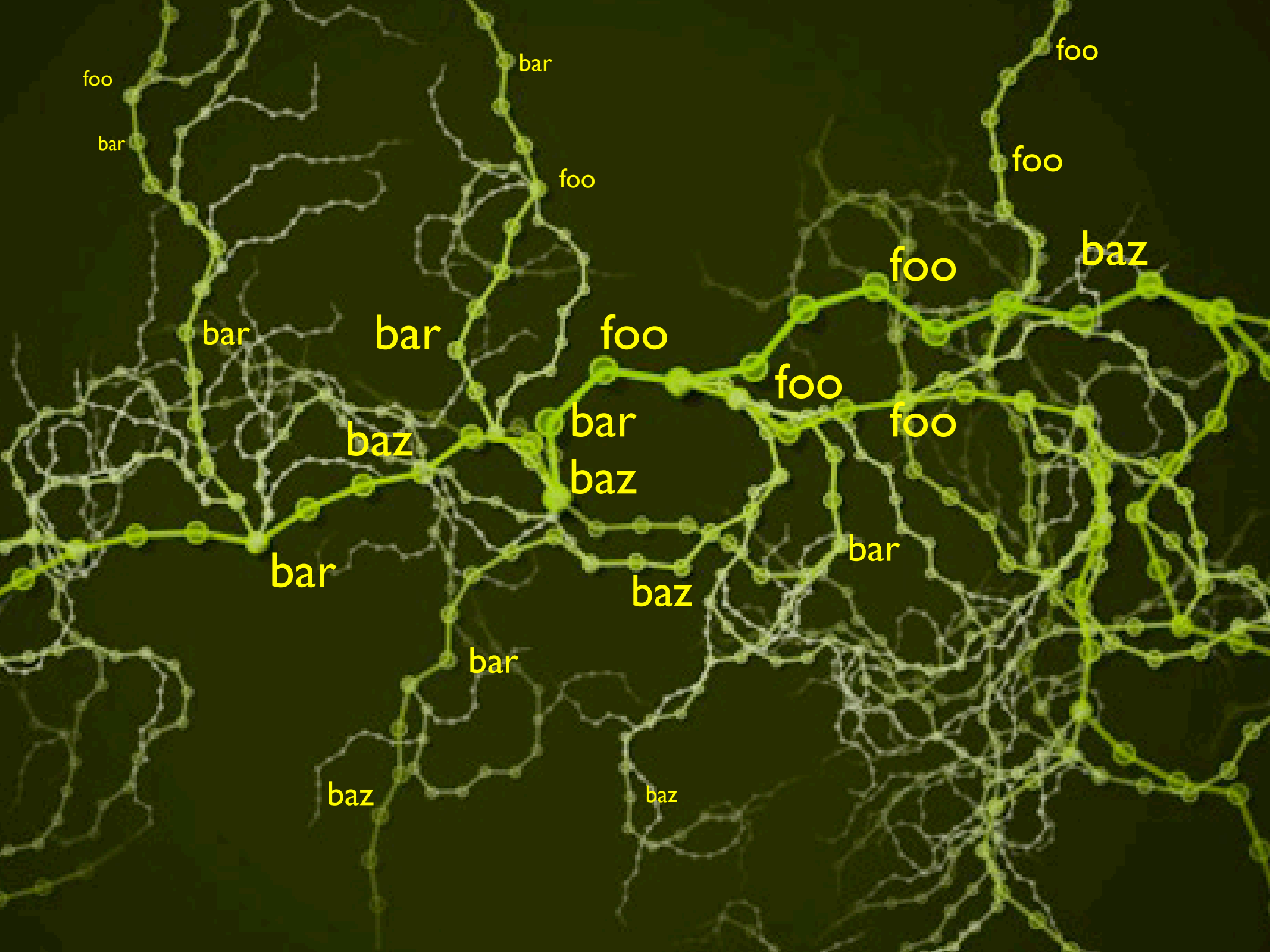
proof of concept




foo


foo
bar

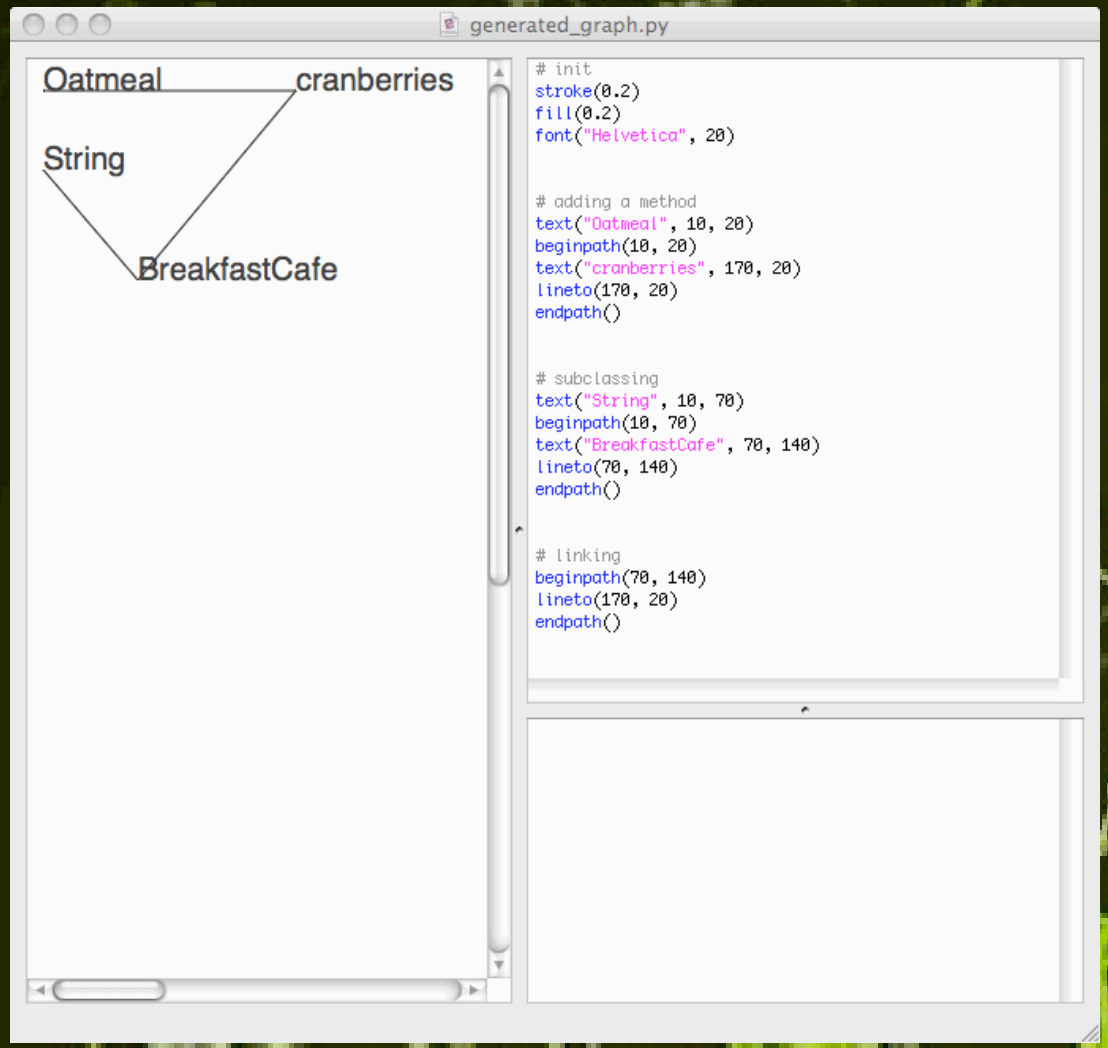

foo
bar
baz


foo
bar
bar
foo
bar
bar
foo
baz
bar
baz
bar
bar
baz
foo
foo
foo
baz
bar
baz
baz

debuggers suck

some people want debuggers

“Ruby has no tool support!”


generated_graph.py
Oatmeal
cranberries
String
BreakfastCafe
# init
stroke(0.2)
fill(0.2)
font("Helvetica", 20)
# adding a method
text("Oatmeal", 10, 20)
beginpath(10, 20)
text("cranberries", 170, 20)
lineto(170, 20)
endpath()
# subclassing
text("String", 10, 70)
beginpath(10, 70)
text("BreakfastCafe", 70, 140)
lineto(70, 140)
endpath()
# linking
beginpath(70, 140)
lineto(170, 20)
endpath()

1 day

shock the monkey

Powerful Tools

Powerful Tools

- Rubinius

Powerful Tools

- Rubinius
- Nodebox

Powerful Tools

- Rubinius
- Nodebox
- code == data

- code == data

code == data

1998

generated HTML

automated cvs and rcs

Code Generation in Action

Code generation is the technique of using or writing programs that write source code. Code generators are tools built to serve engineers in the creation of applications. Just as woodworkers use customized tools called jigs to allow them to build furniture more quickly and accurately, code generators allow engineers to concentrate on building the

EJB

(2003)

legacy app: 150 tables

ejb: 7 files per entity

7 * 150 = 1050 Java files

by hand: 3+ man-years

generators: 2 man-months

four generator types

- code munger generator
- inline code expansion generator
- mixed-code generator
- partial-class generator
- tier generator

generating UIs

generating unit tests

embedding SQL with generators

creating database access generators

generating web services layers

generating models from a business logic DSL

sound familiar?


RAILS

```
script/generate scaffold
```

```
script/generate resource
```

```
script/generate plugin
```

```
script/generate migration
```

script/generate

rails Command

acts_as_authenticated

restful_authentication

ActiveRecord SQL

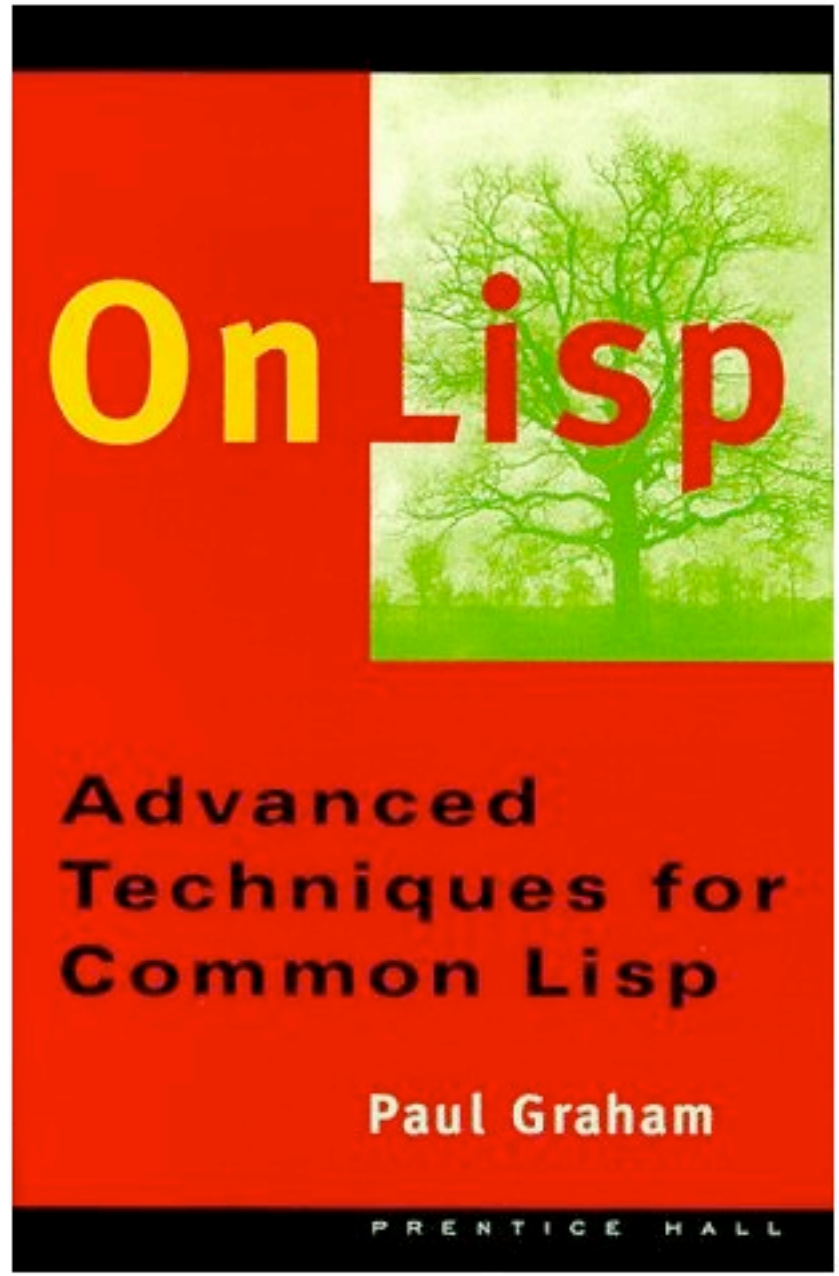
ActionPack Helpers

- ActiveRecordHelper
- AssetTagHelper
- AtomFeedHelper
- BenchmarkHelper
- CacheHelper
- CaptureHelper
- DateHelper
- DebugHelper
- FormHelper
- FormOptionsHelper
- FormTagHelper
- JavascriptHelper
- NumberHelper
- PrototypeHelper
- RecordTagHelper
- SanitizeHelper
- ScriptaculousHelper
- TagHelper
- TextHelper
- UrlHelper
- RecordIdentificationHelper

code generation FTW

code == data

```
(eq (code (data)))
```


On Lisp
Advanced Techniques for Common Lisp
Paul Graham
PRENTICE HALL

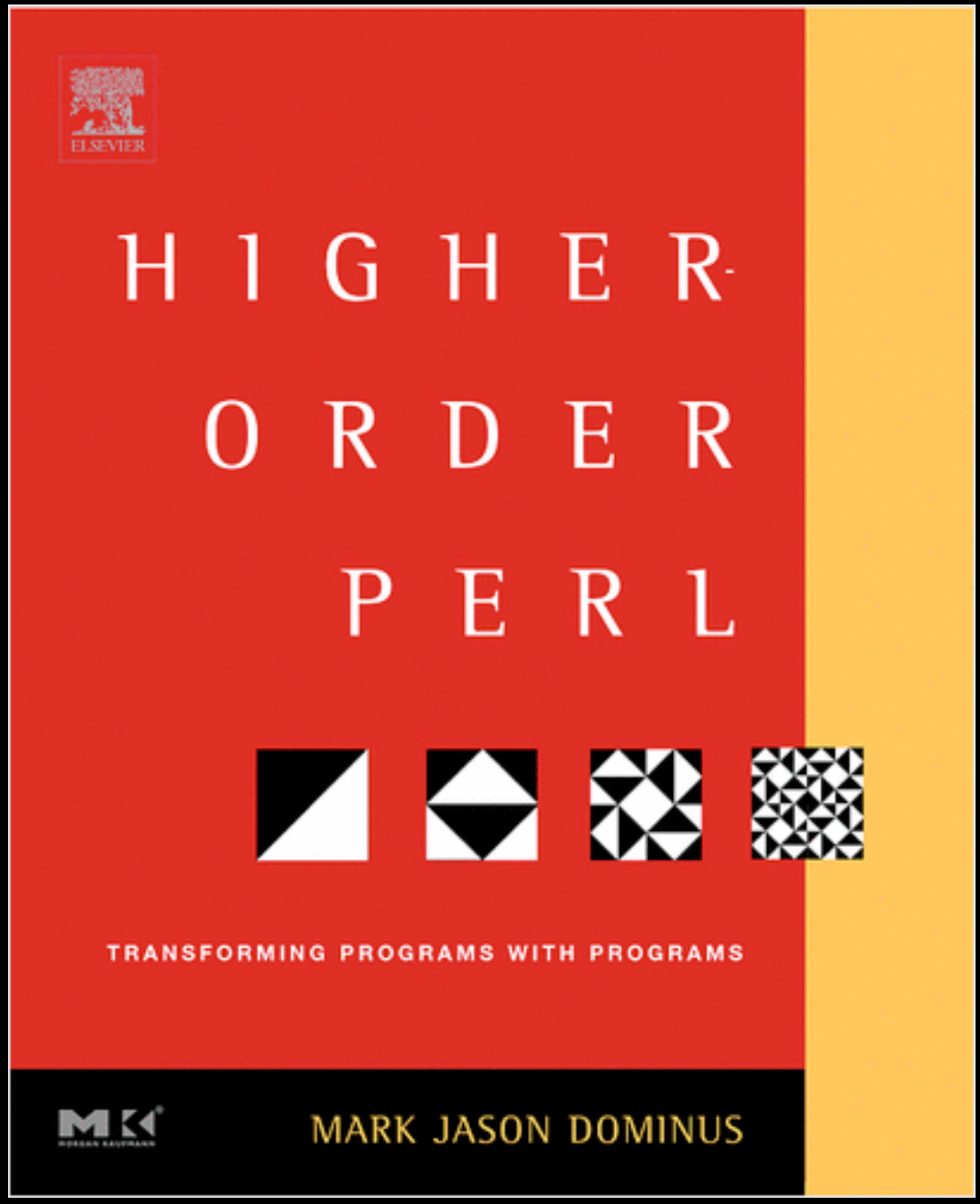
functions
which return functions
which return functions
which return functions
which return functions
which return functions
which return functions
which return functions
which return functions
which return functions

code
which generates data
which runs as code
which generates data
which runs as code
which generates data
which runs as code
which generates data
which runs as code
which generates data
which runs as code

Now here is the alternative. What if we instantiate a form in memory, let the cells compute the layout, and then traverse the form writing out a persistent mirror image of what we find, including computed layout coordinates? Business logic can be written out symbolically and read back in because thanks to Dr. McCarthy code is data. We avoid the redundant computations, but more importantly we now had a changed form specification as a second set of data instead of as a software release. Work on the original performance problem had serendipitously dispatched

lisp macros pwn
“metaprogramming”

and you can do
them in Perl!


ELSEVIER
HIGHER-ORDER PERL
TRANSFORMING PROGRAMS WITH PROGRAMS
MK
MORGAN KAUFMANN
MARK JASON DOMINUS

shocked? perl!

THE ULTIMATE


COMBINATOR

small teams

small teams

lisp macros

great programmers can write better programmers than they can hire

great programmers can write better programmers than they can hire

metaprogramming


great?


incredible
power

simple
clarity

simplicity

---

magic

skilled programmers can write better programmers than they can hire

metaprogramming

Code Generation in Action

Code generation is the technique of using or writing programs that write source code. Code generators are tools built to serve engineers in the creation of applications. Just as woodworkers use customized tools called jigs to allow them to build furniture more quickly and accurately, code generators allow engineers to concentrate on building the

code
which generates data
which runs as code
which generates data
which runs as code
which generates data
which runs as code
which generates data
which runs as code
which generates data
which runs as code

Unix.is_a? Lisp

```
# => true
```

Caveats!

Caveats!

- read the book, the technique is easy

Caveats!

- read the book, the technique is easy
- leveraging it is the clever part

Caveats!

- read the book, the technique is easy
- leveraging it is the clever part
- you can use Ruby as well as Rubinius

Caveats!

- read the book, the technique is easy
- leveraging it is the clever part
- you can use Ruby as well as Rubinius
- if Unix is a Lisp, the filesystem is a compiler

Essential Technique

Essential Technique

- puts

Essential Technique

- puts
- eval

Essential Technique

- puts
- eval
- ERB

Essential Technique

- puts
- eval
- ERB
- (unix pipes)

Essential Technique

- open mouth
- make noise
- profit