Meta-meta-programming

Dr Nic Williams

drnicwilliams.com

Dr Nic Academy

Boring definition

“Program that manipulates the real world”

Programming

Image: http://www.kodak.com/US/images/en/corp/1000nerds/lund/nerdGirl.jpg

Programming with Bash

```
$ cat program
#!/bin/bash
echo 1
echo 2
echo 3
echo 4
echo 5
$ program
1
2
3
4
5
```

consider a silly bash script that prints 5 numbers. Its silly, but its a program.

Less boring definition

“Program that manipulates the real world”

Programming

“Program that manipulates other programs”

Meta-programming

Metaprogramming with Bash

```
#!/bin/bash
echo '#!/bin/bash' >program
for ((I=1; I<=5; I++)) do
    echo "echo $I" >>program
done
chmod +x program
```

http://en.wikipedia.org/wiki/Metaprogramming

On the left is another bash script called “metaprogram”.
On the right we execute the script, and it creates a new text file called program. Which is the program we ran 2 minutes ago.

This might not look like meta-programming, but its my presentation so I make up the definitions. The first program manipulated or created the second program.

Metaprogramming with Bash

```
#!/bin/bash
echo '#!/bin/bash' >program
for ((I=1; I<=5; I++)) do
    echo "echo $I" >>program
done
chmod +x program
```

```
$ metaprogram
$ cat program
#!/bin/bash
echo 1
echo 2
echo 3
echo 4
echo 5
$ program
1
2
3
4
5
```

http://en.wikipedia.org/wiki/Metaprogramming

On the left is another bash script called “metaprogram”.
On the right we execute the script, and it creates a new text file called program. Which is the program we ran 2 minutes ago.

This might not look like meta-programming, but its my presentation so I make up the definitions. The first program manipulated or created the second program.

Generate Ruby with Ruby

```
$ script/generate model Person
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/person.rb
      create  test/unit/person_test.rb
      create  test/fixtures/people.yml
      create  db/migrate
      create  db/migrate/001_create_people.rb
```

You are doing meta-programming when you use a code generator. When you're developing Rails apps or Merb apps or RubyGems you can use code generators.

Generate Ruby with IDE

```
# TextMate’s Rails bundle
# TAB autocompletion gives...
hm # =>
has_many :objects, :class_name => "Objects",
                   :foreign_key => "class_name_id"

renp # =>
redirect_to(parent_child_path(@parent, @child))
```

Your text editor can provide meta-programming tools. Its a program that manipulates your program. If I use TextMate, and expand one of the many delicious Rails snippets from the very sexy new Rails bundle for TextMate, then I’m doing meta-programming.

TextMate

for rails

peepcode.com

^^ blatant advert ^^

Generate Ruby within Ruby

```
class Person
end

Person.module_eval <<-EOS
  def to_s
    "I'm a person, dammit!"
  end
EOS

Person.new.to_s # => "I'm a person, dammit!"
```

But the concept of meta-programming that most people think about is internal meta-programming. Where a programming language has its own tools to modify an application whilst the application is already executing.
On this slide, we're adding a to_s method to a preexisting class Person.

Two types of meta-programming

- Generation
- Reflection

So I like to think there are generally two types of meta-programming: external generation or modification of code, and internal, runtime modification of code. The later is called Reflection.

Reflection

1. Programming language entities are first class objects.

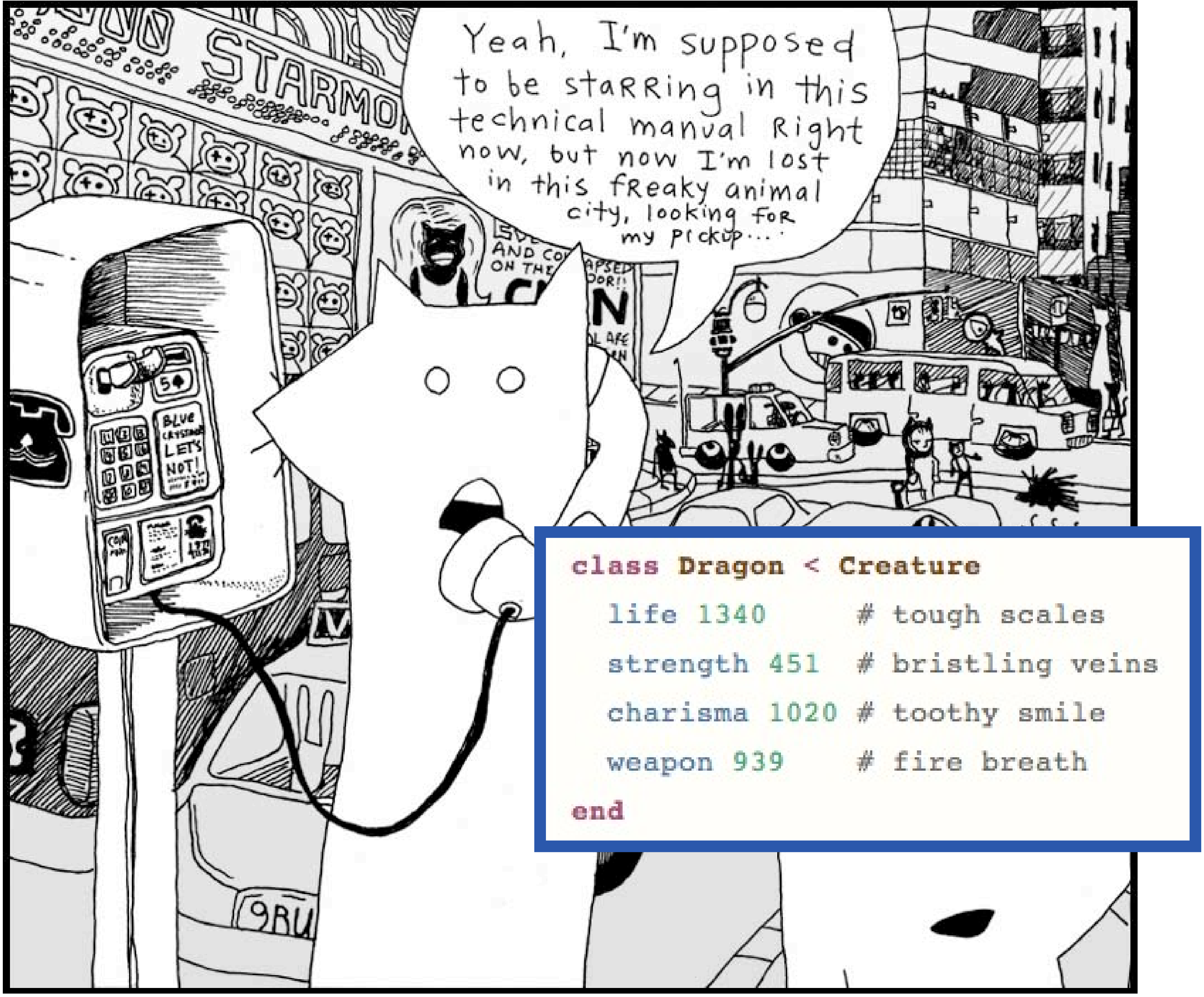
```
class Person
end

Person.new.class # => Person
Person.class # => Class
Person.class.class # => Class
Person.methods # => ["to_s", "nil?", "is_a?", ...]
```

Reflection

2. Modifying these objects modifies the program.

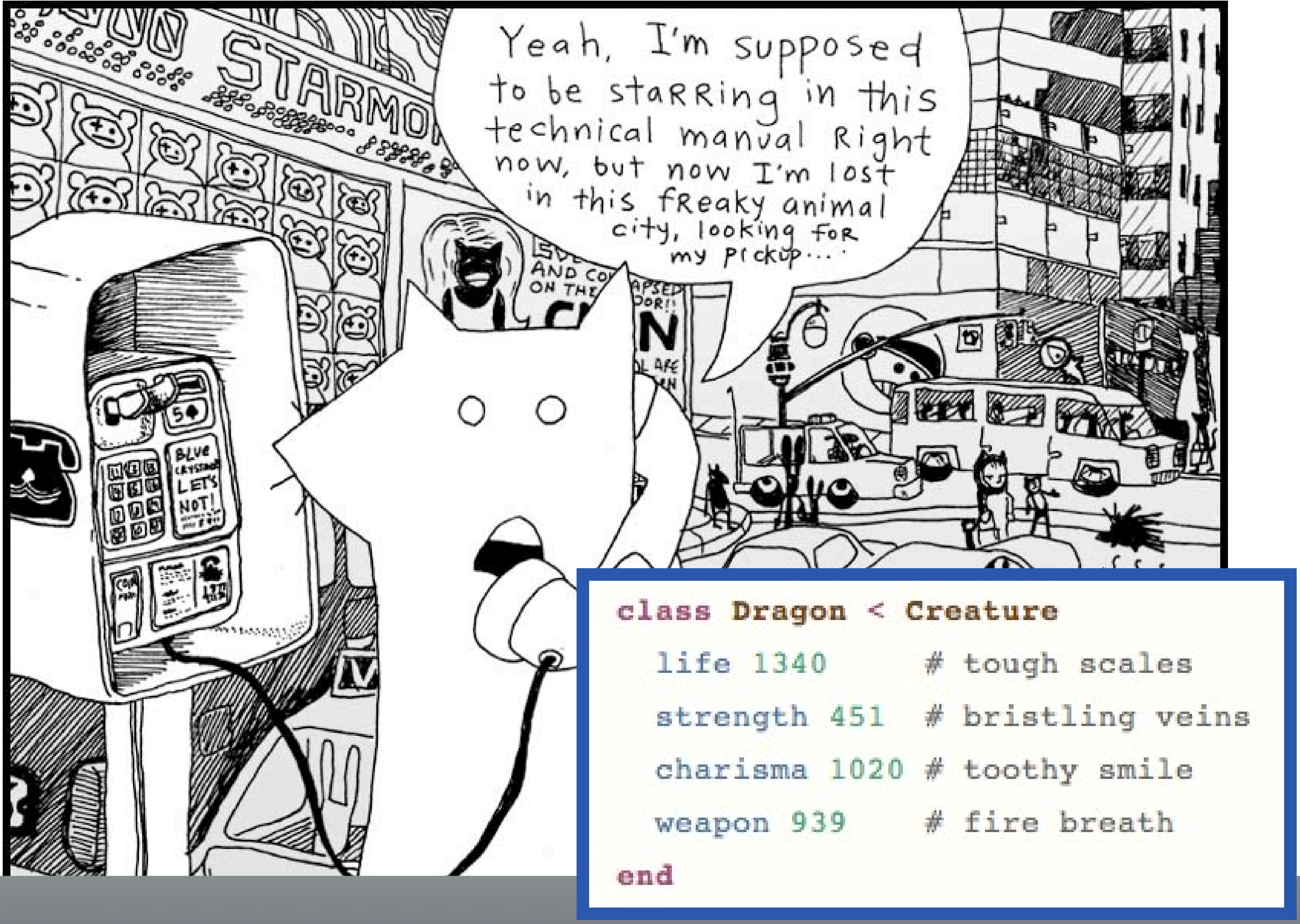
```ruby
class Person
  def self.simple_const(name)
    self.send :define_method, name.to_sym do
      name
    end
  end
end

Person.simple_const "foo"
Person.instance_methods.include?("foo") # => true
Person.new.foo # => "foo"
Person.foo     # =>
# ~> undefined method `foo' for Person:Class (NoMethodError)
```

How many people have read Why’s Poignant Guide to Ruby?


Yeah, I'm supposed to be starring in this technical manual right now, but now I'm lost in this freaky animal city, looking for my pickup...
STARMO
5¢
BLUE CRYSTALS LET'S NOT!
9RU

```
class Dragon < Creature
  life 1340      # tough scales
  strength 451   # bristling veins
  charisma 1020  # toothy smile
  weapon 939     # fire breath
end
```

How many people have read Chapter 6 which talks about Meta-Programming?
How many people wrote out the code for Dwemthy's Array, with the monster classes, and the bomb, sword and the less-than-dangerous lettuce as weapons?
How many people continued on and modified the app cause it was so much fun?


Yeah, I'm supposed to be starring in this technical manual right now, but now I'm lost in this freaky animal city, looking for my pickup...
STARMO
BLUE CRYSTALS LET'S NOT!
5¢

```
class Dragon < Creature
  life 1340      # tough scales
  strength 451   # bristling veins
  charisma 1020 # toothy smile
  weapon 939     # fire breath
end
```

chapter 6

How many people have read Chapter 6 which talks about Meta-Programming?
How many people wrote out the code for Dwemthy's Array, with the monster classes, and the bomb, sword and the less-than-dangerous lettuce as weapons?
How many people continued on and modified the app cause it was so much fun?

Wacky definition

“Program that manipulates the real world”

Programming

“Program that manipulates other programs”

Meta-programming

“Program that manipulates other meta-programs”

Meta-meta-programming

What I want to talk about is actually meta-meta-programming.The breadth of scope of what this statement can encapsulate is well beyond my attention span. For some people it means adding hooks or wrappers around methods. For others it means the ability to create or extend language tools that you use to do normal meta-programming.

But there is no line to be drawn in the sand about what part of the Ruby API is meta-programming and what is meta-meta-programming. I give the word to you instead to open up your mind to how far your can fart-arse about when you are supposed to be doing serious work.

Human learning

Learn new things.

Programming

Learn how to learn new things.

Meta-programming

Learn how to learn how to learn new things.

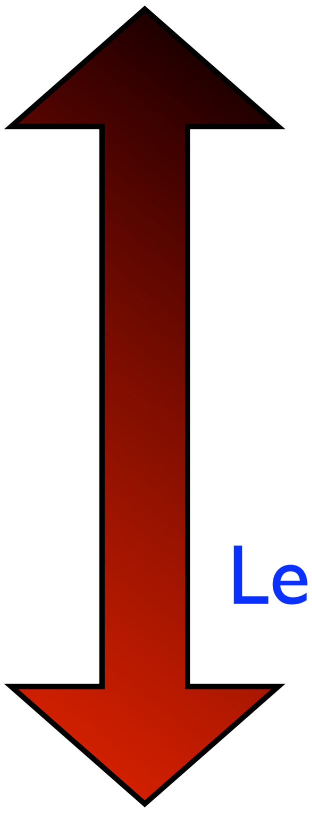
Meta-meta-programming

Consider a human. We know how to learn new things. Running that program is when you “do” something.

But in order to do that, we’re already programmed how to learn new things. That is, we learned how to learn new things.

But how did that happen? Can we change it? Fortunately, humans can learn how to change the way they learn. That is, they can learn how to learn how to learn new things.

Human learning



Learn new things.

Programming

Learn how to learn new things.

Meta-programming

Learn how to learn how to learn new things.

Meta-meta-programming

Learn how to learn how to learn how to learn how to...

Bachelor of Arts/Humanities degree

But becareful of following this to infinity. You'll spend a lot of time thing thinking and never get any real work done.

Arguably, productivity is at the top of this list, and wasting your parents money at university is at the bottom of the list.

You *can't* teach an old *dog* new tricks

THE SATURDAY EVENING POST
JULY 11, 1931
Stewart Edward White—Dwight Mitchell Wiley—Anne Cameron
W. Thornton Martin—Norman Reilly Raine—Eve Garrette Grady

http://www.curtispublishing.com/images/NonRockwell/9310711.jpg

But, you *can* teach an old *man* new tricks

Some of us got married, and then we learnt all sorts of new things. Like mowing the lawn regularly.
http://pro.corbis.com/images/42-15509965.jpg?size=572&uid=%7B89D13686-7574-4513-AA26-819FEC5A7BC1%7D

You *can* teach an old *Ruby program* new tricks

Ruby gives your programs the capacity not just to learn how to learn how to be programmed.

Examples of meta-meta-programming

- Redefine `define_method` as `reverse_method`
- Create a new class syntax
- Generate new generators

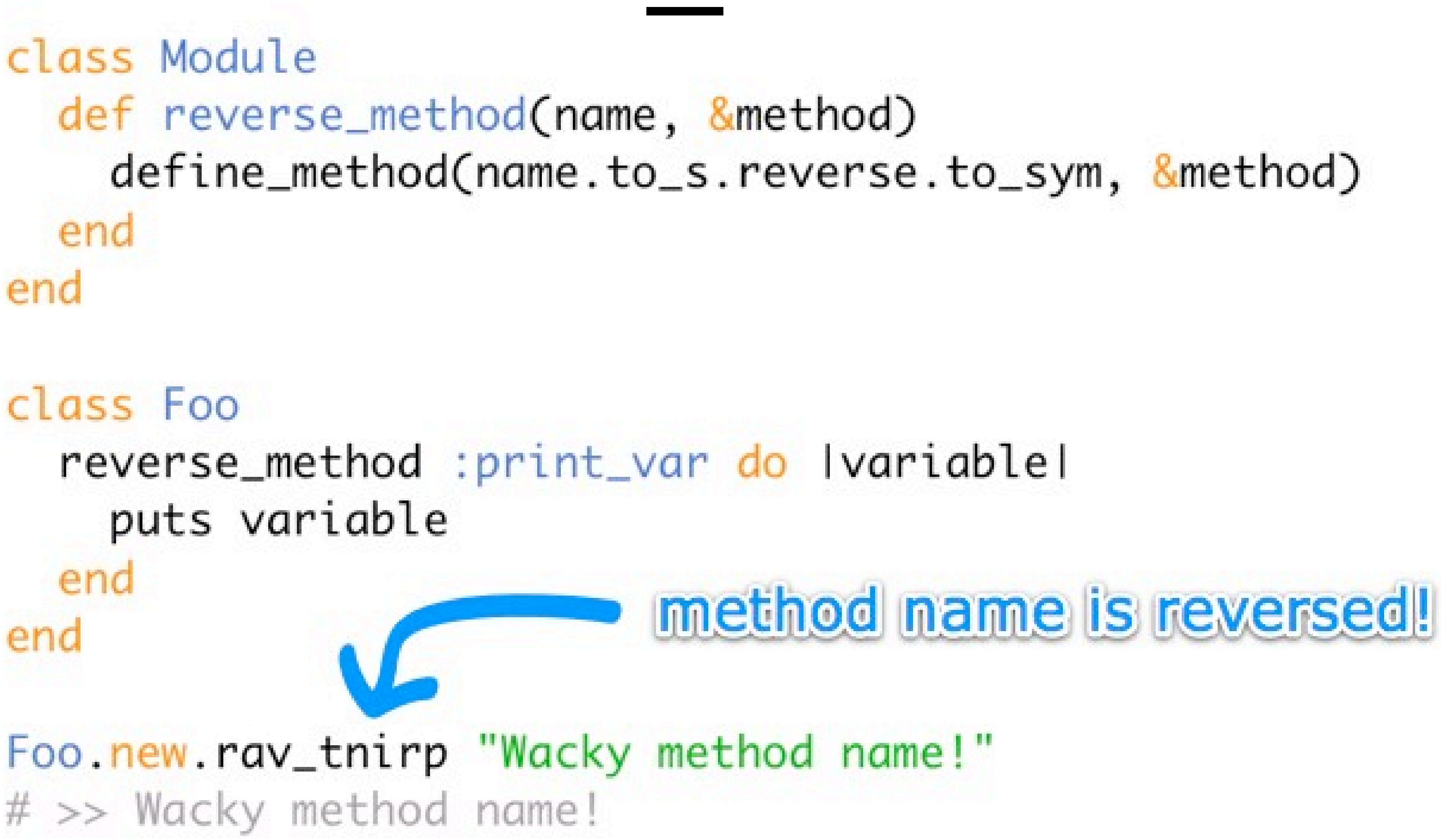
Let’s explore a couple examples of meta-meta-programming to help break down some barriers of what you think you can or cannot do with Ruby.

define_method

```
class Foo
  define_method :print_var do |variable|
    puts variable
  end
end

Foo.new.print_var "Normal method"
# >> Normal method
```

Ruby has a meta-programming helper method ‘define_method’ to allow you to create new methods.

reverse_method



```ruby
class Module
  def reverse_method(name, &method)
    define_method(name.to_s.reverse.to_sym, &method)
  end
end

class Foo
  reverse_method :print_var do |variable|
    puts variable
  end
end

Foo.new.rav_tnirp "Wacky method name!"
# >> Wacky method name!
```

Let’s teach Ruby about a new way to create methods – with the method name in reverse.
Since we manipulated our program to give it a new meta-programming helper, then this is meta-meta-programming.

Create new class syntax

```ruby
# http://blog.jayfields.com/2008/01/write-only-ruby.html
C "Foo" do
  a :first_name, :last_name, :favorite_color
  i Enumerable

  init { |*args| s.first_name, s.last_name = *args }

  d.full_name do
    "#{first_name} #{last_name}"
  end
end

person = Foo.new("Jay", "Fields")
puts person.full_name
# >> Jay Fields
```

Here’s an idea I first saw from Jay Fields. Instead of the unnecessary verbose Ruby syntax for creating classes and methods, we replace each with a shorter word. Class = C, accessor = a, constructor = init, define_method = d

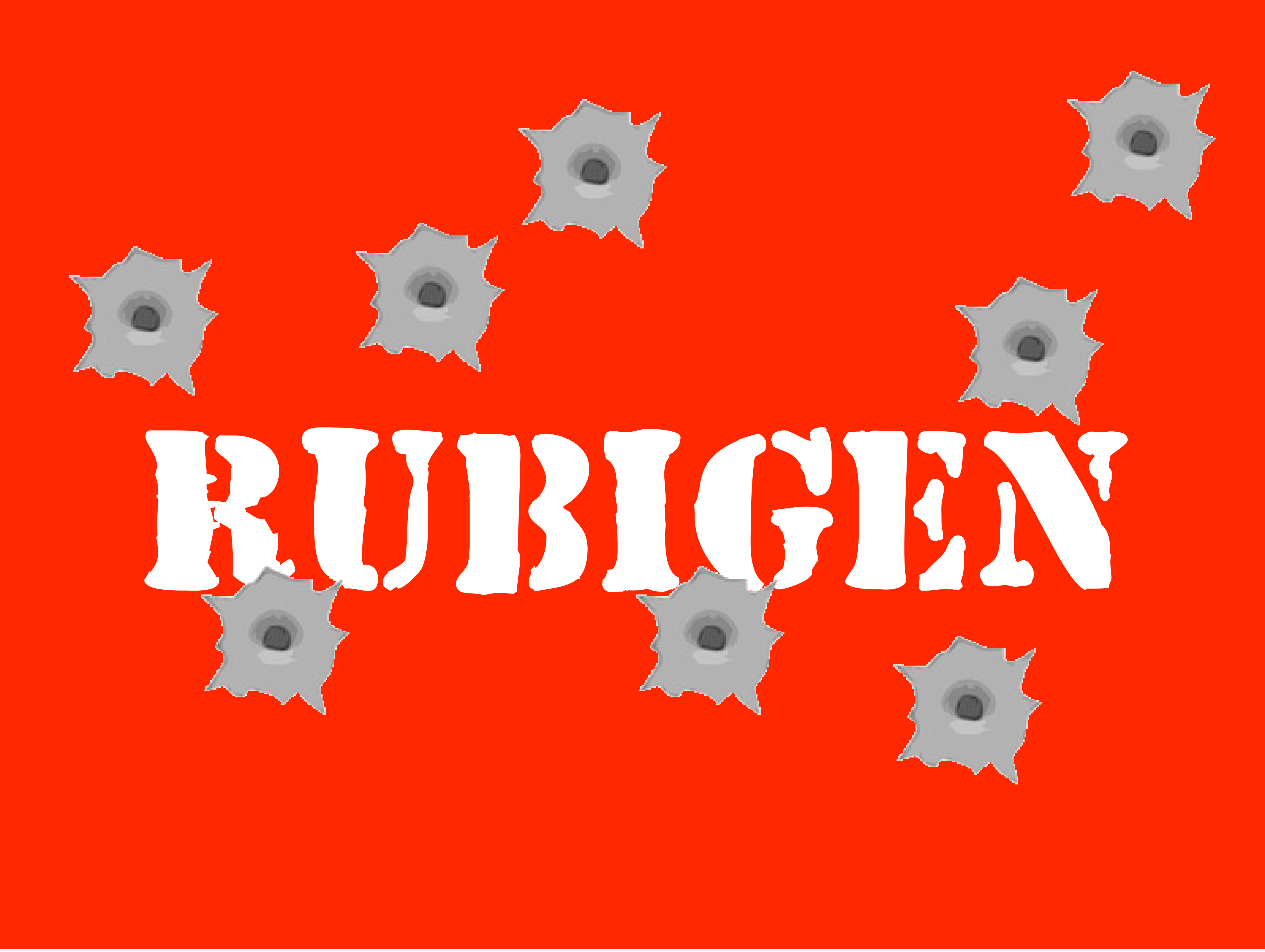
This is meta-meta-programming – you are teaching Ruby how you want to do your meta-programming.

You are limited by the Ruby parser of course. You cannot change the Ruby syntax with Ruby MRI. But perhaps with Rubinius our power to reprogram Ruby will be limitless?

Generate new generators

```
meta_gem$ ruby script/generate component_generator onepageapp merb
      create  merb_generators/onepageapp/templates
      create  merb_generators/onepageapp/onepageapp_generator.rb
      create  test/test_ onepageapp_generator.rb
      create  test/test_generator_helper.rb
      create  merb_generators/onepageapp/USAGE
```

If a code generator is an example of meta-programming, then creating a new code generator is meta-meta-programming.



RUBIGEN

There is a project called RubiGen which allows you to use generators like you have in Rails in other frameworks like Merb.

How many generator generators?

```
meta_gem$ ruby script/generate
...
Installed Generators
  Rubygems:
      application_generator,
      component_generator, ...
```

When you are writing a RubyGem for a project, there are two generators that you can use to create new generators. application_generator is used for creating the scaffolding for an entirely new application, like the “rails” command. component_generator creates a generator that is used within a framework application, like the “model” or “scaffold” generators within Rails.

If you are creating new frameworks, you can use these two generators to give your new users scaffolding generators to help them learn about your framework. They save you having to create generators from scratch.

Using macros...

```
## Our comment model
class Comment < ActiveRecord::Base
  belongs_to :commentable, :polymorphic => true
end

## A model using Comment
class Timesheet < ActiveRecord::Base
  has_many :comments, :as => :commentable
end

## And another
class ExpenseReport < ActiveRecord::Base
  has_many :comments, :as => :commentable
end
```

Example from "The Rails Way"

The belongs_to and has_many macros use meta-programming to work. When executed they create new methods on the class.

They look like ordinary language constructs – which is a beautiful by-product of its syntax flexibility. Parentheses are optional.

But they are method calls upon the class.

...reflectively!

```
module Commentable
  def self.included(base)
    base.class_eval do
      has_many :comments, :as => :commentable
    end
  end
end

## A model using Comment
class Timesheet < ActiveRecord::Base
  include Commentable
end

## And another
class ExpenseReport < ActiveRecord::Base
  include Commentable
end
```

Example from “The Rails Way”

Here we’re invoking the meta-programming filled macro “has_many” programmatically via the #include call.
Its arguably meta-meta-programming, but its something we do all the time in plugins/libraries.

Point to make - the meta-meta-code in self.included is ugly; but the application code is concise and meaningful. You document somewhere that “include Commentable” gives your model a #comments association and you’re done. If you use this in all your applications - say via a plugin - it just becomes another language construct.

How it works:
When you include a module into a class, a couple things happen.

First, all that modules methods are added to the class. In this example there aren’t any, but this is a common use for modules to “mix-in” methods, esp since Ruby has no multiple inheritence.

Secondly, and importantly here, that module’s included method is involved, and passed the target class.

This means the module can perform meta-programming on the class.

In this example, we execute a block of code upon the class using the class_eval call. In this example, each class that includes the Commentable module has a comments association added to it.

DIY meta-meta-programming

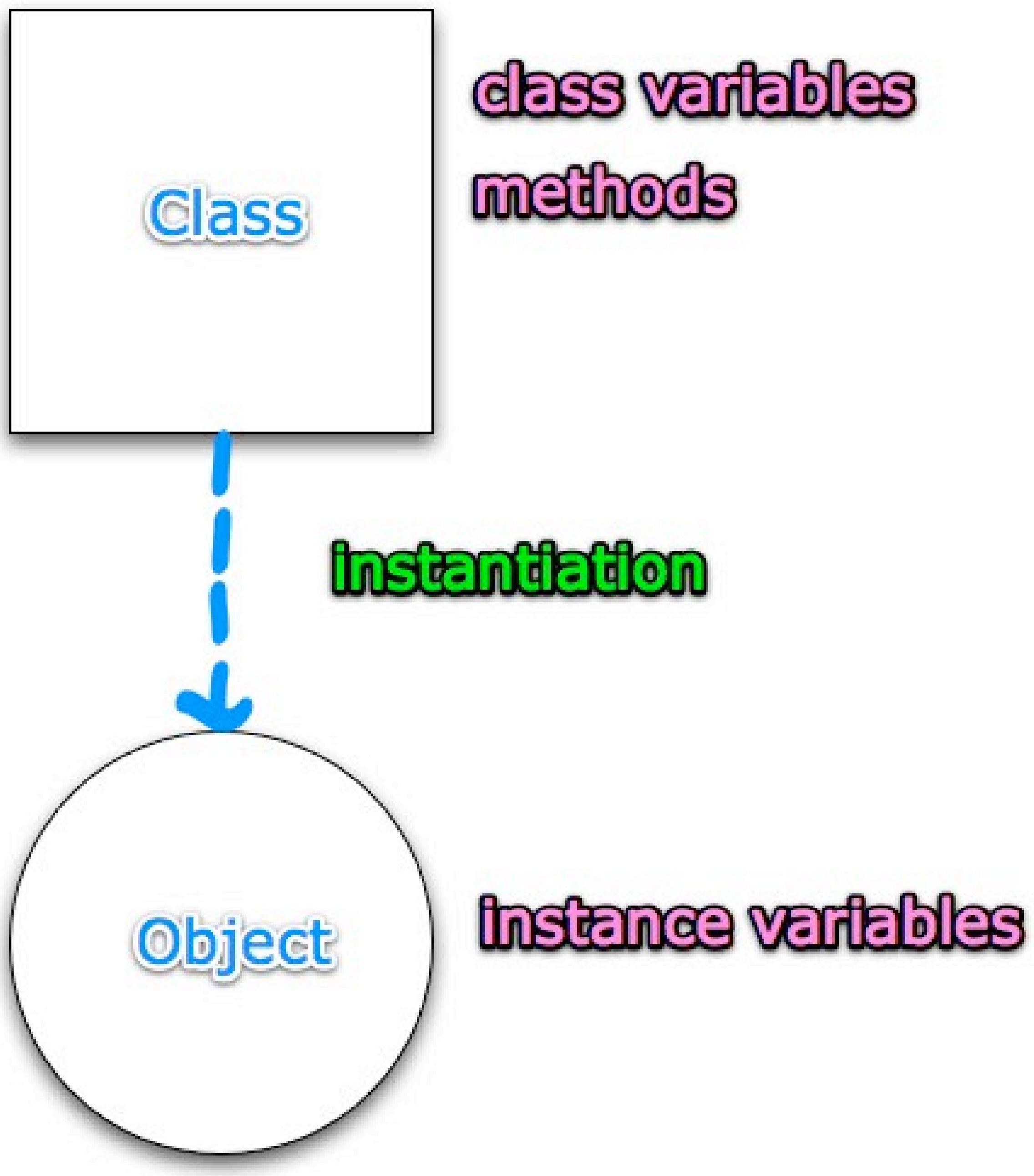
- Write new generator generators
- Write TextMate snippet generator
- Write Ruby “macros”

The reason to show you all this is to explode your mind with ideas. Write new generators or new generator generators. For example, Merb could have its own generator generator since its generators look different from generic RubiGen generators or Rails generators.

Learn to write TextMate snippets, or perhaps figure out how to generate or maintain snippet files manually.

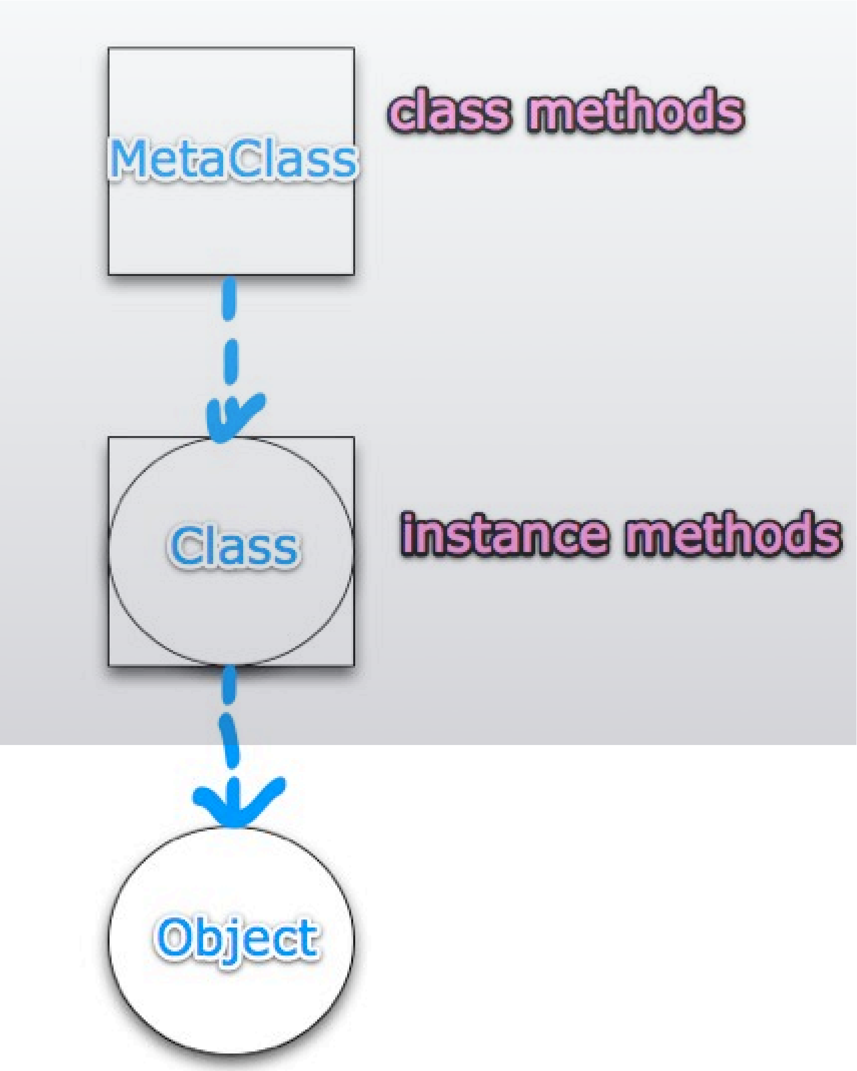
Learn how to write

How to think like how Ruby thinks...



Let’s use circles for objects, and squares for classes. The blue arrow shows an instantiation: an object belonging to a class. Objects hold instance variables, and Classes hold methods. Objects don’t hold methods.

MetaClass concept



But this is Ruby, and we know that Classes are Objects. So its a combo of a square and a circle.

But if a class is an object, then it must have a class of its own. Let’s call this the metaclass. If a class contains instance methods for an object, then the metaclass contains the class methods for the class.

This is not exactly how Ruby implements classes and metaclasses, but its a very clean example of how it could be.

Classes have Classes

```ruby
class Person; end

person = Person.new
person.class  # => Person
Person.class  # => Class
Class.class   # => Class
```

Let’s look at some Ruby code and see if we can map it across to our diagram.

The person object has a class Person, and it seems the Person class has a class called Class. Finally, there’s some circularity: the class of Class is itself.

So, if the instantiating class of Person is a thing called Class, then we’d expect to find class methods there.

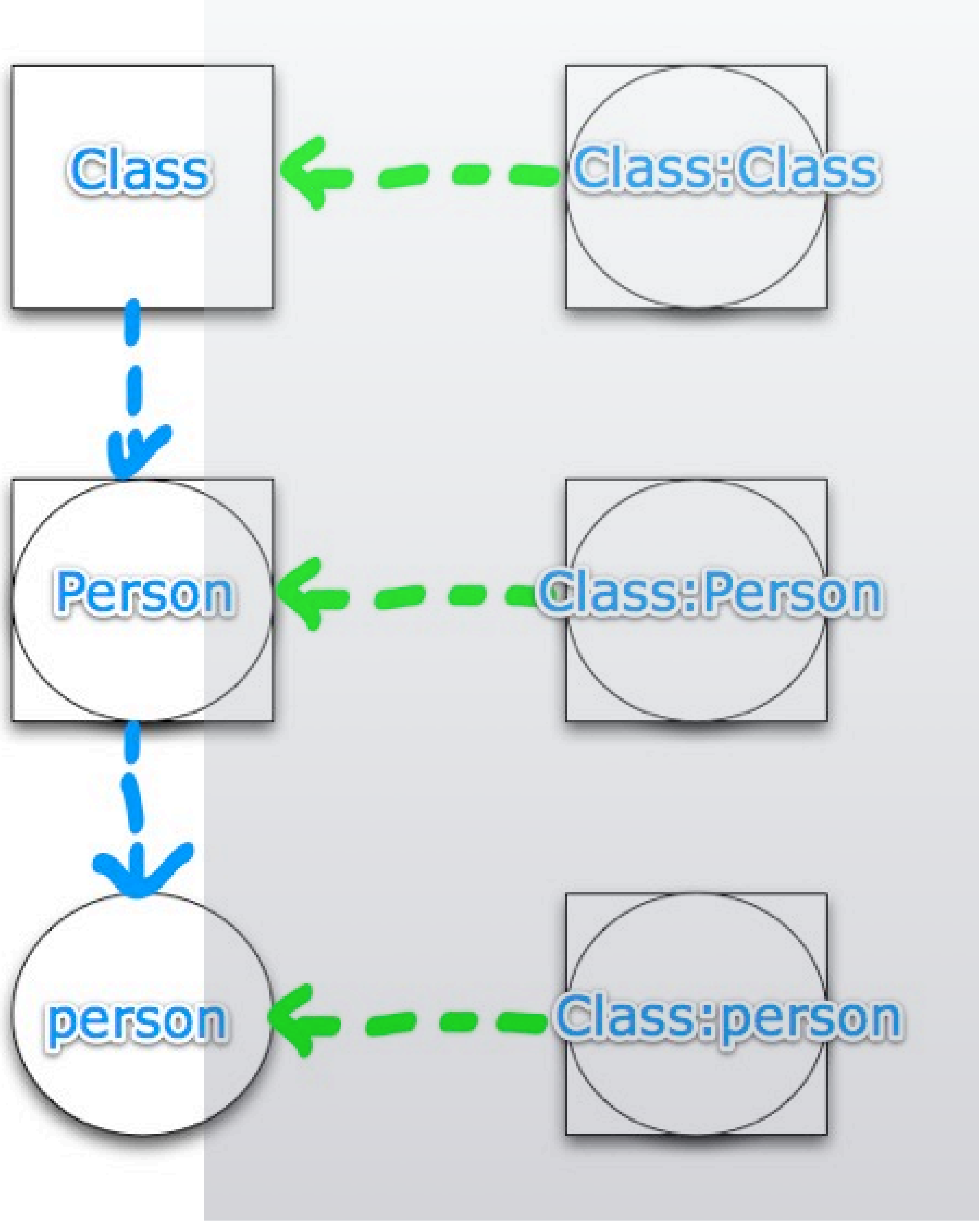
Where are ‘class methods’?

```
class Person
  def self.my_class_method; end
end

Class.instance_methods.grep(/my_class_method/)  # => []
Class.methods.grep(/my_class_method/)           # => []

Person.methods.grep(/my_class_method/)
# => ["my_class_method"]
Person.instance_methods.grep(/my_class_method/) # => []
Person.metaclass.instance_methods.grep(/my_class_method/)
# => ["my_class_method"]

Person.class == Person.metaclass  # => false
```

At the top, we define a class or static method on the Person class. But when we investigate the methods on Class, it is nowhere to be found?!

Yet, it obviously does exist as a method on the Person class, though its a class method not an instance method. So, where is it defined if its not in the class called Class?

There is actually something else, something we’ll again call the metaclass. But this metaclass, whatever it is, is different from the Person class’s own class.



Let’s replace the words Class and MetaClass on our old diagram with actual Ruby class names: Person and Class, and we’ll assume that an instance of the Person class is in a variable called person.

On the RHS we’re showing something new: the metaclass object introduced a moment ago. As you can see, every object in Ruby world can have its own metaclass, also called a singleton or eigenclass.

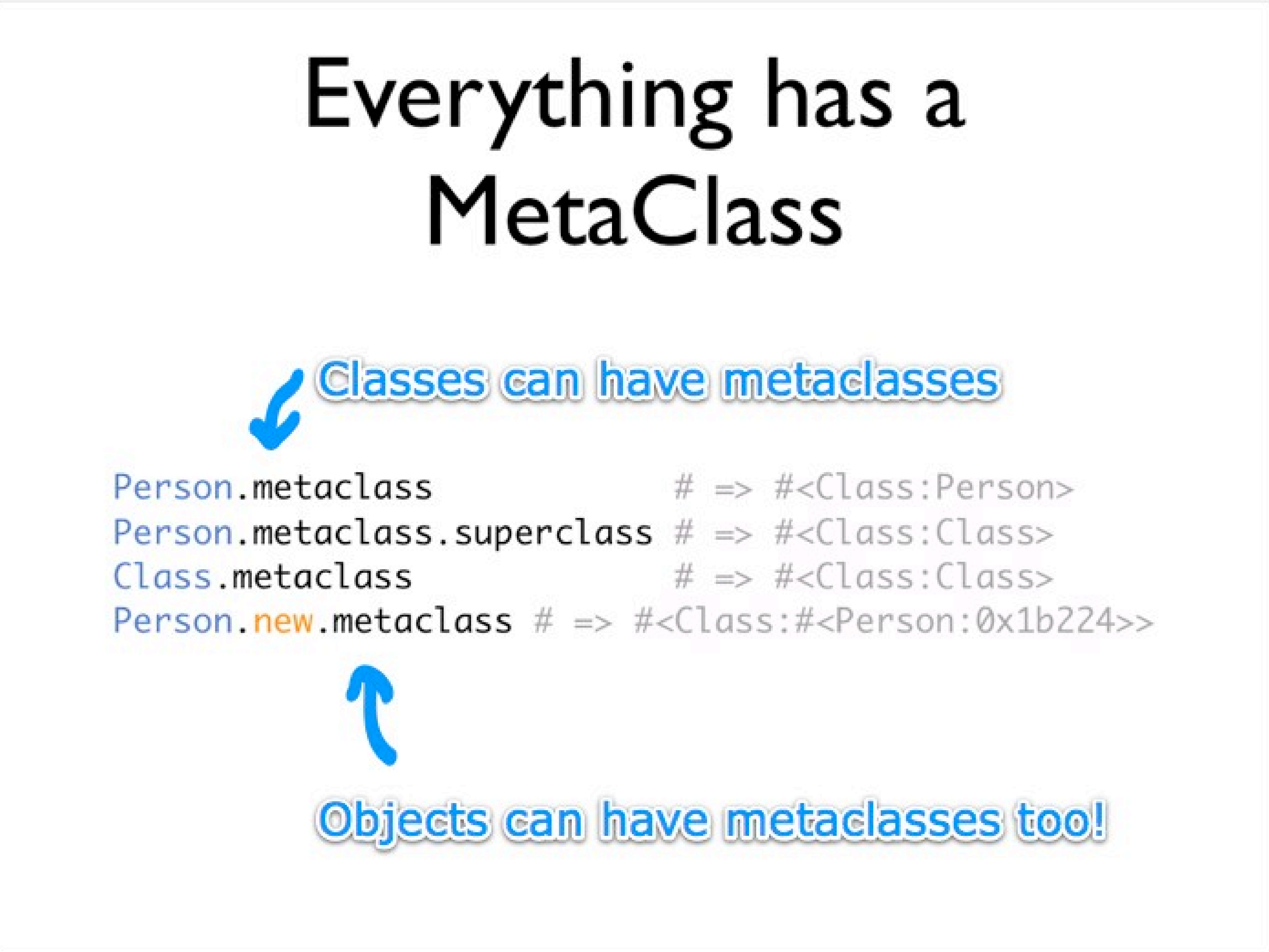
metaclass accessor

```
class Object
  def metaclass
    class << self
      self
    end
  end
end

# found in gem 'metaid' by _why
```

Now, the metaclass method doesn’t exist in standard Ruby. So, here’s its implementation; and its packaged within a RubyGem called ‘metaid’ by whytheluckystiff.

You can see its implementation is to use the class-arrow-arrow-self syntax you’ve probably seen before or even used.

In fact, the single purpose of this Ruby syntax is to expose an object’s metaclass and let you do stuff to it. Here, we’re just returning it as a result of the method call. But we could define methods on it, invoke methods on it, etc.



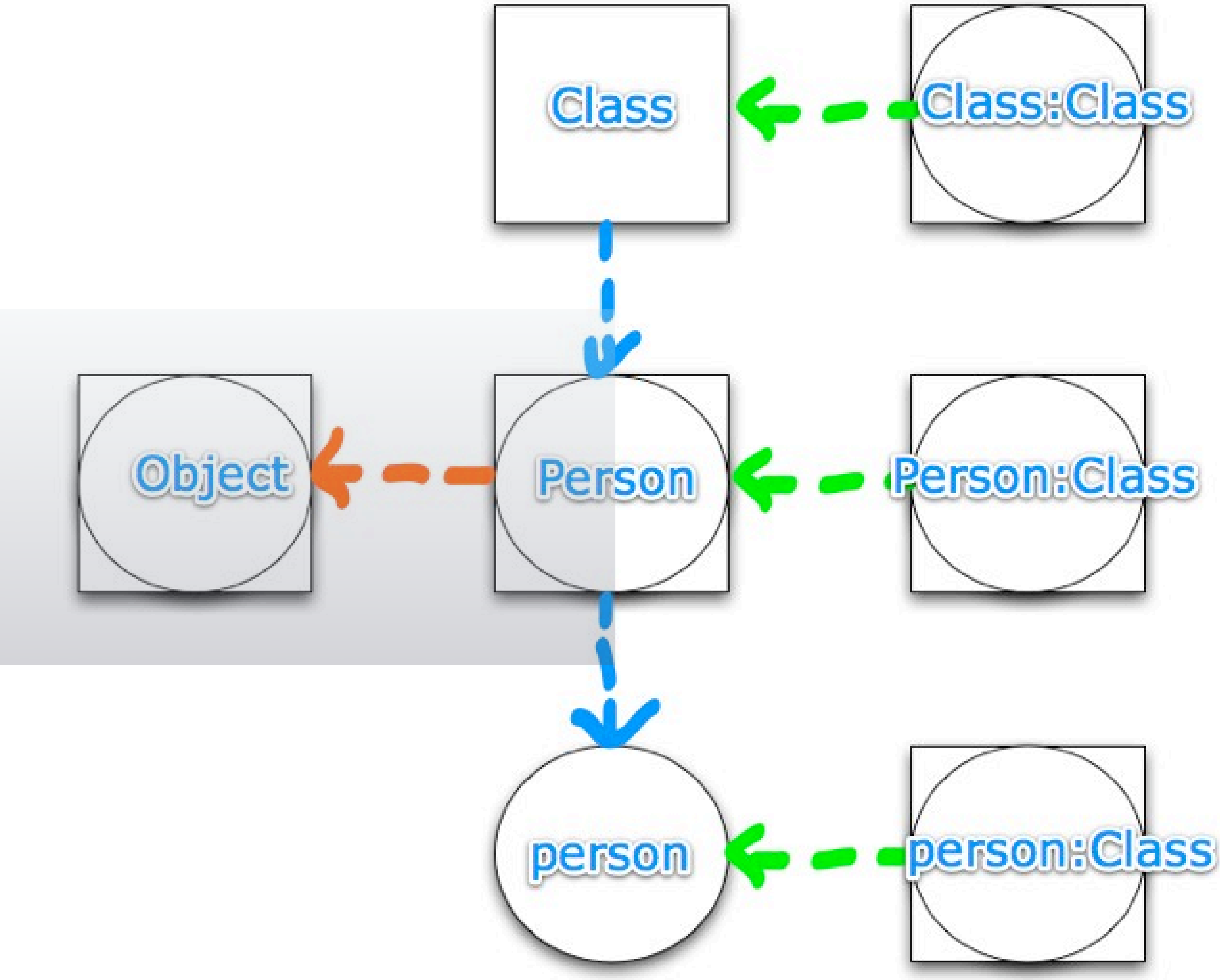
Everything has a MetaClass

Classes can have metaclasses

```
Person.metaclass                  # => #<Class:Person>
Person.metaclass.superclass # => #<Class:Class>
Class.metaclass                   # => #<Class:Class>
Person.new.metaclass # => #<Class:#<Person:0x1b224>>
```

Objects can have metaclasses too!

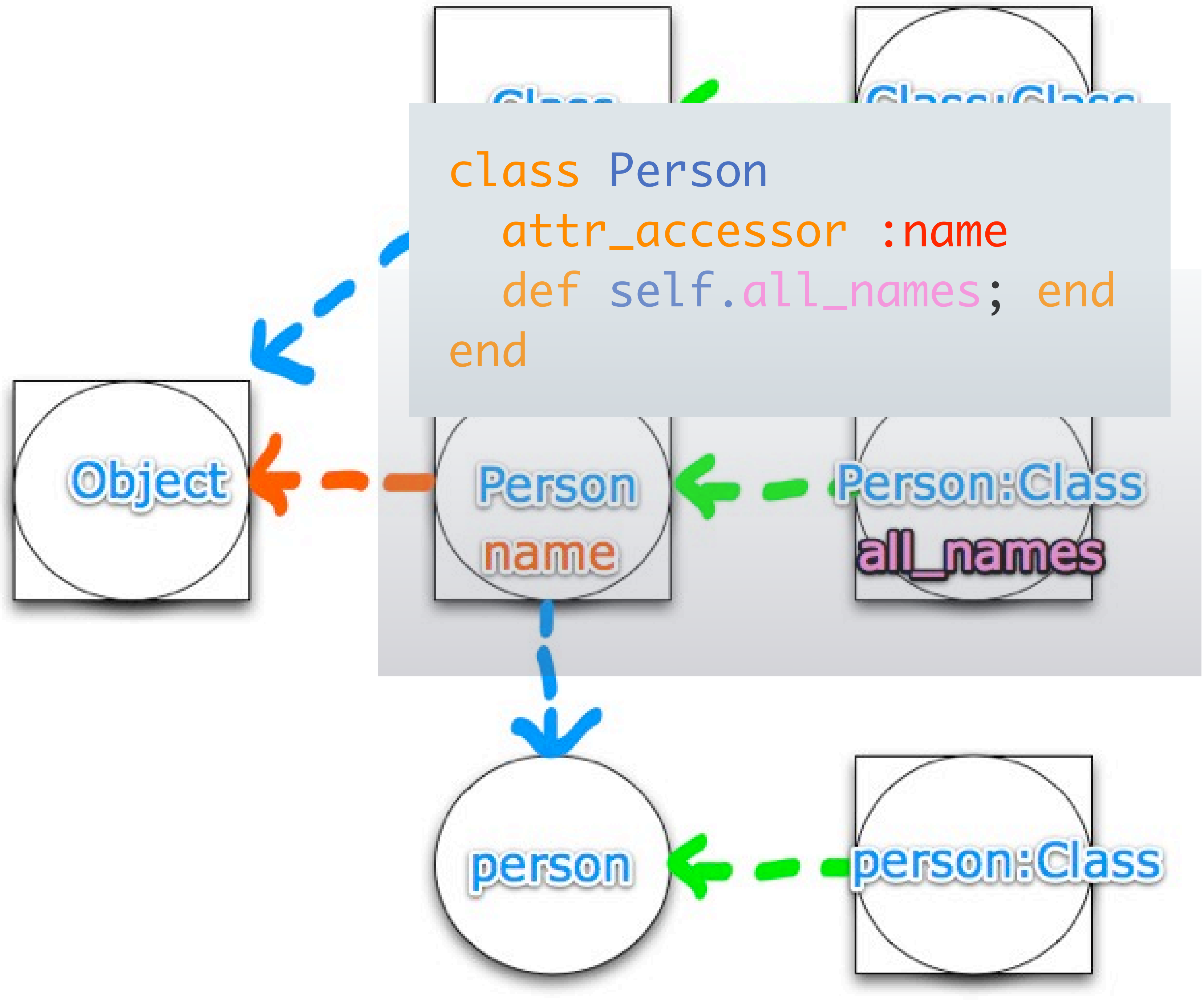
If we print out our metaclasses, we see they have unique names, based on their underlying object.



So let's build up the complete picture of classes and metaclasses and their relationships.

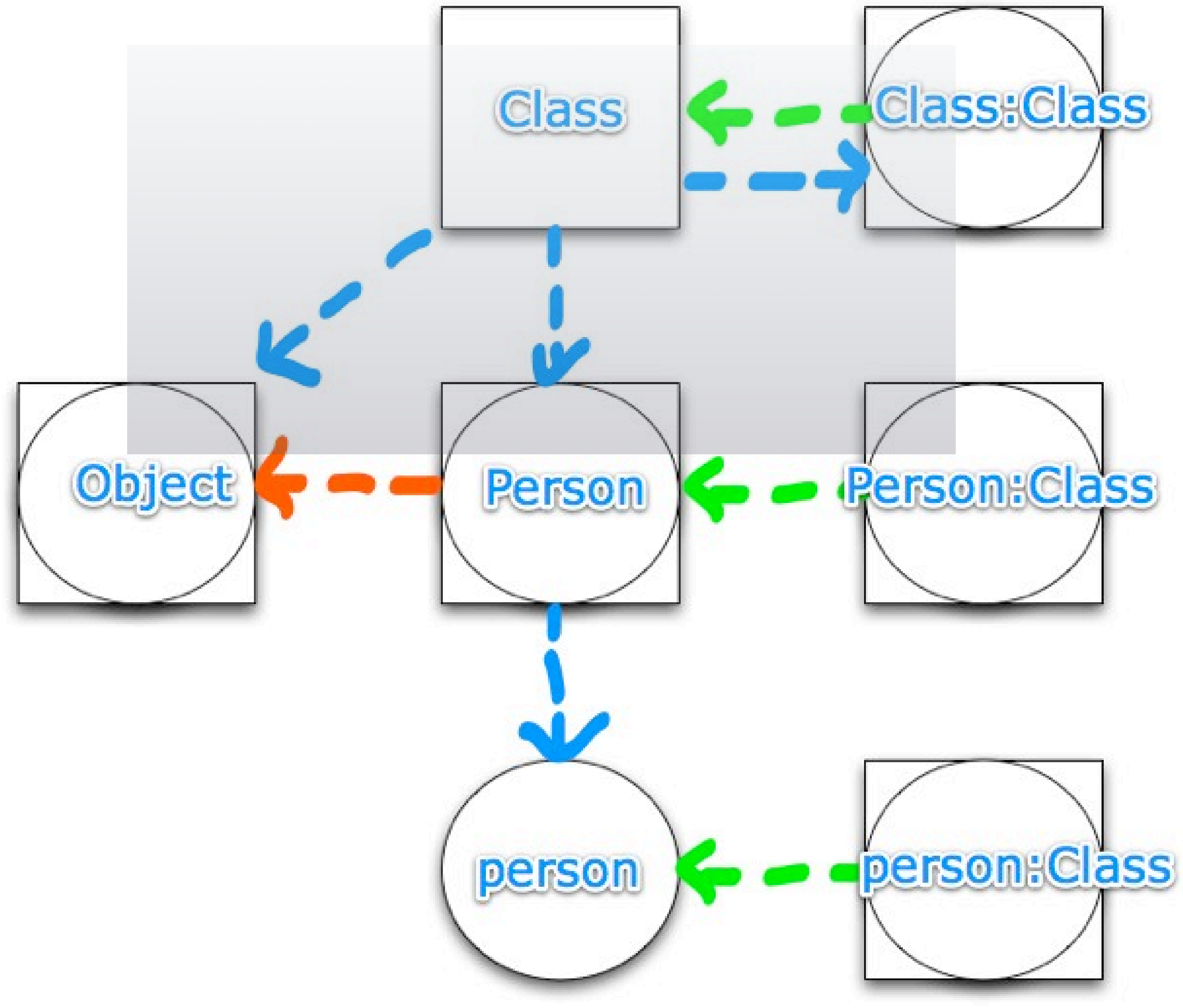
First, let's add in the root Object class that is the superclass of nearly every class. Here, the red arrow shows a subclass relationship: Person subclasses Object. Object itself has no superclass, so there are no red arrows coming from it.

But in our picture we have 4 other classes: Class and the 3 metaclasses.

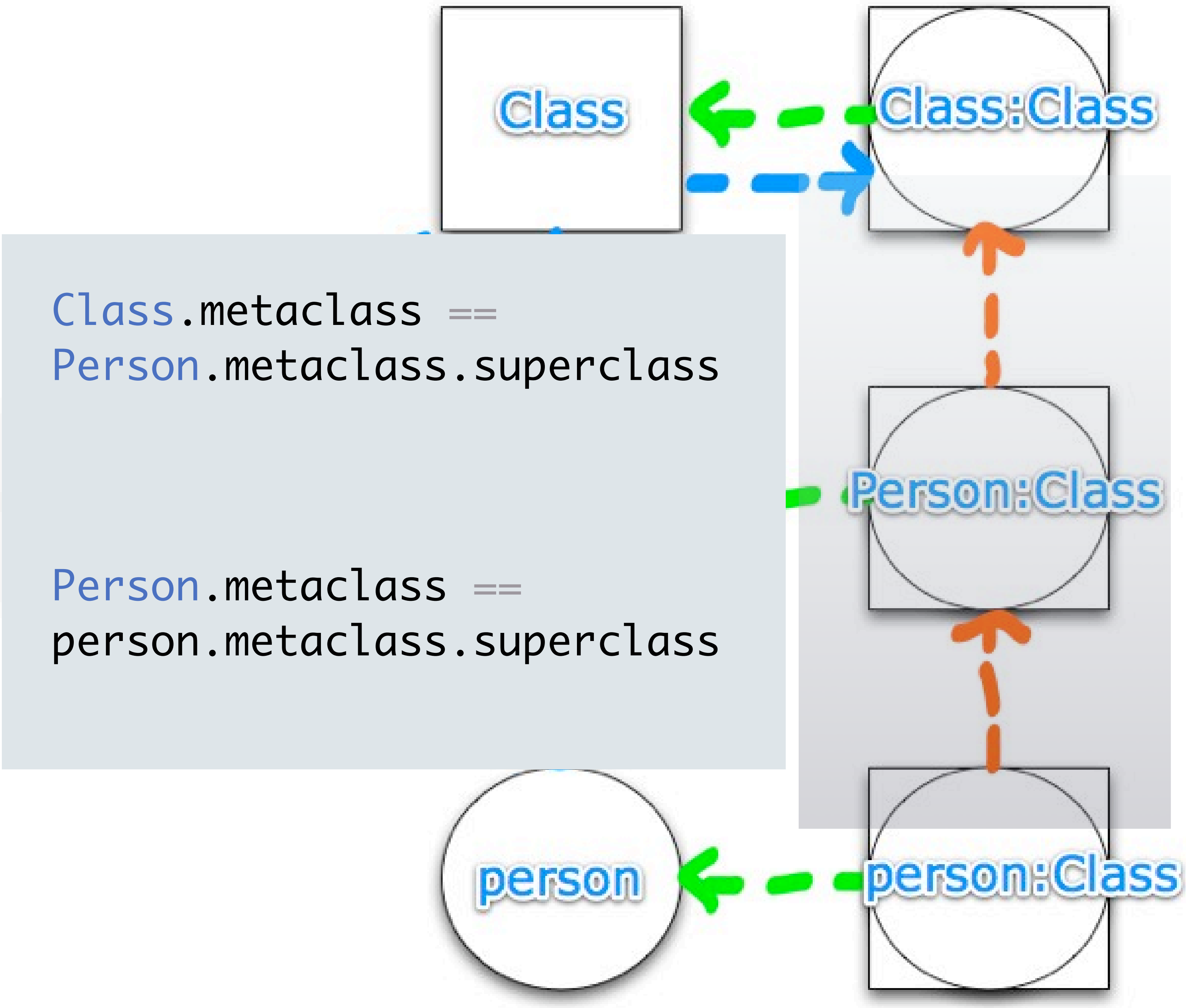


```ruby
class Person
  attr_accessor :name
  def self.all_names; end
end
```

When we add methods and class methods to our Person class we can now see where they are stored. It is the Person's metaclass that stores the class method, not the Class class.



Here I've added in some more blue arrows which shows that all classes are instances of Class. I'm missing two blue arrows cause it makes the picture look ugly. But there should be blue arrows from Class to all the other squares.

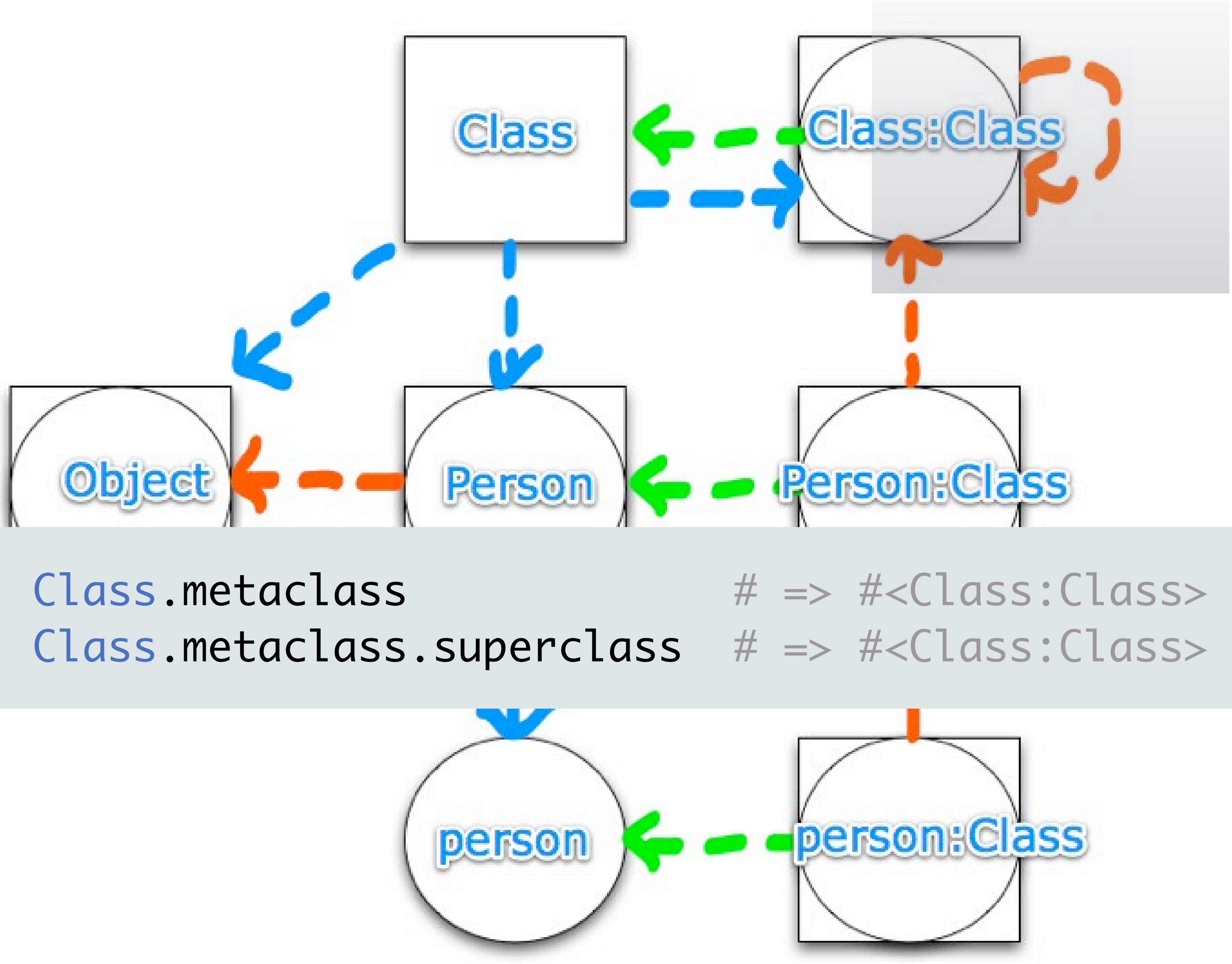


```
Class.metaclass ==
Person.metaclass.superclass


Person.metaclass ==
person.metaclass.superclass
```

Metaclass's superclasses

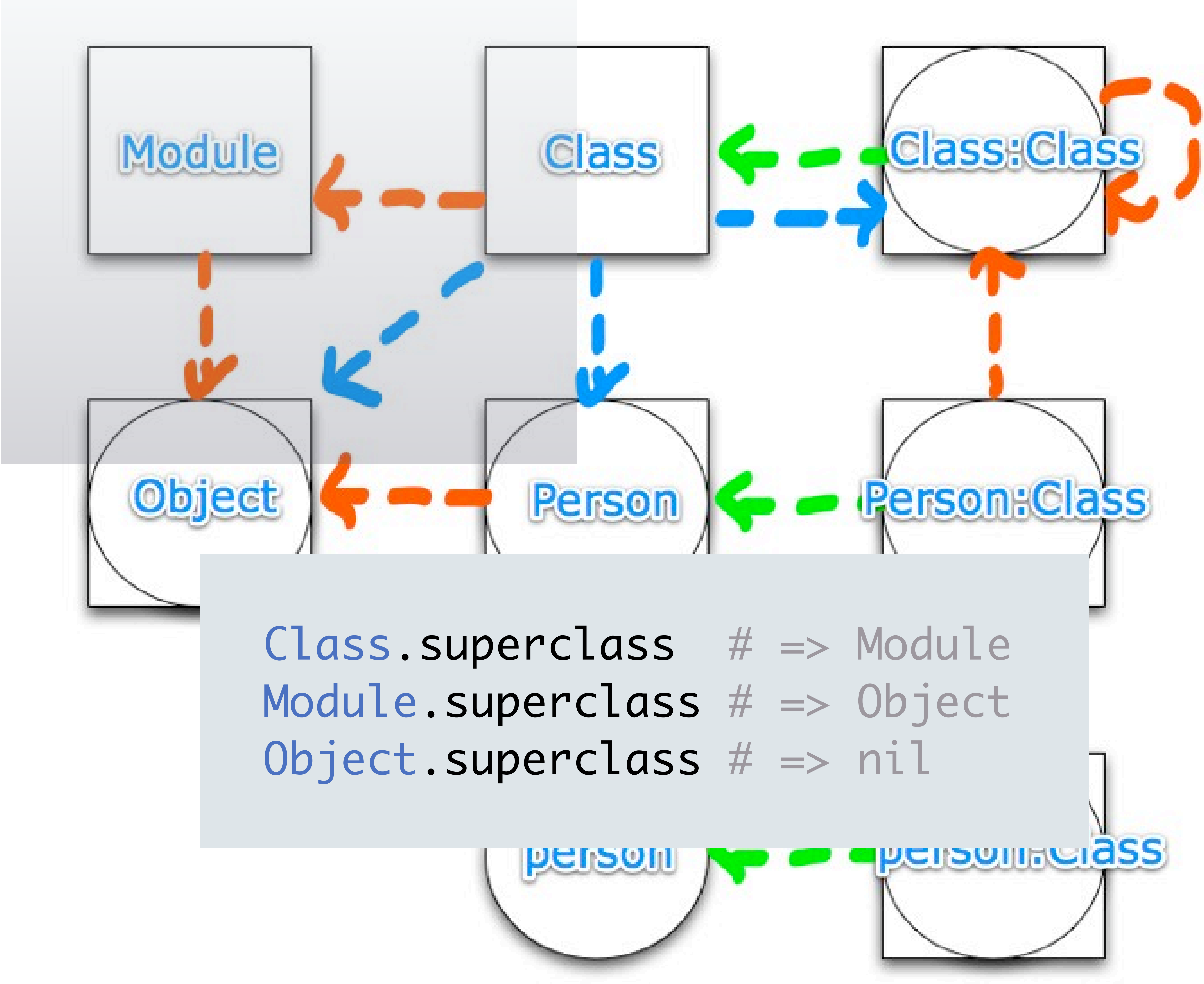
It turns out that whilst a Person object is an instance of the Person class, the metaclasses of the two things are superclasses of each other.

That is, the metaclasses on the right have red arrows between them matching the blue instantiation arrows of their owner objects.



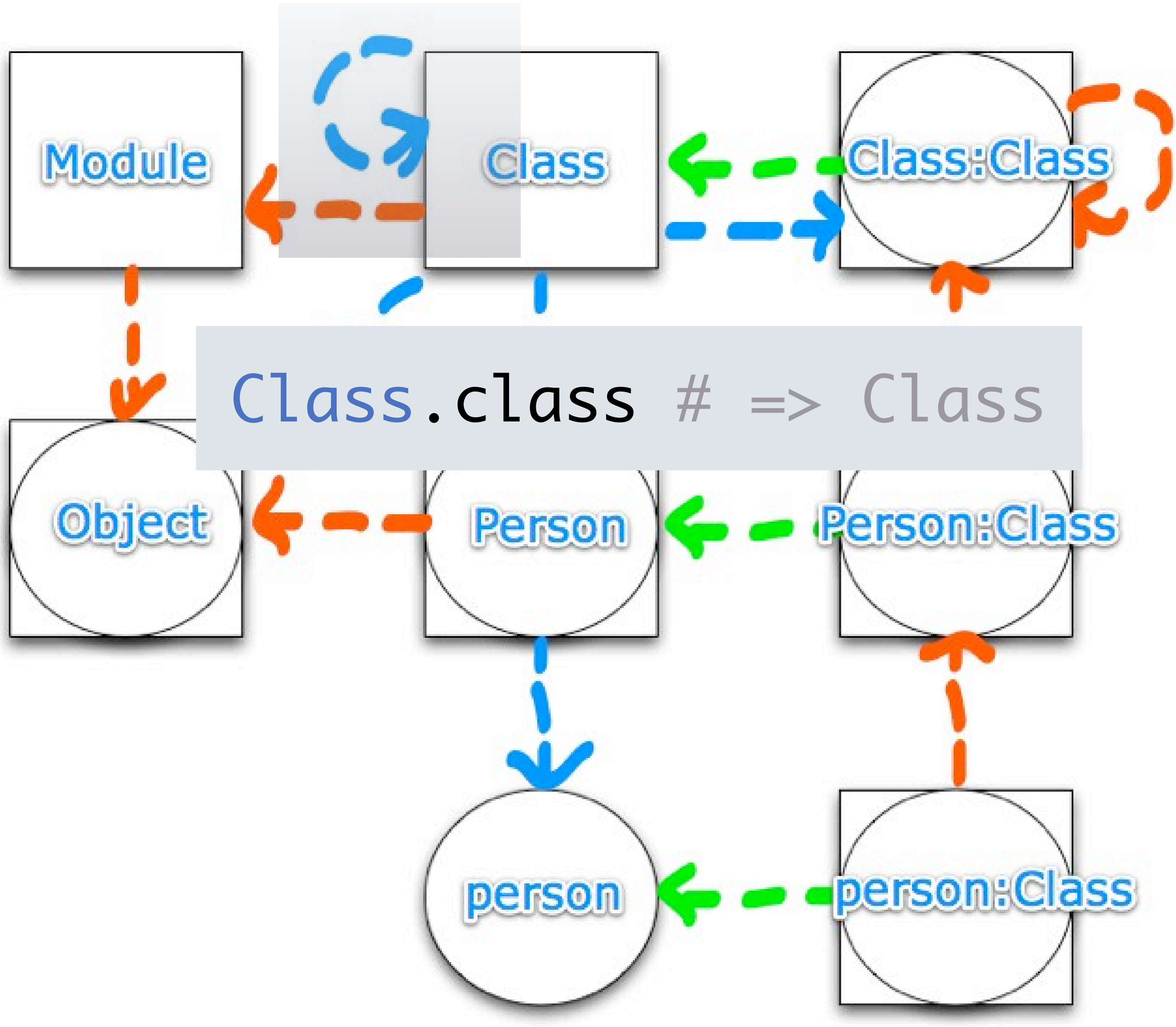
```
Class.metaclass               # => #<Class:Class>
Class.metaclass.superclass    # => #<Class:Class>
```

At the top of the metaclass tree, the Class Class is its own superclass.



```
Class.superclass  # => Module
Module.superclass # => Object
Object.superclass # => nil
```

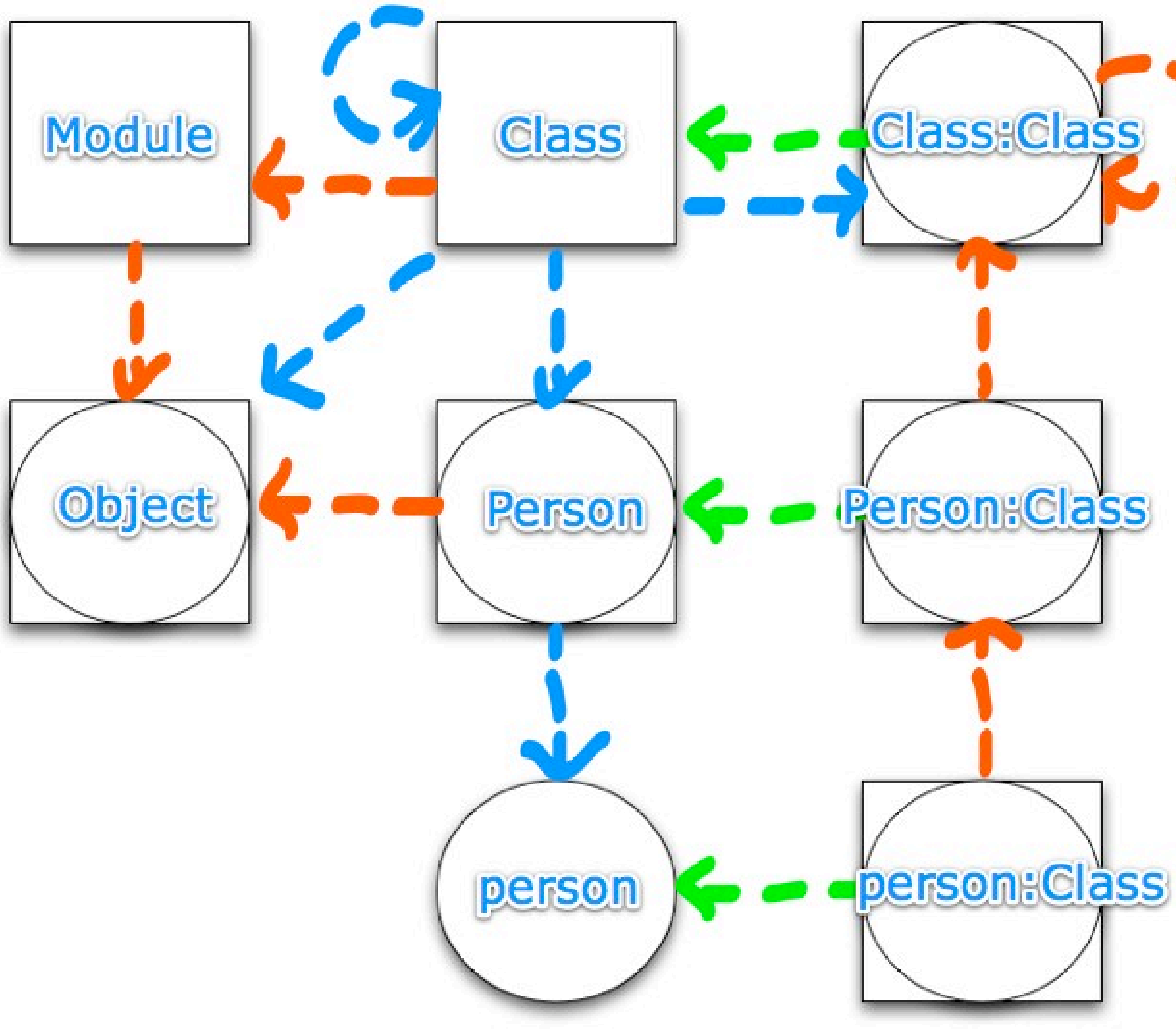
Class superclass
Unlike the metaclasses, the superclass of Class is Module, which superclasses Object.



```
Class.class # => Class
```

And finally, like every other class in Ruby, the class called Class is its own Class, thus giving us more meta-circularity.

And we’re done! Stick that up on your wall at work and star at it like those old Magic Eye pictures. Perhaps if you look long enough you’ll see an Eagle or Where’s Wally.

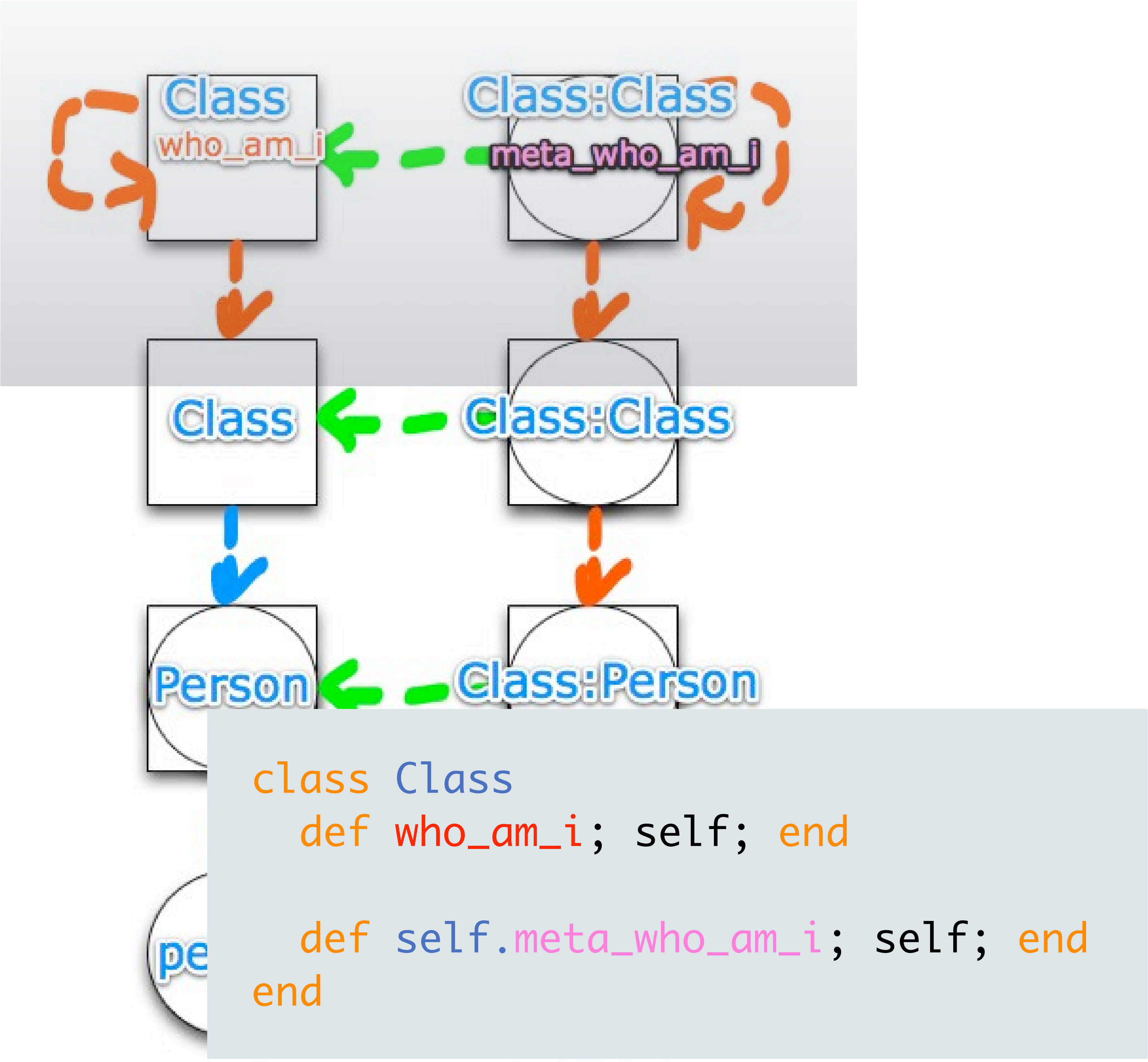


The final diagram

Meta-circularity

```ruby
class Class
  def who_am_i; self; end

  def self.meta_who_am_i; self; end
end

Class.instance_methods.grep(/who_am_i/)
# => ["who_am_i"]
Class.methods.grep(/who_am_i/)
# => ["meta_who_am_i", "who_am_i"]
```

So let’s explore what happens at the top of that picture, where Class is an superclass of itself, and Metaclass is an superclass of itself.



```ruby
class Class
  def who_am_i; self; end

  def self.meta_who_am_i; self; end
end
```

Like the example earlier, here is a class with an instance method and a class method. Except, the class is at the point of meta-circularity - at the top. So the who_am_i instance method is actually stored on Class, because there is nowhere else to store it. But the class method “meta_who_am_i” is still stored on its metaclas.

Meta-meta-programming

It's just for **programmers**, but it's for all **programmers**

DR NIC ACADEMY

http://www.celebbinge.com/wp-content/uploads/2007/08/jessica-simpson.jpg