The Ruby Racer

Under the Hood

Charles Lowell
@cowboyd

THE FrontSide

The Ruby Racer

Embeds the (ridiculous) V8 JavaScript interpreter into your Ruby process

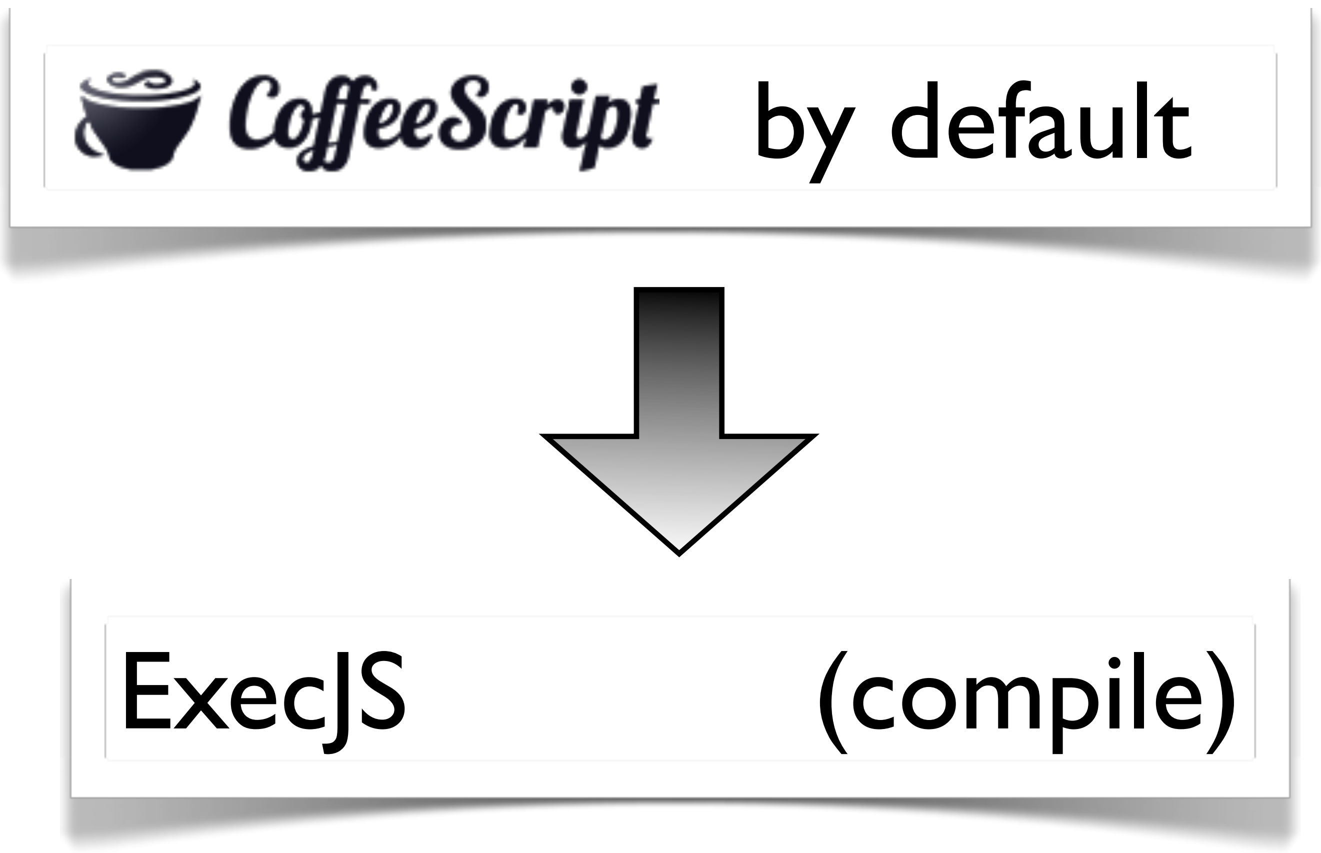
Buzz!

Buzz! Rails 3.1

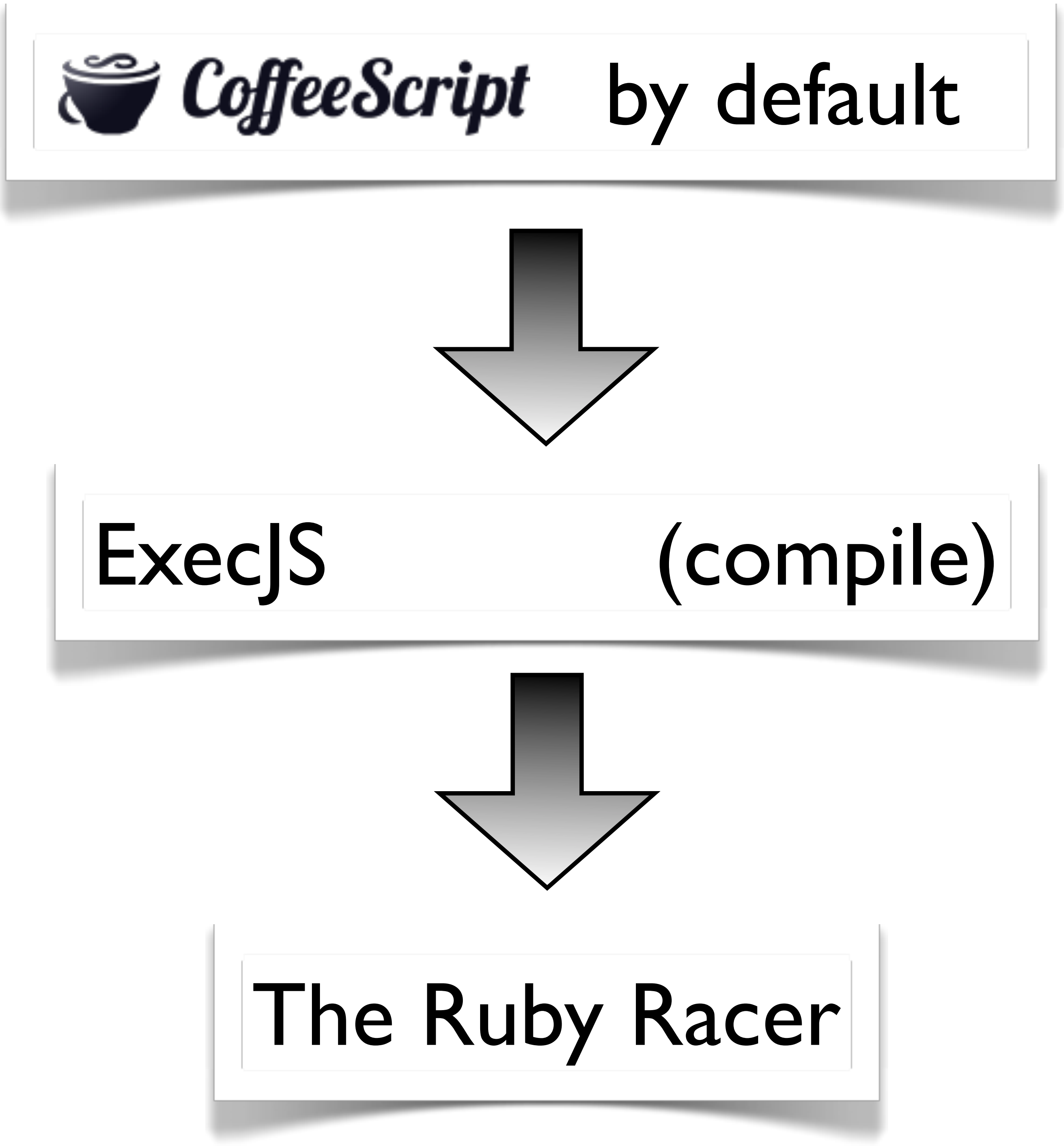
Buzz! Rails 3.1

CoffeeScript by default

Buzz! Rails 3.1



Buzz! Rails 3.1



THE FrontSide

But that’s just the cool kids

But that’s just the cool kids

why else?

Headless JavaScript Unit Testing

Headless JavaScript Unit Testing

Our first use at the FrontSide

THE FrontSide

Stealing Node.js utilities

Stealing Node.js utilities

less.js, handlebars.js, asciimo.js, jsdom.js, etc....

Client/Server code-sharing

Client/Server code-sharing

templating, validation, business logic

Performance

Performance

Re-implementing hotspots with V8 can be just as fast as with C.

Performance

Re-implementing hotspots with V8 can be just as fast as with C.

and all at a savings of over 10 rage units.

Performance

Re-implementing hotspots with V8 can be just as fast as with C.

and all at a savings of over 10 rage units.

-10x 😠

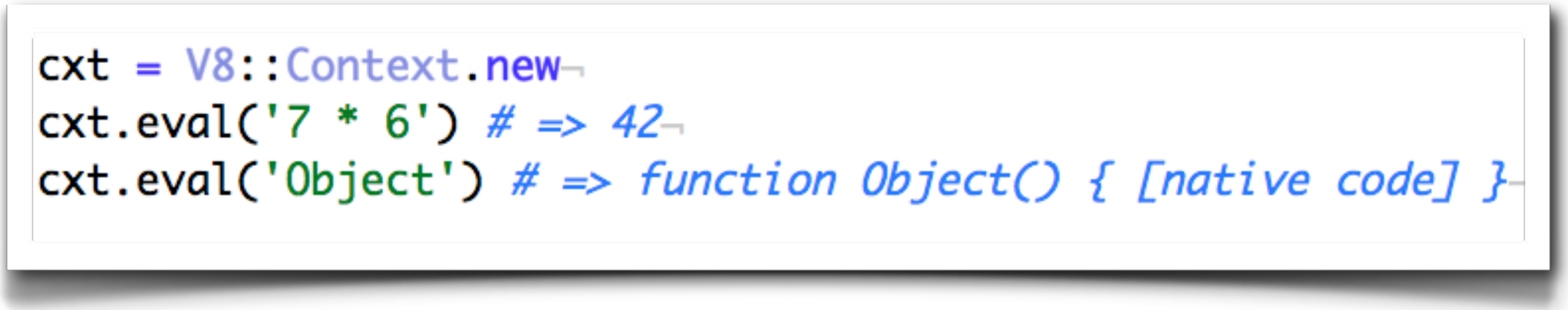
THE FrontSide

The Ruby Racer

What does embedding a JavaScript interpreter into your Ruby process mean?

Evaluate JavaScript from Ruby

Evaluate JavaScript from Ruby

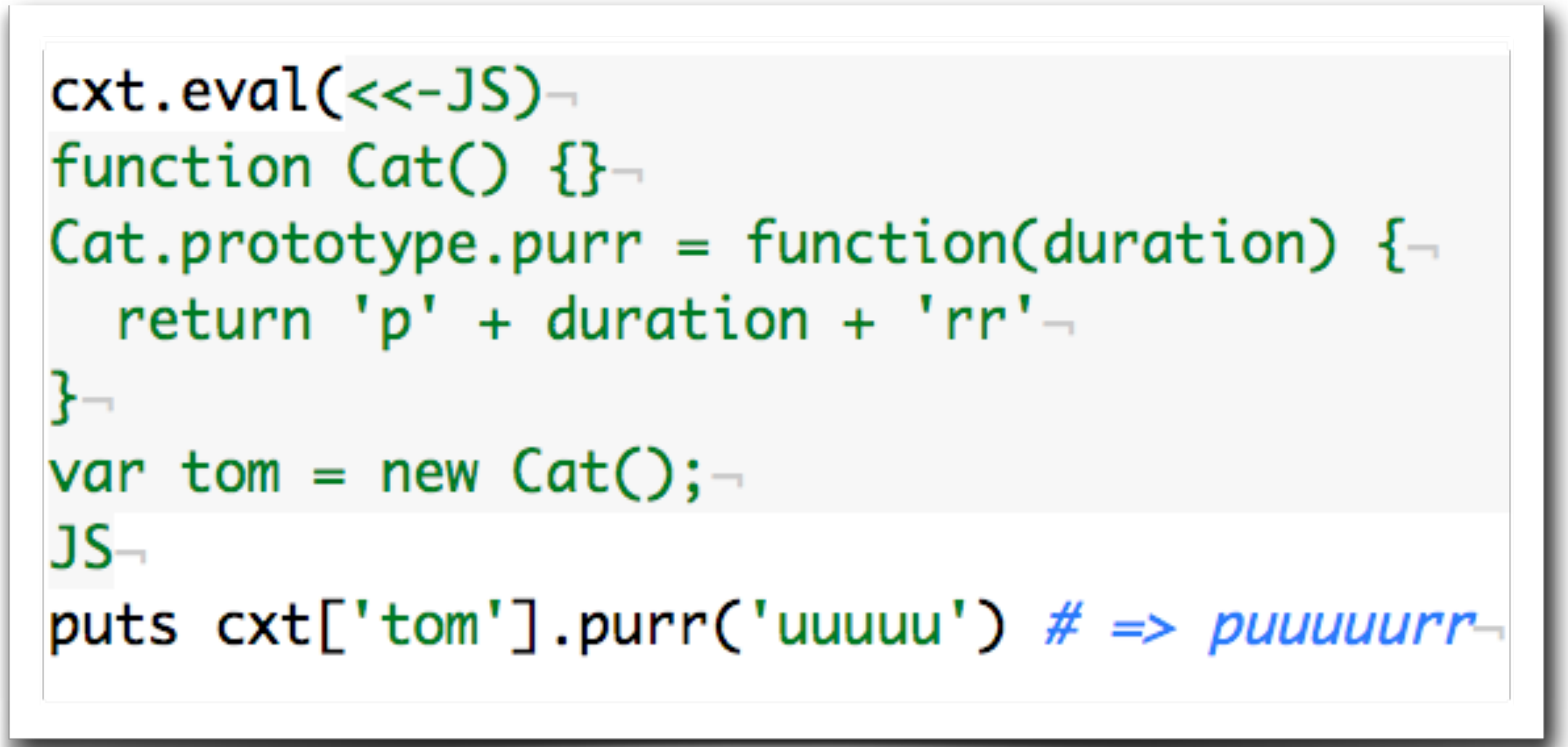


```ruby
cxt = V8::Context.new
cxt.eval('7 * 6') # => 42
cxt.eval('Object') # => function Object() { [native code] }
```

Access JavaScript Objects from Ruby

V8::Object

V8::Object



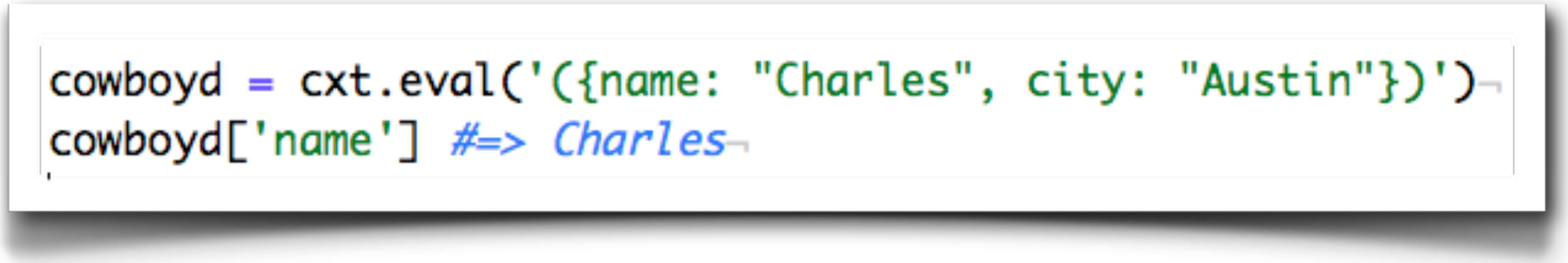
```
cxt.eval(<<-JS)
function Cat() {}
Cat.prototype.purr = function(duration) {
  return 'p' + duration + 'rr'
}
var tom = new Cat();
JS
puts cxt['tom'].purr('uuuuu') # => puuuuurr
```

V8::Object

Every JS Object looks kinda like a Ruby hash

V8::Object

Every JS Object looks kinda like a Ruby hash



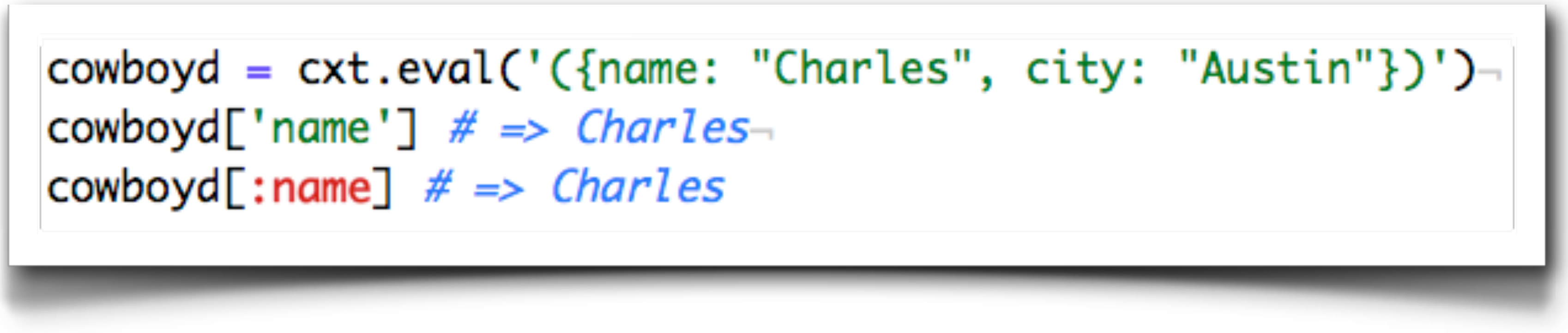
```
cowboyd = cxt.eval('({name: "Charles", city: "Austin"})')
cowboyd['name'] #=> Charles
```

V8::Object

symbol/string agnostic

V8::Object

symbol/string agnostic

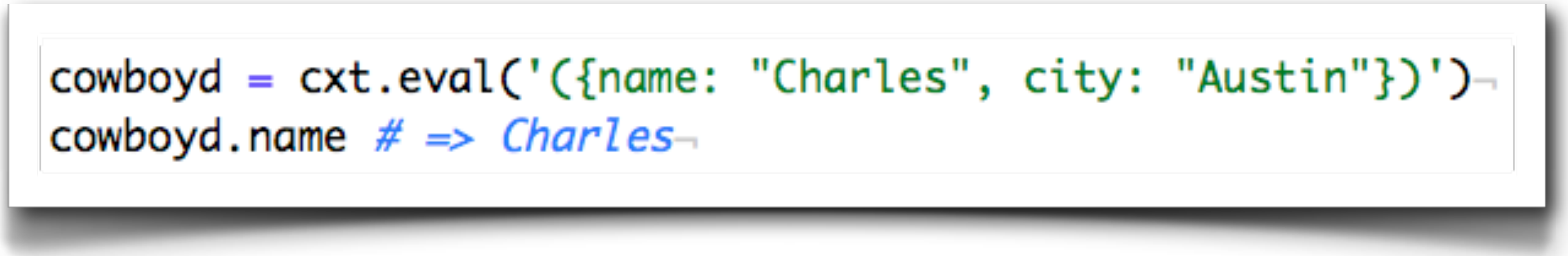

```
cowboyd = cxt.eval('({name: "Charles", city: "Austin"})')
cowboyd['name'] # => Charles
cowboyd[:name] # => Charles
```

V8::Object

steez: attr_reader

V8::Object

steez: attr_reader



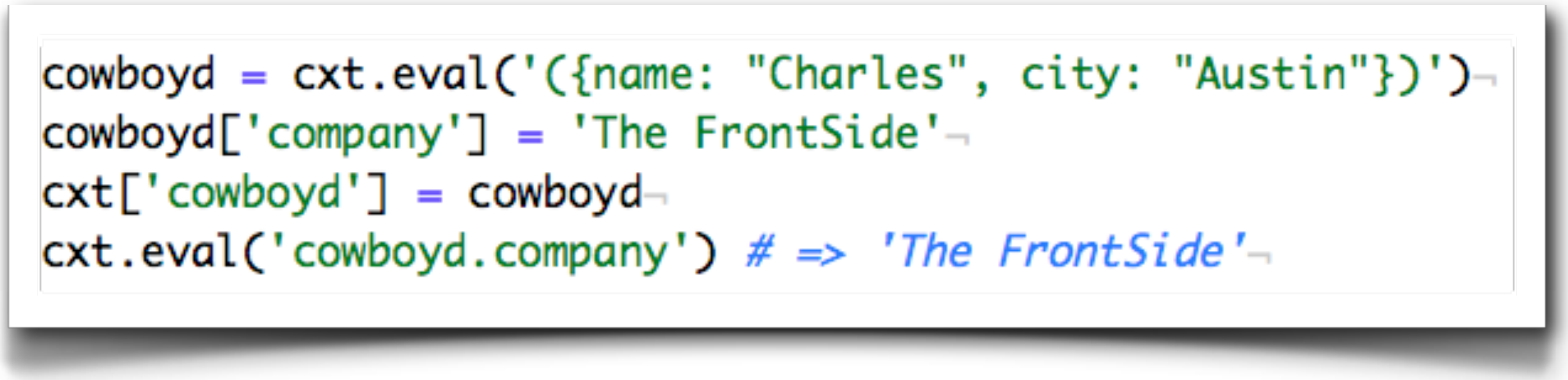
```
cowboyd = cxt.eval('({name: "Charles", city: "Austin"})')
cowboyd.name # => Charles
```

V8::Object

writeable? yes.

V8::Object

writeable? yes.



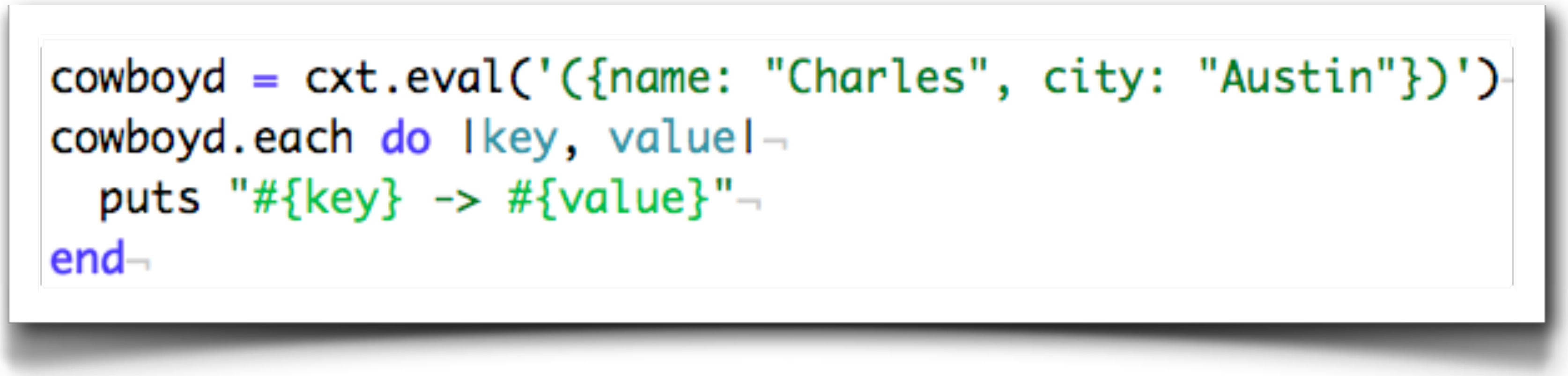
```
cowboyd = cxt.eval('({name: "Charles", city: "Austin"})')
cowboyd['company'] = 'The FrontSide'
cxt['cowboyd'] = cowboyd
cxt.eval('cowboyd.company') # => 'The FrontSide'
```

V8::Object

Enumerable? totally!

V8::Object

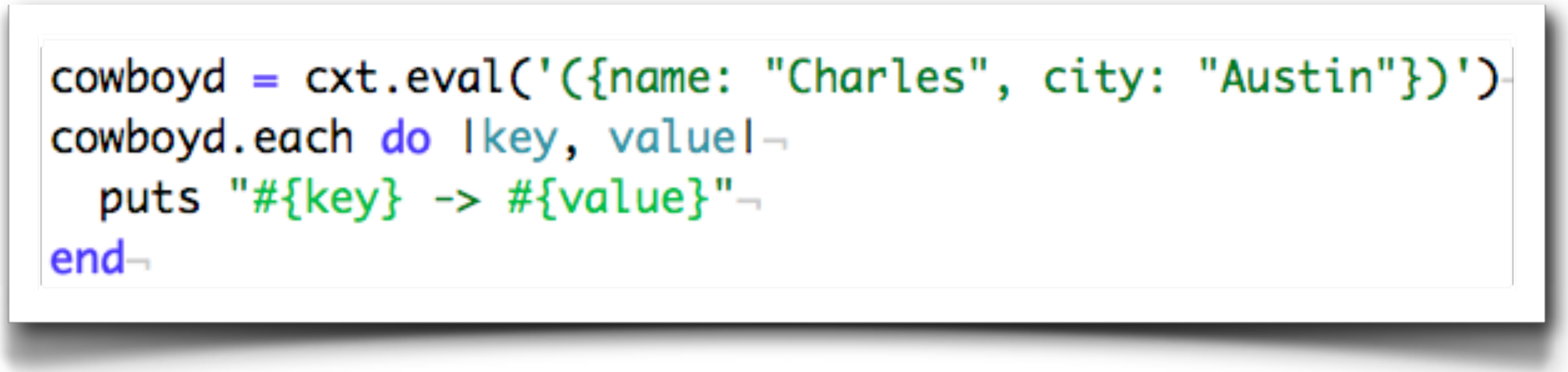
Enumerable? totally!



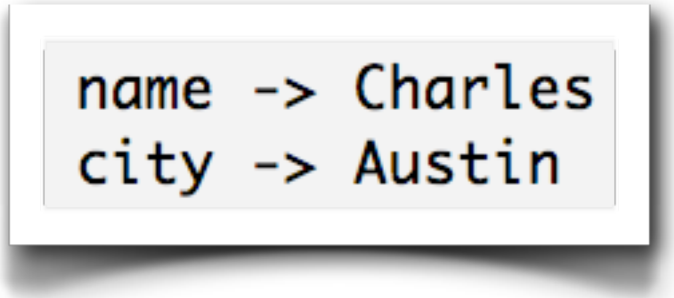
```
cowboyd = cxt.eval('({name: "Charles", city: "Austin"})')
cowboyd.each do |key, value|
  puts "#{key} -> #{value}"
end
```

V8::Object

Enumerable? totally!



```ruby
cowboyd = cxt.eval('({name: "Charles", city: "Austin"})')
cowboyd.each do |key, value|
  puts "#{key} -> #{value}"
end
```



```
name -> Charles
city -> Austin
```

V8::Object

Methods? no, but yes.

V8::Object

Methods? no, but yes.

In JavaScript, methods are just properties that happen to be functions

V8::Object

Methods? no, but yes.

In JavaScript, methods are just properties that happen to be functions

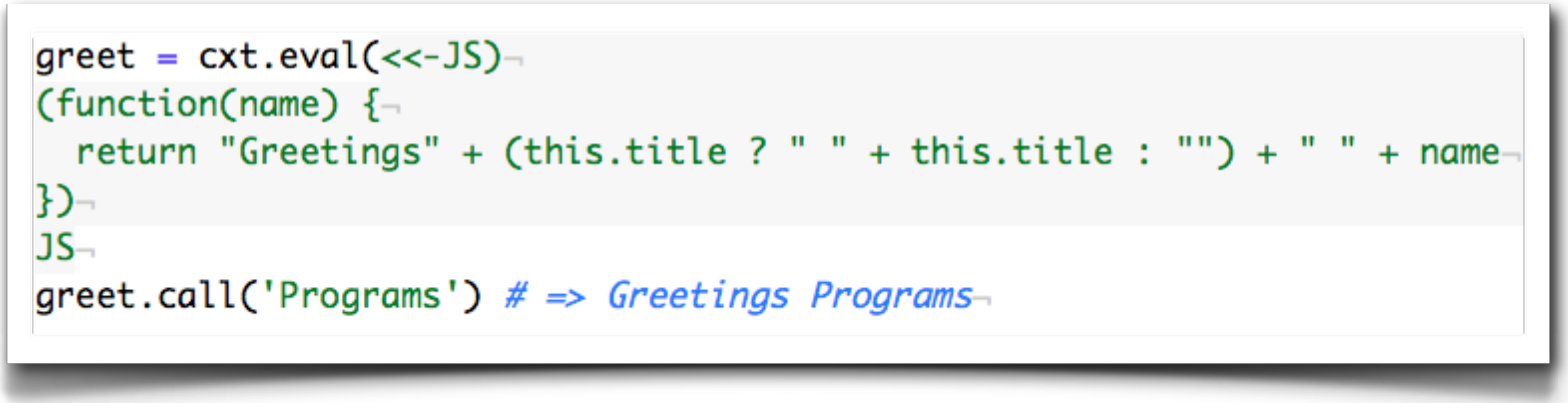
JavaScript function calls are “richer”

V8::Function

call it like a proc

V8::Function

call it like a proc



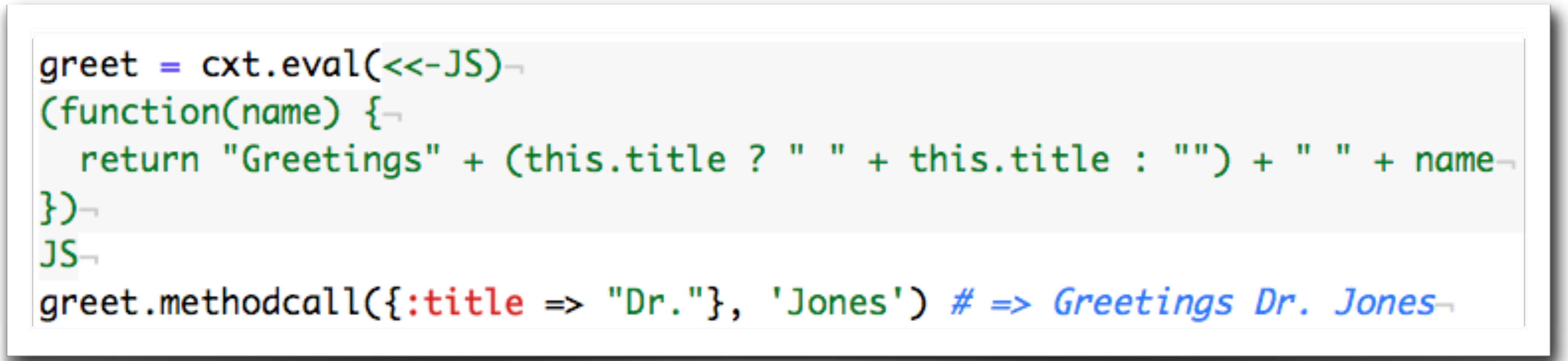
```
greet = cxt.eval(<<-JS)
(function(name) {
  return "Greetings" + (this.title ? " " + this.title : "") + " " + name
})
JS
greet.call('Programs') # => Greetings Programs
```

V8::Function

call it like a method

V8::Function

call it like a method



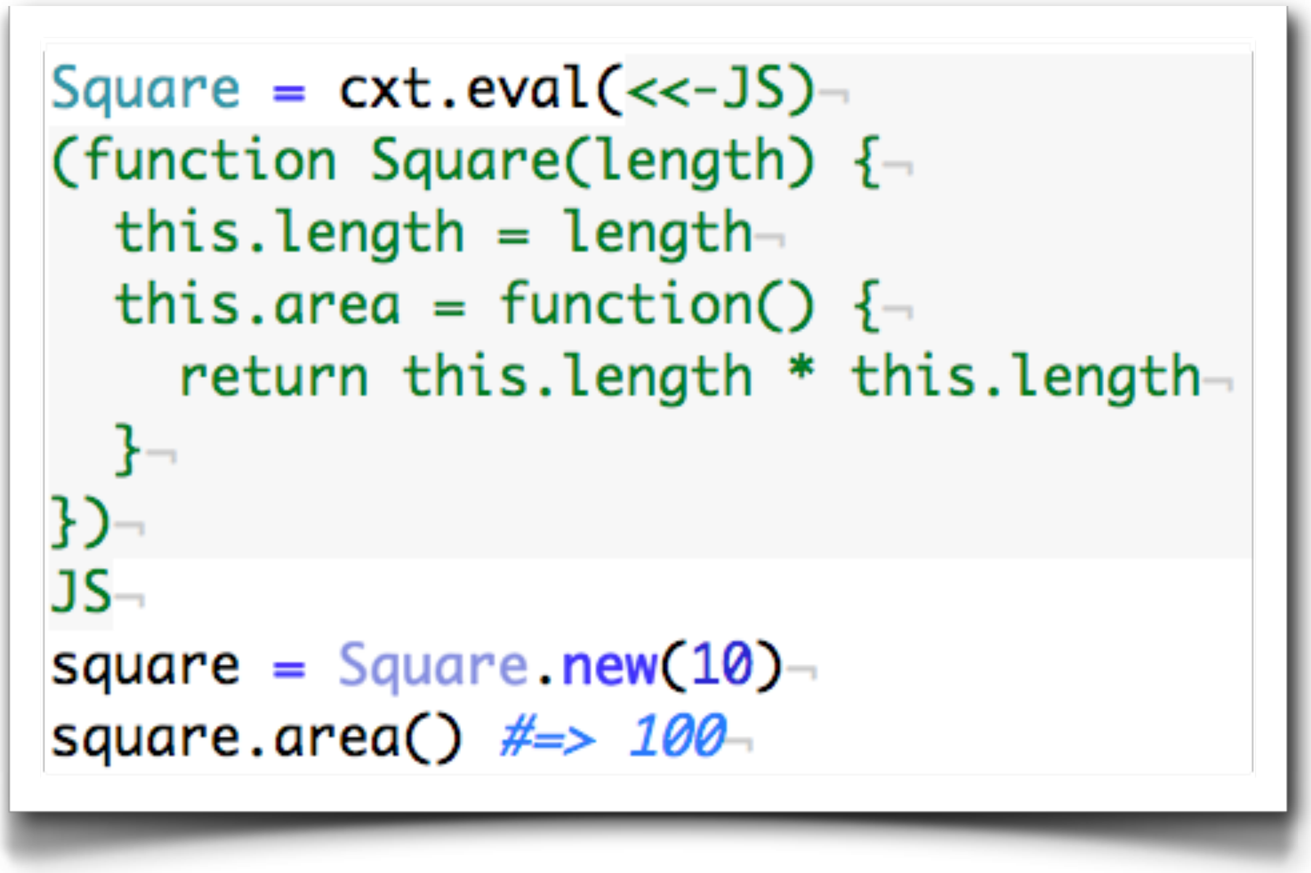
```
greet = cxt.eval(<<-JS)
(function(name) {
  return "Greetings" + (this.title ? " " + this.title : "") + " " + name
})
JS
greet.methodcall({:title => "Dr."}, 'Jones') # => Greetings Dr. Jones
```

V8::Function

call it like a constructor

V8::Function

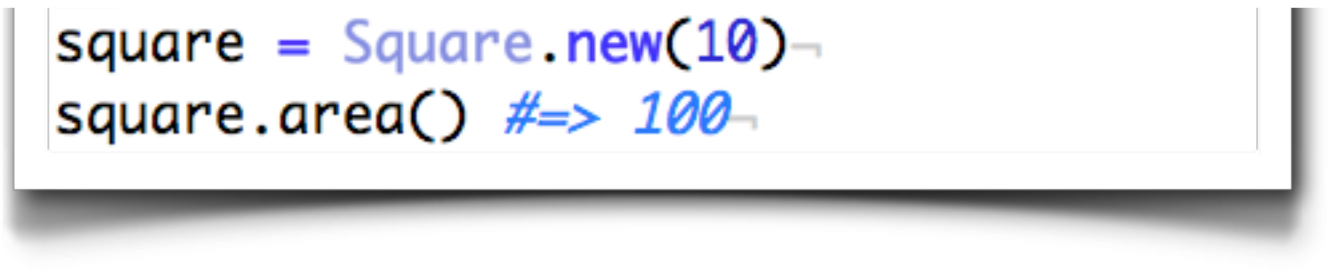
call it like a constructor



```
Square = cxt.eval(<<-JS)
(function Square(length) {
  this.length = length
  this.area = function() {
    return this.length * this.length
  }
})
JS
square = Square.new(10)
square.area() #=> 100
```

V8::Function

call it like a constructor



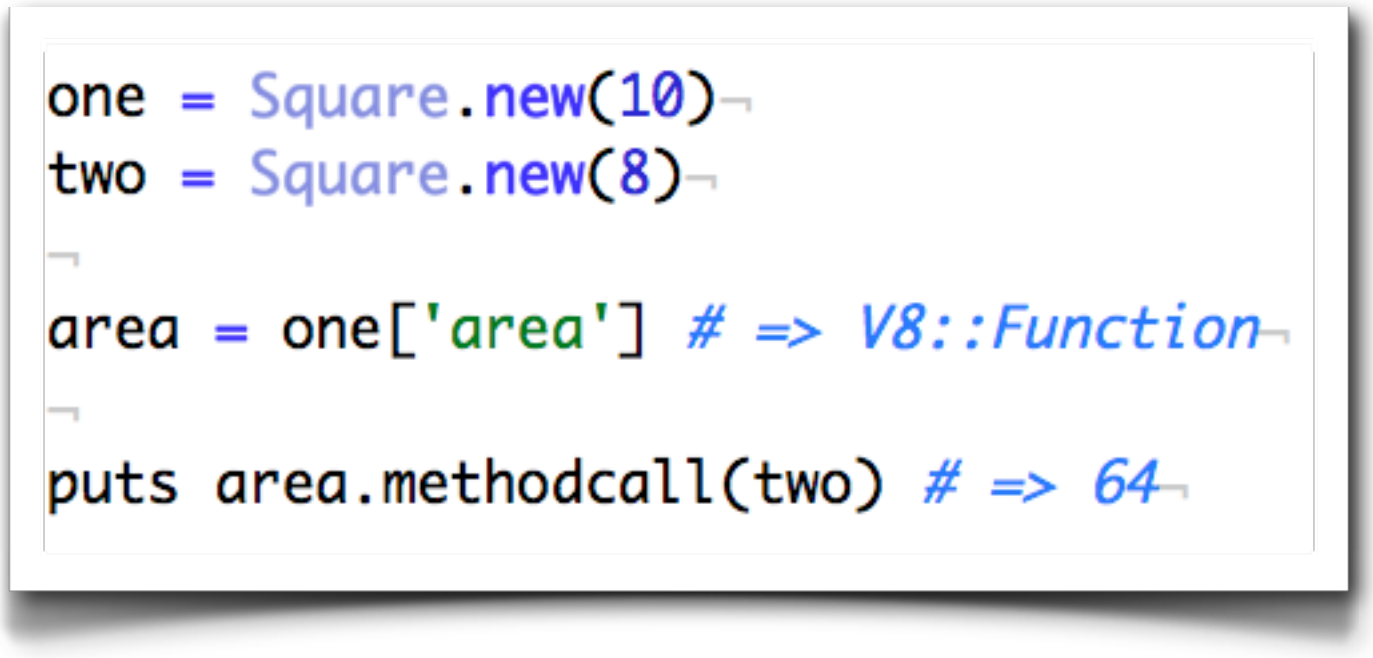
```
square = Square.new(10)
square.area() #=> 100
```

V8::Object

Methods? no, but yes.

In JavaScript, methods are just properties that happen to be functions

Methodish



```
one = Square.new(10)
two = Square.new(8)

area = one['area'] # => V8::Function

puts area.methodcall(two) # => 64
```

Call JavaScript from Ruby

Call JavaScript from Ruby

- eval() js code directly

Call JavaScript from Ruby

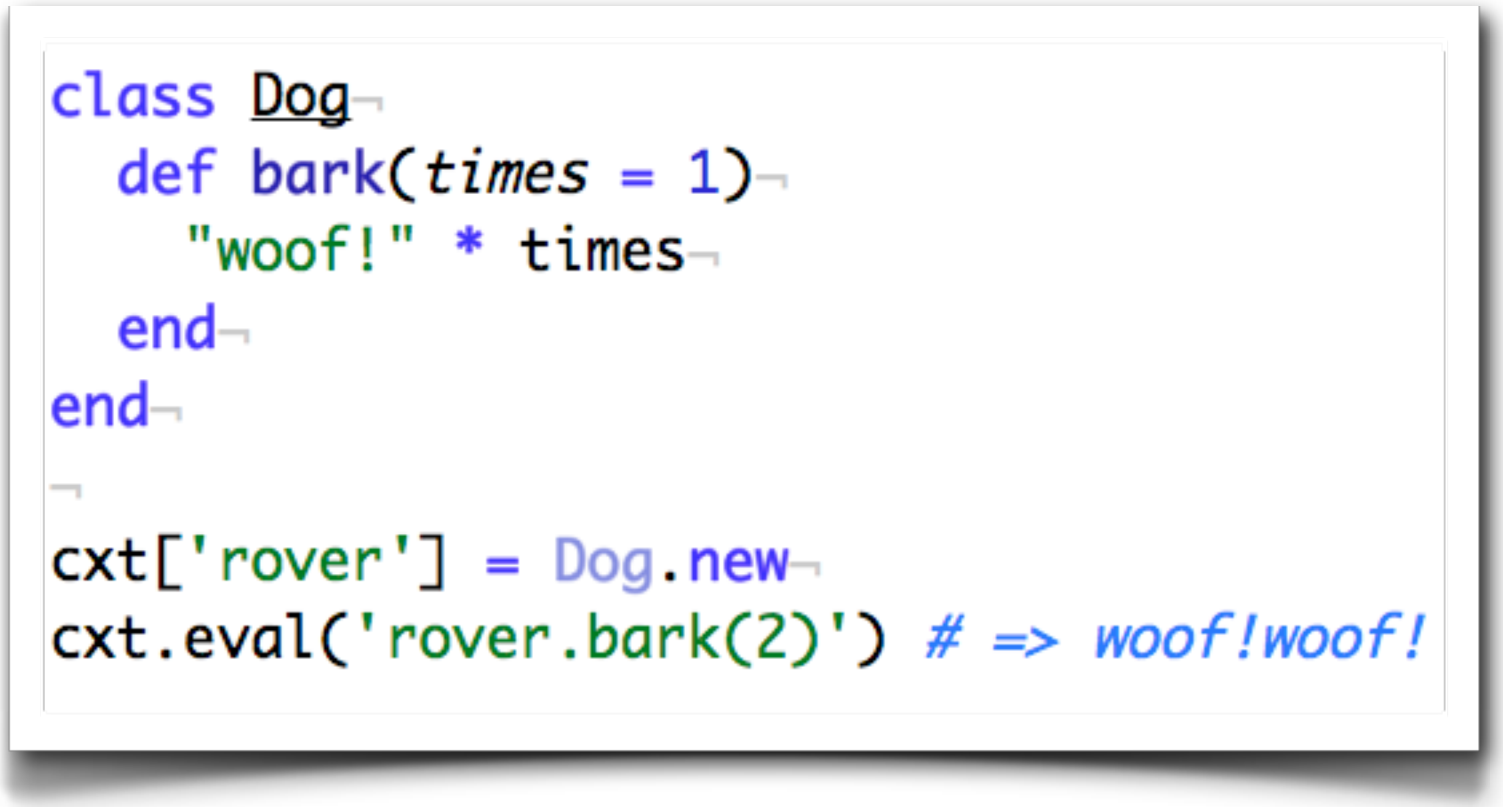
- eval() js code directly
- fiddle directly with js objects

Call JavaScript from Ruby

- eval() js code directly
- fiddle directly with js objects
- fiddle directly with js functions

Call Ruby code from JavaScript

Call Ruby code from JavaScript



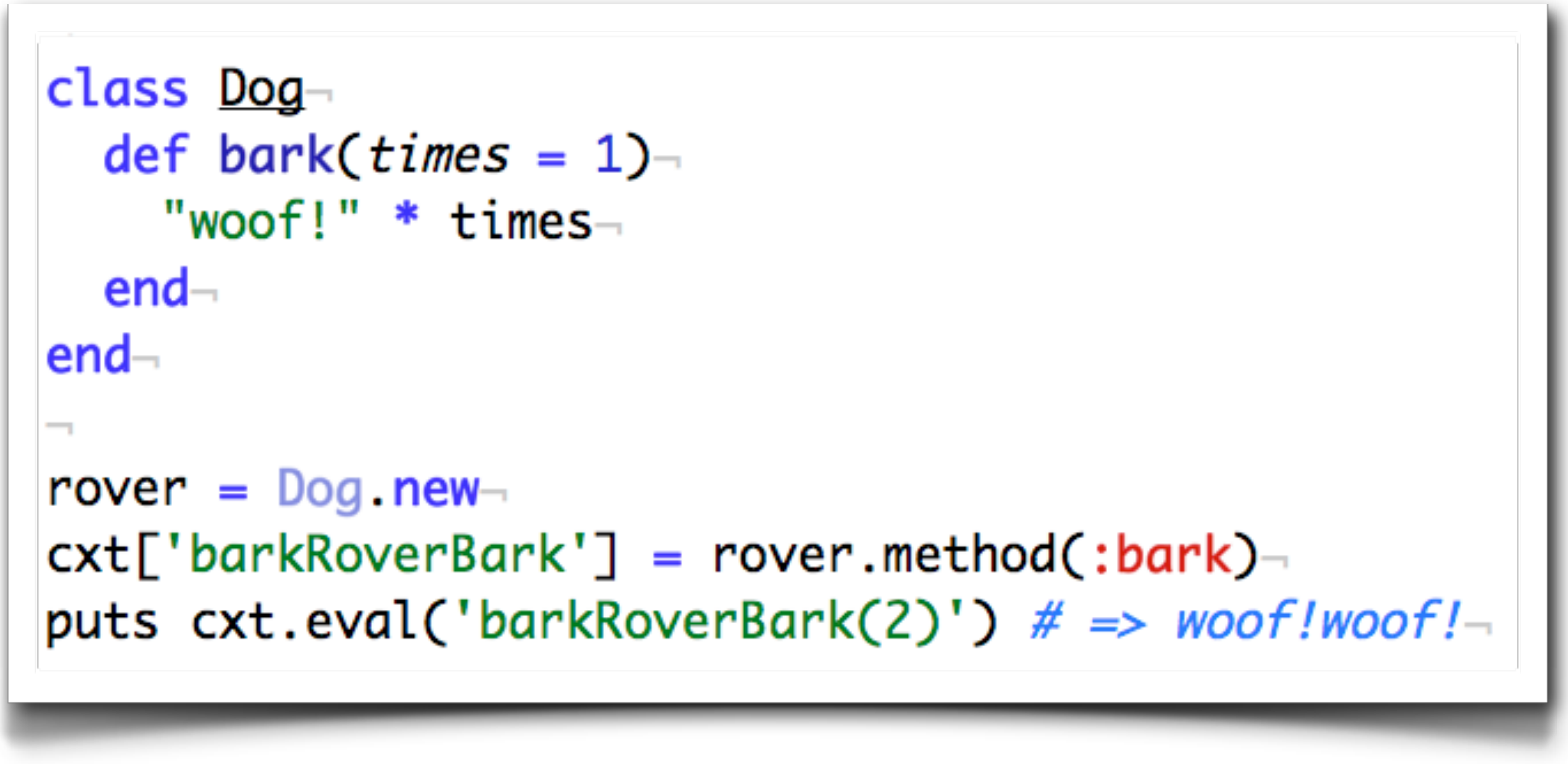
```
class Dog
  def bark(times = 1)
    "woof!" * times
  end
end

cxt['rover'] = Dog.new
cxt.eval('rover.bark(2)') # => woof!woof!
```

Call Ruby

bound methods

Call Ruby

bound methods



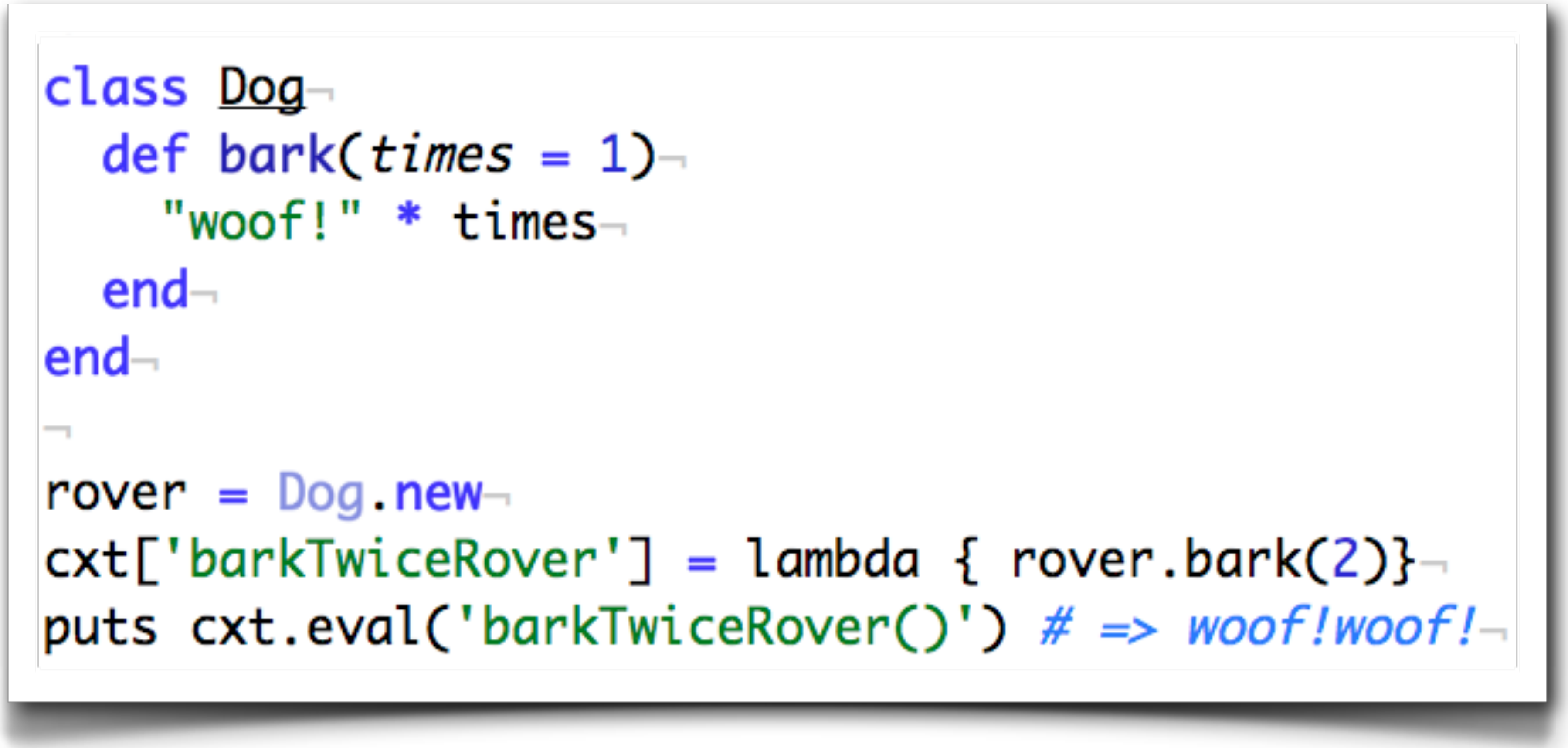
```ruby
class Dog
  def bark(times = 1)
    "woof!" * times
  end
end

rover = Dog.new
cxt['barkRoverBark'] = rover.method(:bark)
puts cxt.eval('barkRoverBark(2)') # => woof!woof!
```

Call Ruby

procs/lambdas

Call Ruby

procs/lambdas



```
class Dog
  def bark(times = 1)
    "woof!" * times
  end
end

rover = Dog.new
cxt['barkTwiceRover'] = lambda { rover.bark(2)}
puts cxt.eval('barkTwiceRover()') # => woof!woof!
```

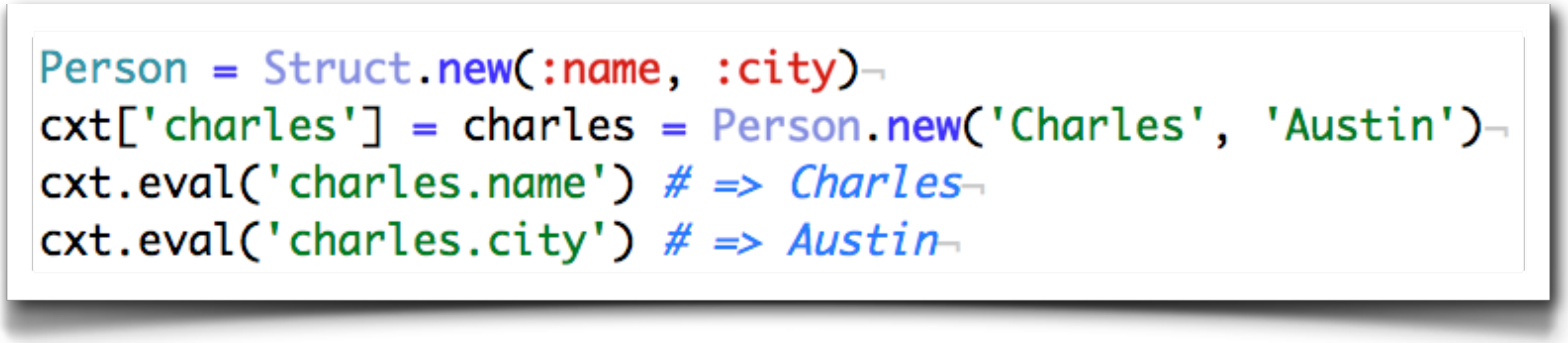
Ruby Properties

Properties

attr_reader

Properties

attr_reader



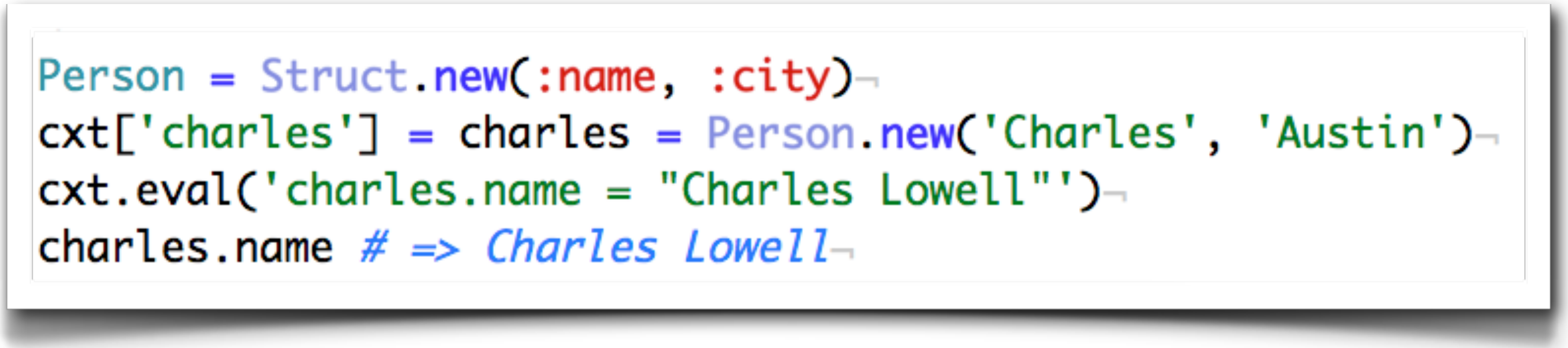
```
Person = Struct.new(:name, :city)¬
cxt['charles'] = charles = Person.new('Charles', 'Austin')¬
cxt.eval('charles.name') # => Charles¬
cxt.eval('charles.city') # => Austin¬
```

Properties

attr_accessor

Properties

attr_accessor



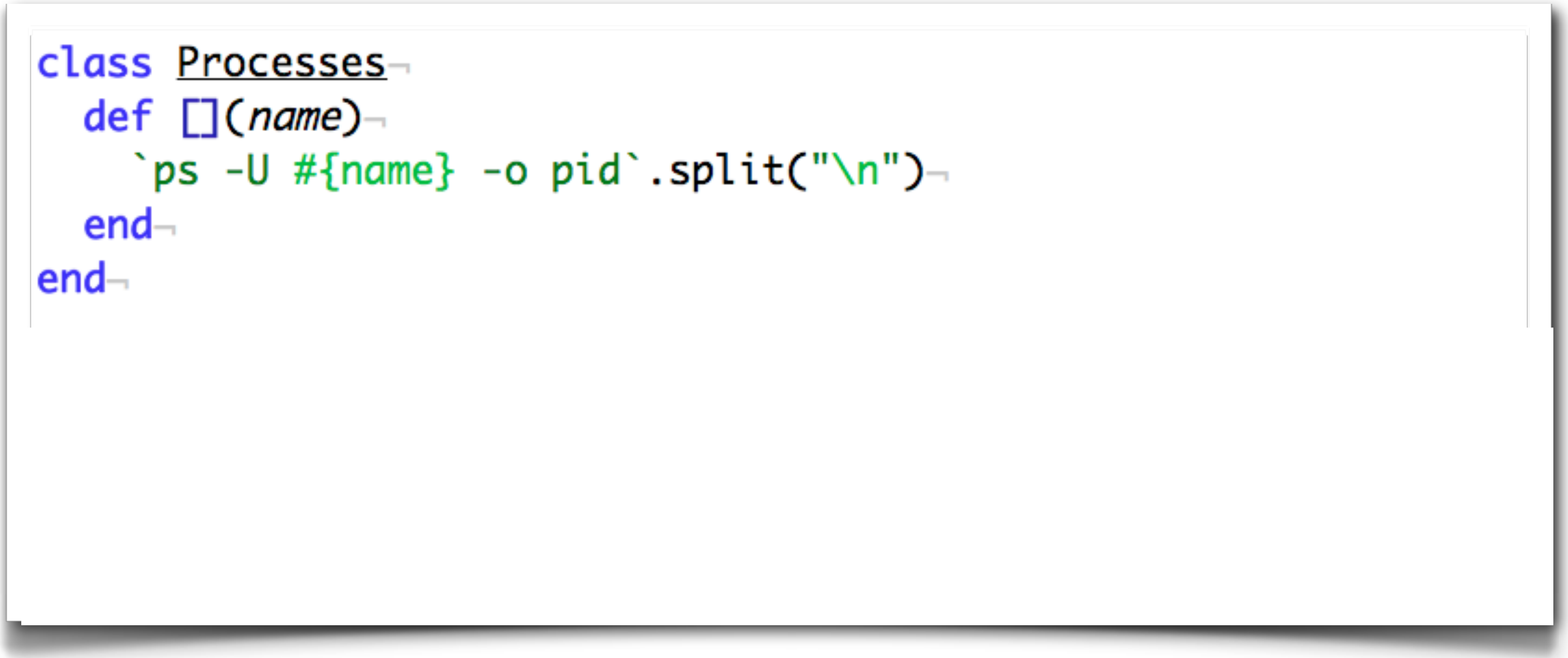
```
Person = Struct.new(:name, :city)
cxt['charles'] = charles = Person.new('Charles', 'Austin')
cxt.eval('charles.name = "Charles Lowell"')
charles.name # => Charles Lowell
```

Properties

Dynamic (hashish)

Properties

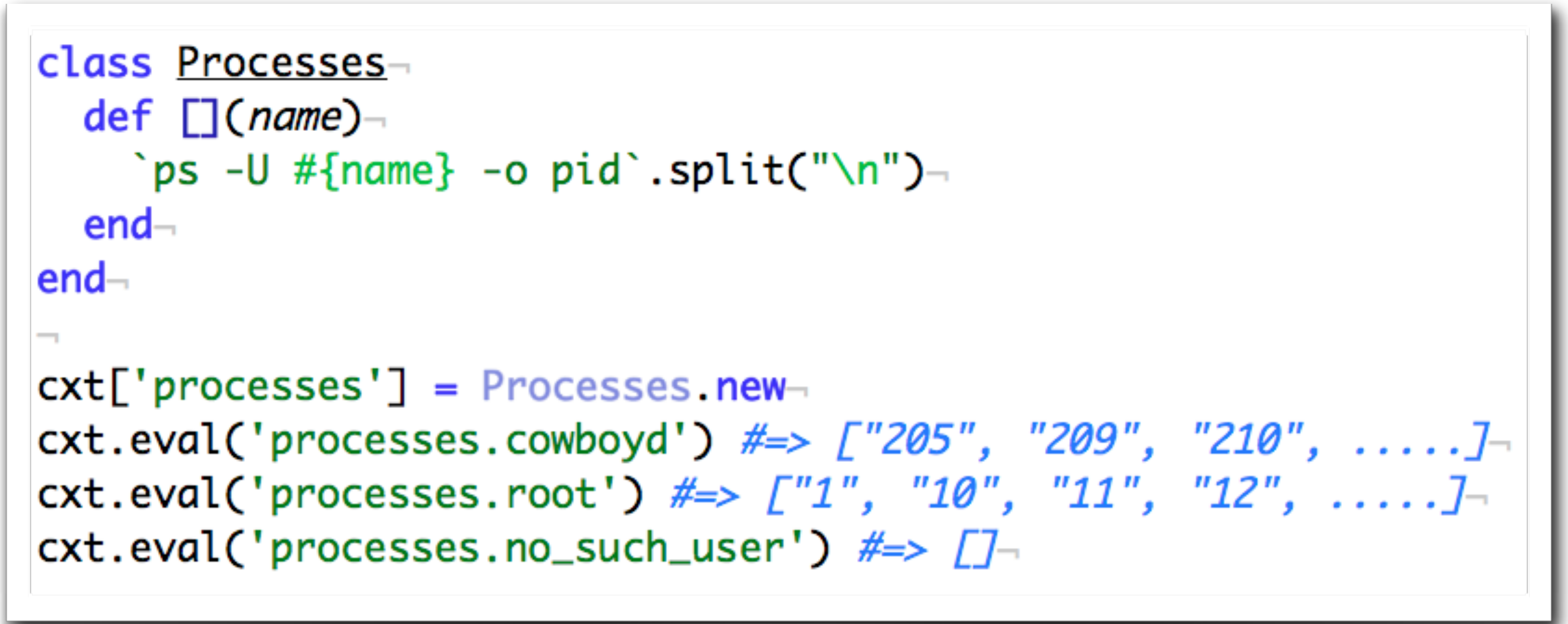
Dynamic (hashish)



```ruby
class Processes
  def [](name)
    `ps -U #{name} -o pid`.split("\n")
  end
end
```

Properties

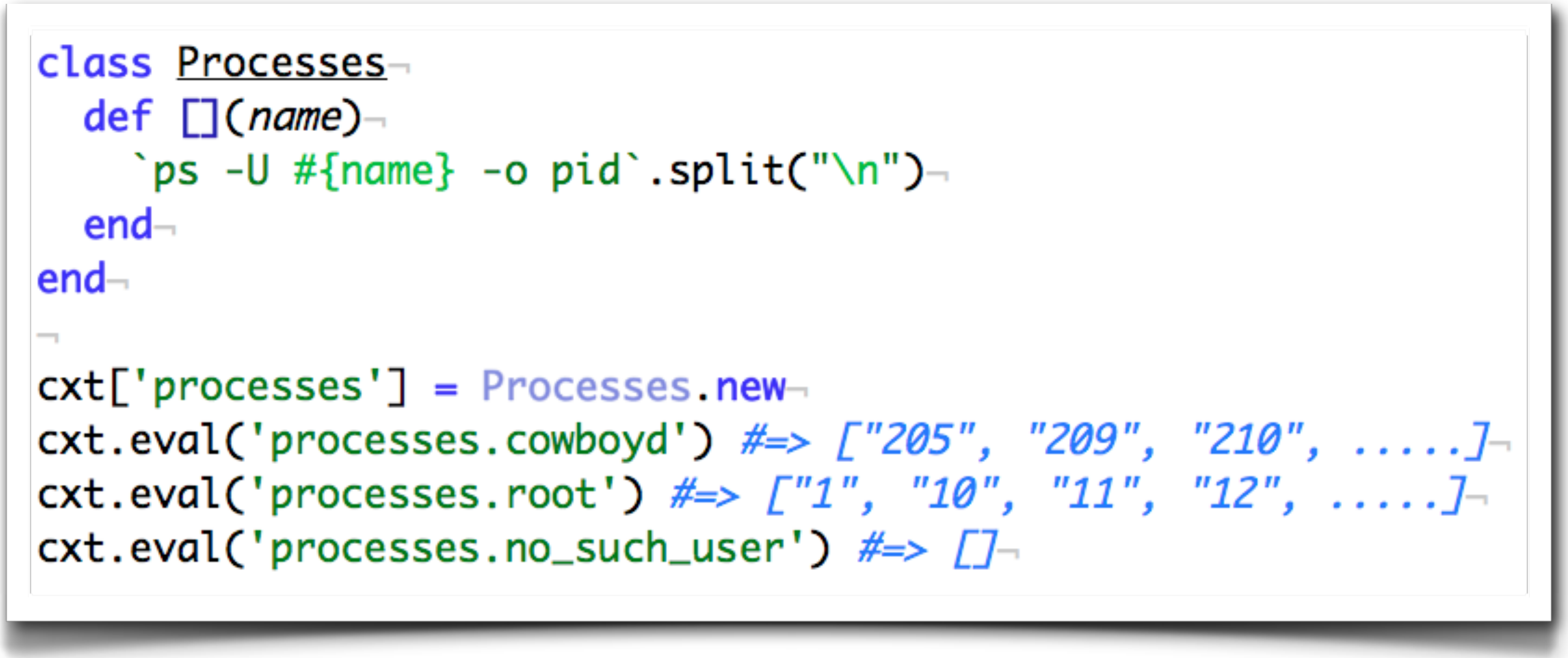
Dynamic (hashish)



```
class Processes
  def [](name)
    `ps -U #{name} -o pid`.split("\n")
  end
end

cxt['processes'] = Processes.new
cxt.eval('processes.cowboyd') #=> ["205", "209", "210", .....]
cxt.eval('processes.root') #=> ["1", "10", "11", "12", .....]
cxt.eval('processes.no_such_user') #=> []
```

Properties

Dynamic (hashish)



```ruby
class Processes
  def [](name)
    `ps -U #{name} -o pid`.split("\n")
  end
end

cxt['processes'] = Processes.new
cxt.eval('processes.cowboyd') #=> ["205", "209", "210", .....]
cxt.eval('processes.root') #=> ["1", "10", "11", "12", .....]
cxt.eval('processes.no_such_user') #=> []
```

method_missing() !

Access Ruby Code

- call ruby methods
- fiddle with ruby properties
- dynamically fiddle with ruby properties

The Ruby Racer

In the year 3000

Safe eval()

Safe eval()

```
while (true) {}
```

Safe eval()

```
while (true) {}
```

```
for (var i = 0; i < 10,000,000,000) {¬
  big.push(new Object())¬
}¬
```

Safe eval()

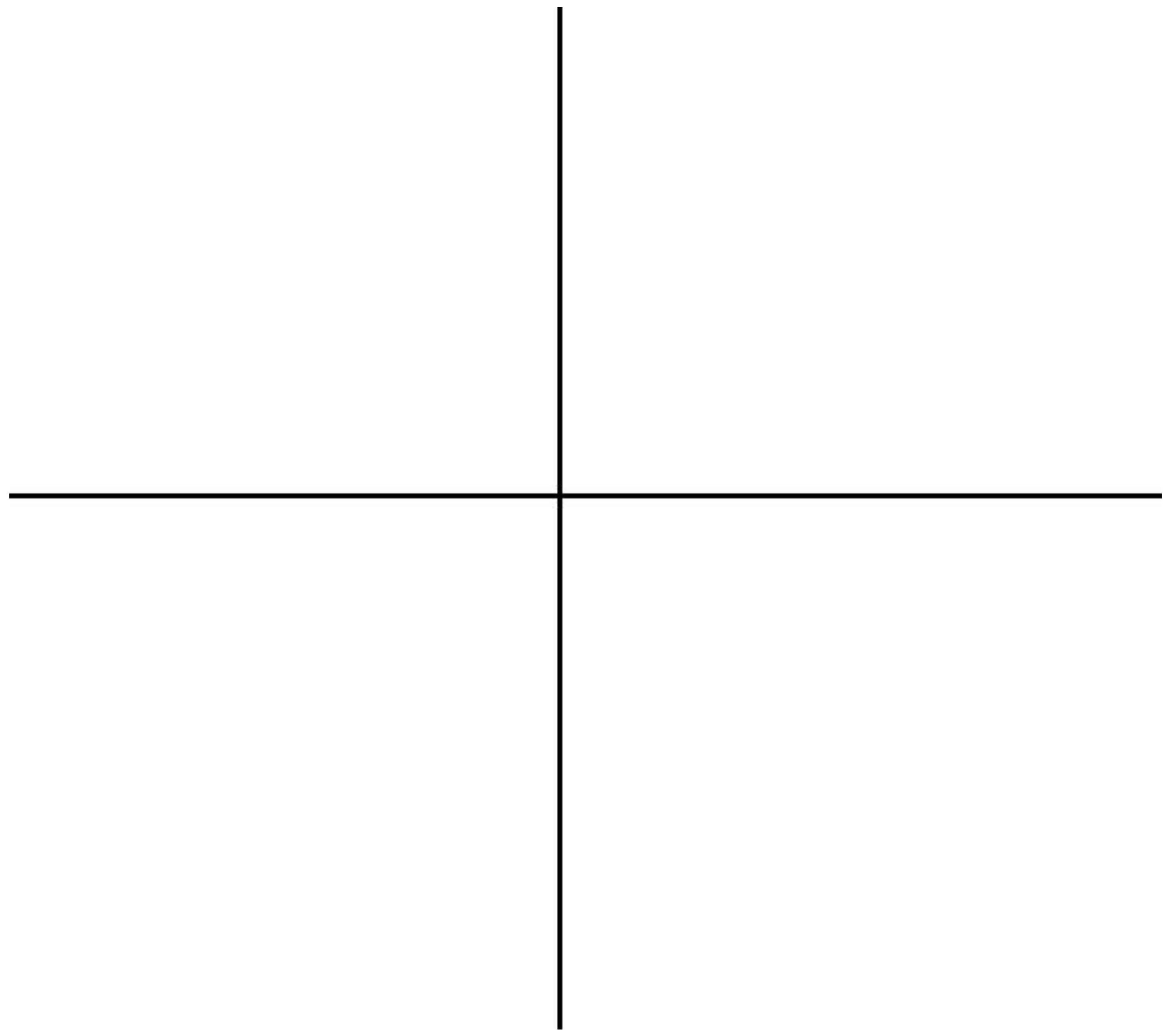
```
while (true) {}
```

```
for (var i = 0; i < 10,000,000,000) {¬
  big.push(new Object())¬
}¬
```

```
FileUtils.rm_rf('/')
```

Binary Gem

Binary Gem



Binary Gem

```
$ gem install therubyracer
```

Binary Gem

```
$ gem install therubyracer
```

```
$ gem install therubyracer
Building native extensions. This could take a while...
```

Binary Gem

```
$ gem install therubyracer
```

```
$ gem install therubyracer
Building native extensions. This could take a while...
```

5 minutes...

Binary Gem

```
$ gem install therubyracer
```

```
$ gem install therubyracer
Building native extensions. This could take a while...
```

5 minutes...

Other Stuff

- Multi threaded
- Custom Heap Snapshots
- Multiple V8 virtual machines per Ruby Process

The Ruby Racer

- github.com/cowboyd/therubyracer
- irc://irc.freenode.net/therubyracer
- therubyracer@googlegroups.com