MetaRails

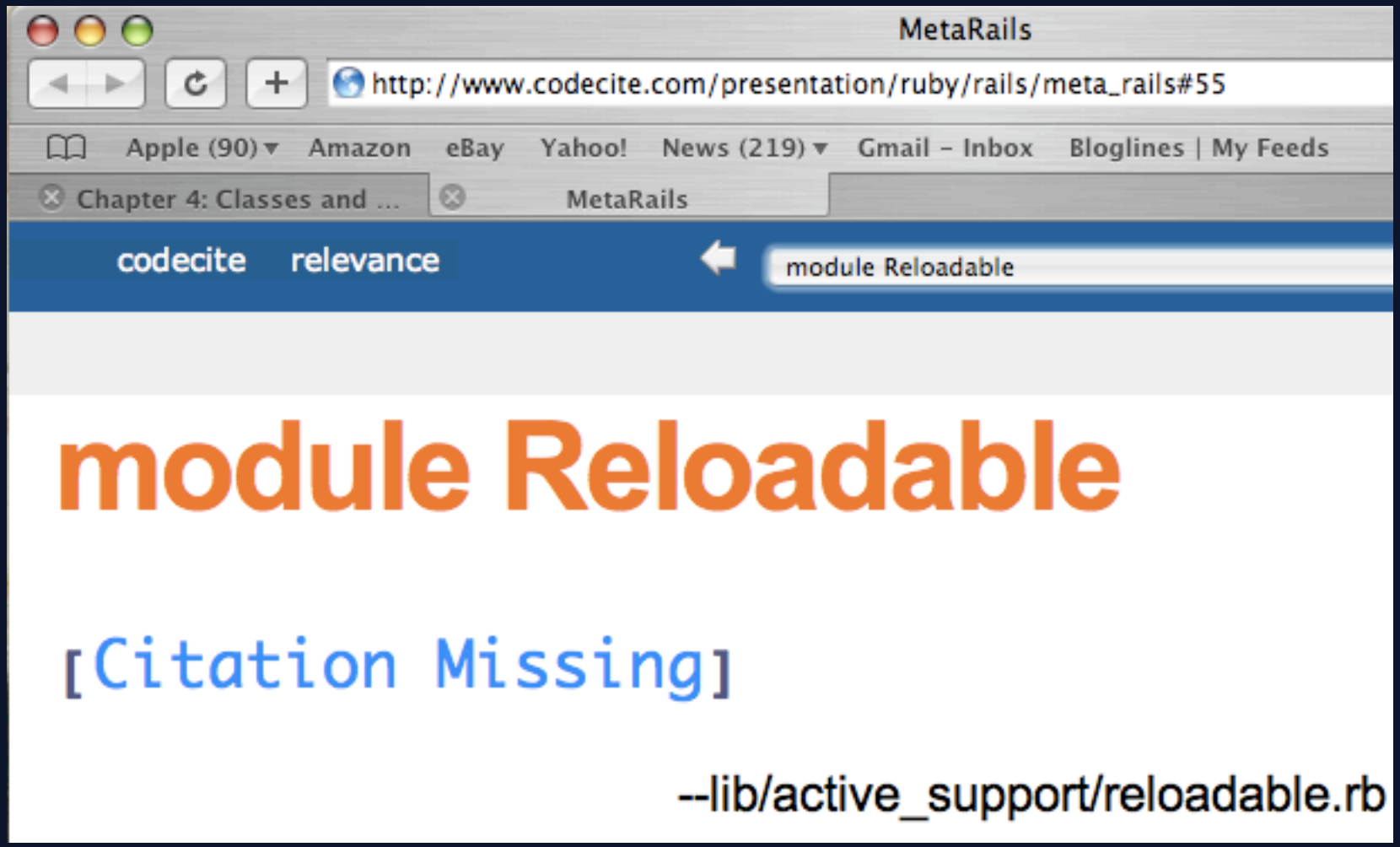
Metaprogramming by example

Nick Sieger

RUM - July 25, 2006

cue Stuart Holloway's MetaRails slides from RailsConf...

should be easy, right?


MetaRails
http://www.codecite.com/presentation/ruby/rails/meta_rails#55
Apple (90)
Amazon
eBay
Yahoo!
News (219)
Gmail - Inbox
Bloglines | My Feeds
Chapter 4: Classes and ...
MetaRails
codecite
relevance
module Reloadable
module Reloadable
[Citation Missing]
--lib/active_support/reloadable.rb

...maybe not

ruby's meta model

Class definition

```
class Fruit; end
```

Class definition

```
Fruit = Class.new
```

Method definition

```
class Fruit
  def peel; end
end
```

Method definition

```
Fruit.module_eval do
  def peel; end
end
```

Method definition

```
Fruit.module_eval "def peel; end"
```

Method invocation

```
f = Fruit.new
f.peel # => nil
```

Subclassing

```
class Apple < Fruit; end
```

Subclassing

```
Apple = Class.new(Fruit)
```

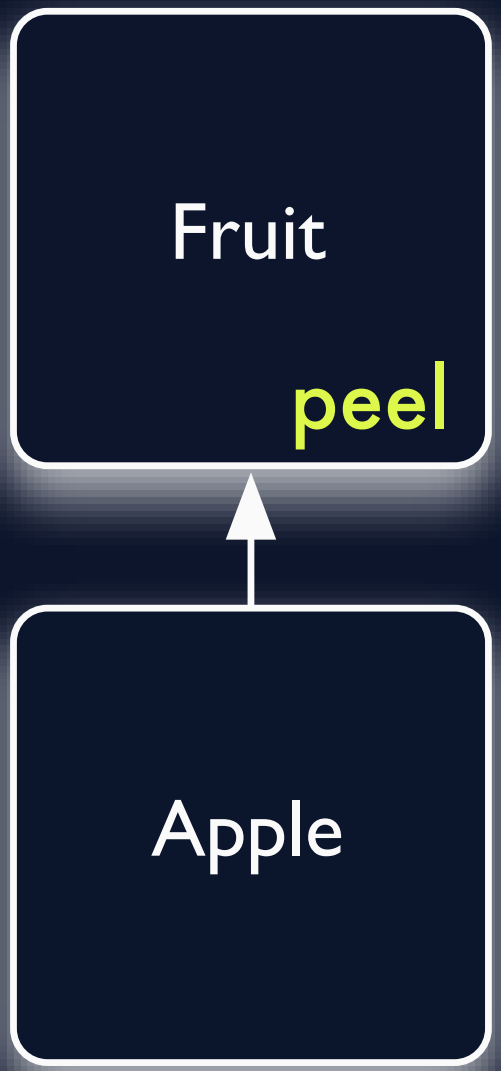
Class method

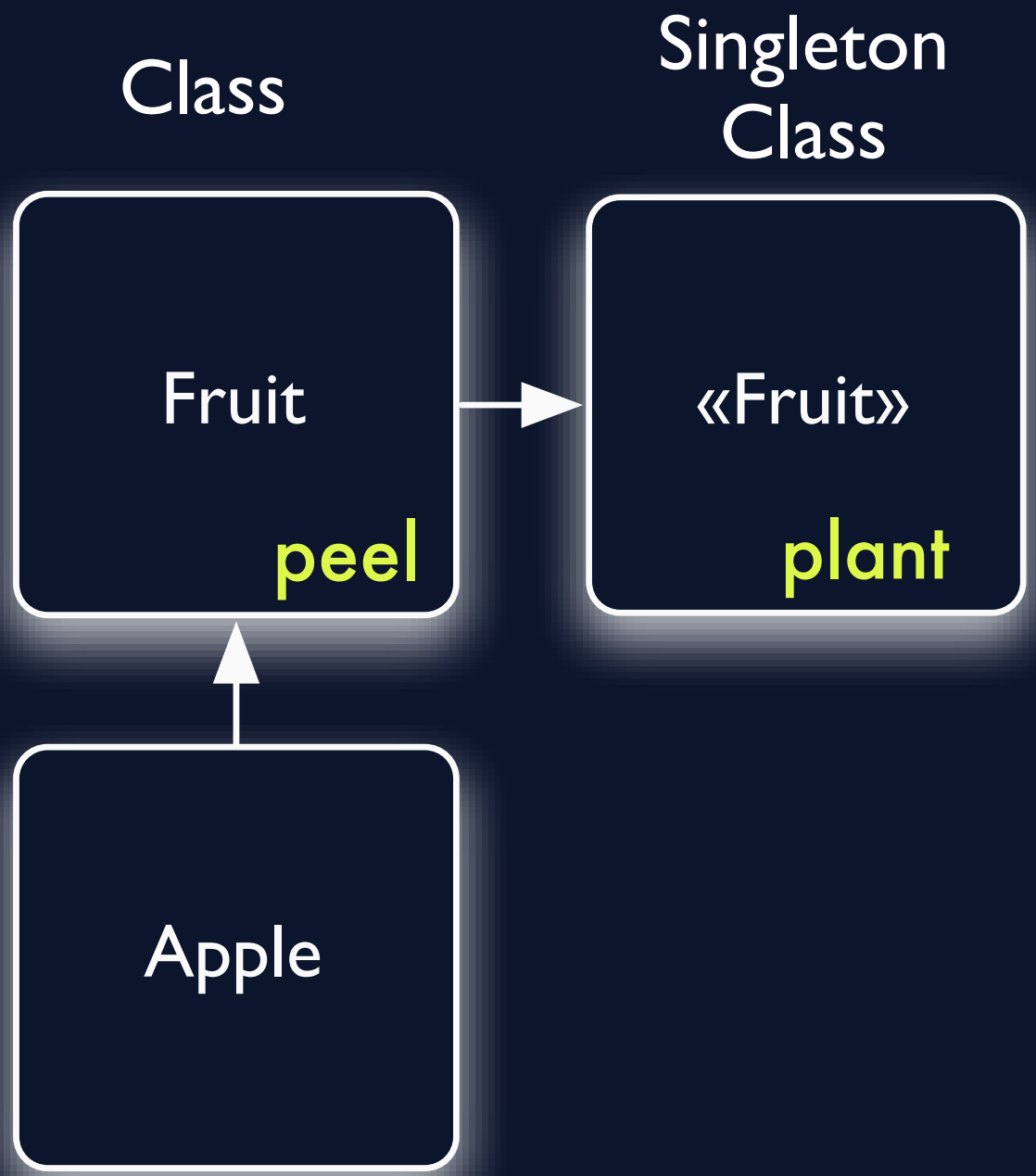
```
class Fruit
  def Fruit.plant
    raise "don't know"
  end
end
```

Class method invocation

```
Fruit.plant
  # => raises "don't know"
```

Class




Class
Singleton Class
Fruit
peel
«Fruit»
plant
Apple

Subclass w/ class method

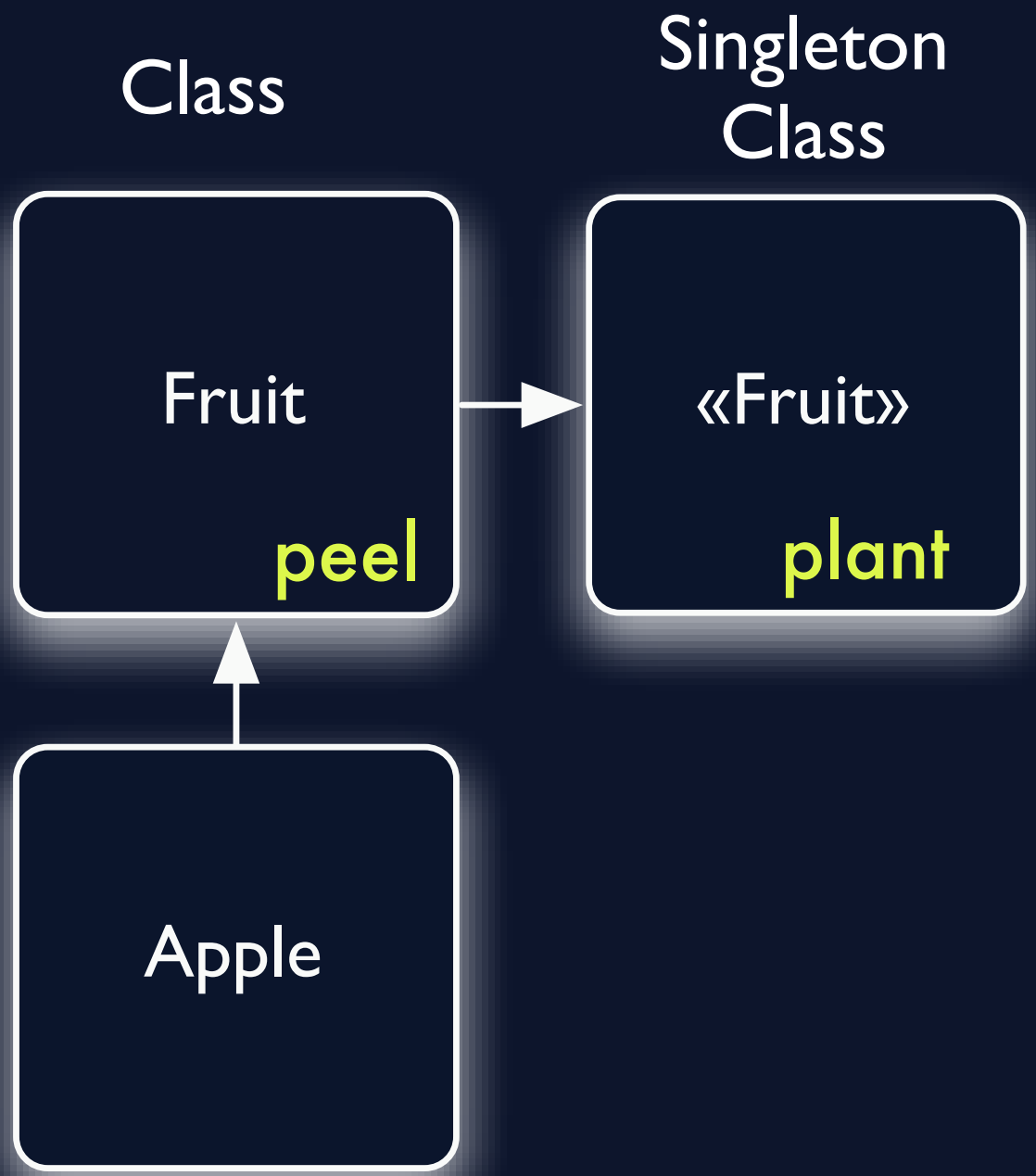
```
Apple.plant
```

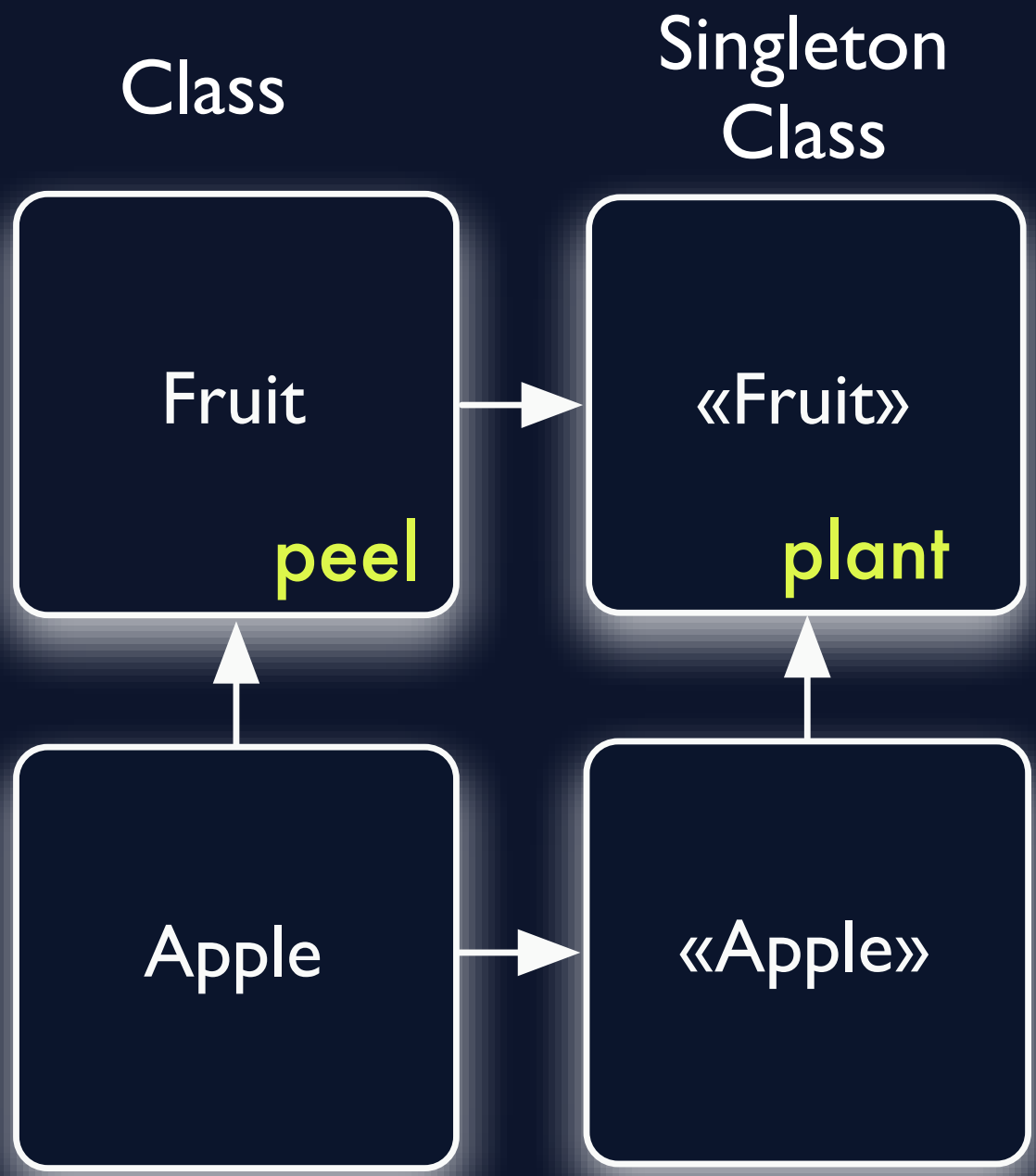
Subclass w/ class method

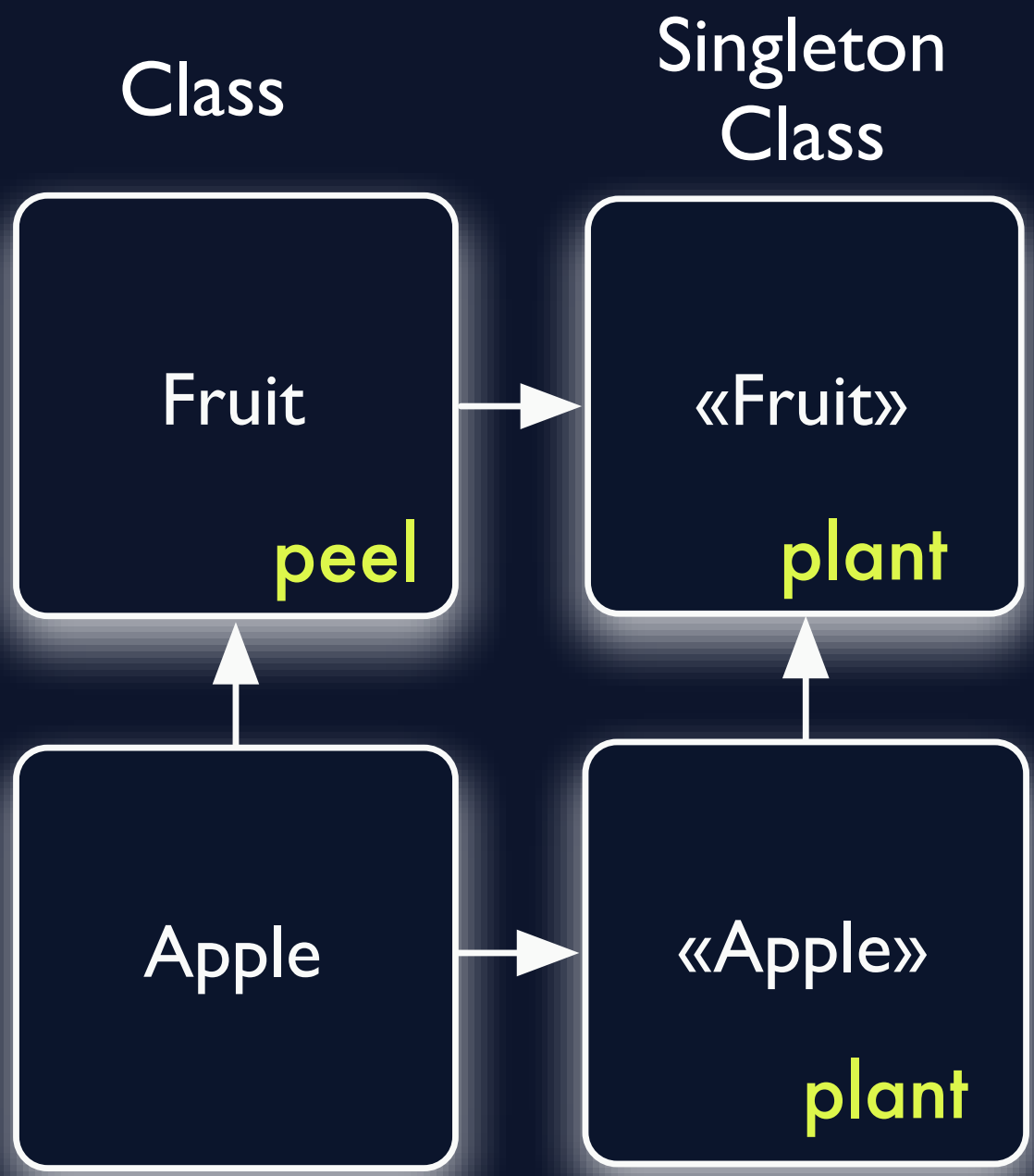
```
Apple.plant
   # => raises "don't know"
```

Override class method

```
def Apple.plant
  :apple_tree
end
```


Class
Singleton Class
Fruit
peel
«Fruit»
plant
Apple


Class
Singleton Class
Fruit
peel
«Fruit»
plant
Apple
«Apple»


Class
Singleton Class
Fruit
peel
«Fruit»
plant
Apple
«Apple»
plant

Override class method

```
Apple.plant
  # => :apple_tree
```

Accessing singleton class

```
class << Apple; self; end
```

«Apple»

plant

```
class << Apple
  def plant
    :apple_tree
  end
end

Apple.instance_eval do
  def plant
    :apple_tree
  end
end
```

Class	Singleton Class
holds instance methods	holds singleton/class methods
object.class	class << object; self; end
module_eval	instance_eval

Web development that doesn't hurt

Ruby on Rails is an open-source web framework that's optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.

ActiveSupport

humane web development

Numeric

```
10.minutes.ago
```

Numeric

```
class Numeric
  def minutes
    self * 60
  end

  def ago(time = ::Time.now)
    time - self
  end
end
```

Hash

```
class MyController < ActionController::Base
  def index1
    @message = params[:message]
  end
  def index2
    @message = params['message']
  end
end
```

Hash

```
class HashWithIndifferentAccess < Hash
  alias_method :regular_writer, :[]=
  def []=(key, value)
    regular_writer(convert_key(key), convert_value(value))
  end
  protected
  def convert_key(key)
    key.kind_of?(Symbol) ? key.to_s : key
  end
  def convert_value(value)
    value.is_a?(Hash) ? value.with_indifferent_access : value
  end
end
```

Hash

```
class Hash
  def with_indifferent_access
    hash = HashWithIndifferentAccess.new(self)
    hash.default = self.default
    hash
  end
end
```

Hash

```
module ActionController
  class AbstractRequest
    def parameters
      @parameters ||= request_parameters
        .update(query_parameters)
        .update(path_parameters)
        .with_indifferent_access
    end
  end
end
```

Array

- #in_groups_of

```
%w(1 2 3 4 5 6 7).in_groups_of(3) {|g| puts g}
  # => ["1", "2", "3"]
  #    ["4", "5", "6"]
  #    ["7", nil, nil]
```

Enumerable

- #index_by: convert enumerable of models to a hash

```
people.index_by(&:login)
  # => { "fred" => <Person ...>, "betty" => <Person ...>, ...}
```

Symbol-to-proc

```
people.select(&:manager?).collect(&:salary)

people.select { |p| p.manager? }.collect { |p| p.salary }
```

Symbol-to-proc

```ruby
class Symbol
  def to_proc
    Proc.new { |*args| args.shift.__send__(self, *args) }
  end
end
```

Whiny nil

```
# Extensions to nil which allow for more helpful error messages for
# people who are new to rails.
#
# The aim is to ensure that when users pass nil to methods where that isn't
# appropriate, instead of NoMethodError and the name of some method used
# by the framework users will see a message explaining what type of object
# was expected.
```

Who says Rails isn't newbie friendly!

Whiny nil

```
foo = nil
def report_errors
  begin; yield; rescue Exception =>e; puts e; end
end
```

Whiny nil

```
>> report_errors { foo.hello }
# You have a nil object when you didn't expect it!
# The error occured while evaluating nil.hello

>> report_errors { foo.id }
# Called id for nil, which would mistakenly be 4 --
# if you really wanted the id of nil, use object_id
```

Whiny nil

```
>> report_errors { foo.find "something" }
# You have a nil object when you didn't expect it!
# You might have expected an instance of Array.
# The error occured while evaluating nil.find

>> report_errors { foo.transaction }
# You have a nil object when you didn't expect it!
# You might have expected an instance of ActiveRecord::Base.
# The error occured while evaluating nil.transaction
```

Whiny nil

```
class NilClass
  def id
    raise RuntimeError, "Called id for nil...", caller
  end

  def method_missing(method, *args, &block)
    raise_nil_warning_for(
        @@method_class_map[method], method, caller)
  end
end
```

Theme	Example
mini-DSLs	add methods to core classes
nothing to remember	indifferent hash
verb-oriented programming	interpret symbols as procs
learn from mistakes, then automate	whiny nil
guesses may be better than certainty	whiny nil

ActiveRecord

your personal database programming assistant

Example

```
class CreateTeams < ActiveRecord::Migration
  def self.up
    create_table :teams do |t|
      t.column :name, :string
      t.column :mascot, :string
      t.column :updated_at, :datetime
      t.column :created_at, :datetime
    end
  end
  def self.down
    drop_table :teams
  end
end
```

Example

```
class Team < ActiveRecord::Base
end
```

Magic

```
>> Team.methods.find_all {|x| x=~/find_by/}
# ["find_by_sql"]
>> bulls = Team.find_by_name("Bulls")
# <#Team...>
>> bulls.mascot
# "Bull"
>> puts Team.instance_methods.select {|x| x=~/^(name|mascot)/}
# ["mascot", "name", "name?", "mascot?" ]
```

method_missing

```ruby
class ActiveRecord::Base
  def self.method_missing(method_id, *arguments)
    # dynamic finder logic
  end

  def method_missing(method_id, *args, &block)
    # lazily create attribute methods
  end
end
```

Has Many

```ruby
class Team < ActiveRecord::Base
  has_many :players
end
```

```ruby
class ActiveRecord::Base
  class << self
    def has_many(association_id, options = {}, &extension)
      reflection = create_has_many_reflection(...)
      collection_accessor_methods(reflection, HasManyAssociation)
    end

    def collection_reader_method(reflection, association_proxy_class)
      # force reload code omitted
      define_method(reflection.name) do |*params|
        association = instance_variable_get("@#{reflection.name}")
        unless association.respond_to?(:loaded?)
          association = association_proxy_class.new(self, reflection)
          instance_variable_set("@#{reflection.name}", association)
        end
        association
      end
    end
  end
end
```

Theme	Example
borrowing other object models	accessors inferred from schema
object model on demand	method_missing
configuration is code	association helpers

Thank you!

props to Stuart Halloway for the original MetaRails