Agile Webentwicklung mit Ruby on Rails

Prof. Dr. Oliver Vornberger
Nils Haldenwang, B.Sc.

UNIVERSITÄT OSNABRÜCK

Institut für Informatik
AG Medieninformatik

Institut für Informatik
Prof. Dr. Oliver Vornberger
Nils Haldenwang, B.Sc.

Universität Osnabrück
http://www-lehre.inf.uos.de/~ror
26.04.2012

Agile Webentwicklung mit Ruby on Rails

Sommersemester 2012

Blatt 1

Aufgabe 1: Sortieren

Übersetzen Sie einen oder mehrere aus der Veranstaltung *Algorithmen und Datenstrukturen* bekannten Sortieralgorithmus Ihrer Wahl von Java nach Ruby.

Schreiben Sie dazu eine Methode, die nach dem Sortieralgorithmus benannt ist und als Parameter ein Array mit ganzen Zahlen erwartet. Der Rückgabewert soll das sortierte Array sein.

Aufgabe 2: Array summieren

Ruby

Einführung in die Syntax

Exceptions

Ruby

Einführung in die Syntax

Syntax

Fehler behandeln

```
begin
  # something that can raise IOError
rescue IOError => e
  puts e # show error message
ensure
  # this is guaranteed to happen
end

raise StandardError, "message goes here"
```

Fehler werfen

Ruby

Exception Hierarchie

- Exception
  - fatal
  - NoMemoryError
  - SecurityError
  - StandardError
    - ArgumentError
    - IOError
    - NameError
    - TypeError

Oberklasse aller Exceptions

Systemfehler

Oberklasse für Applikationsfehler

Programming Ruby, S. 153

Module

Ruby

Einführung in die Syntax

Module als Namespaces

```ruby
module MyMath
  def MyMath.square(number)
    number * number
  end
end

puts MyMath.square(2)
# => 4

puts MyMath::square(2)
# => 4
```


Ruby

Module als Mixins

```ruby
module Squarable
  def square
    self * self
  end
end

class Fixnum
  include Squarable
end

puts 2.square    # => 4

class Float
  include Squarable
end

puts 2.0.square # => 4.0
```

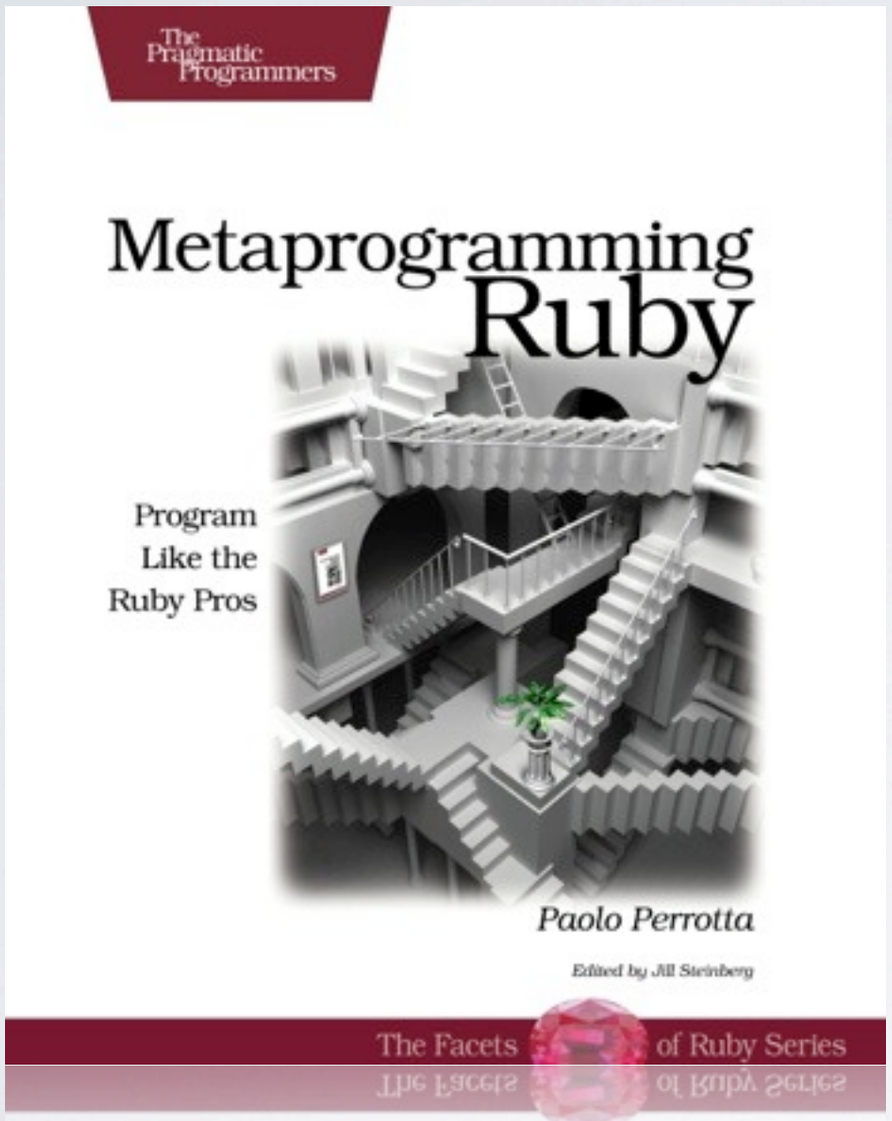
Einbindung beliebig vieler Module möglich

Ruby

Ruby

Object Model

Quellen / Literaturempfehlungen



Paolo Perrotta,
Metaprogramming Ruby,
Pragmatic Bookshelf, 2010

S. 3-35.

Instanzen und Klassen

Ruby

Object Model

Klassendefinition

```ruby
class MyClass
  def my_method
    @v = 1
  end
end

obj = MyClass.new
obj.class # => MyClass
```


Ruby

Instanzvariablen

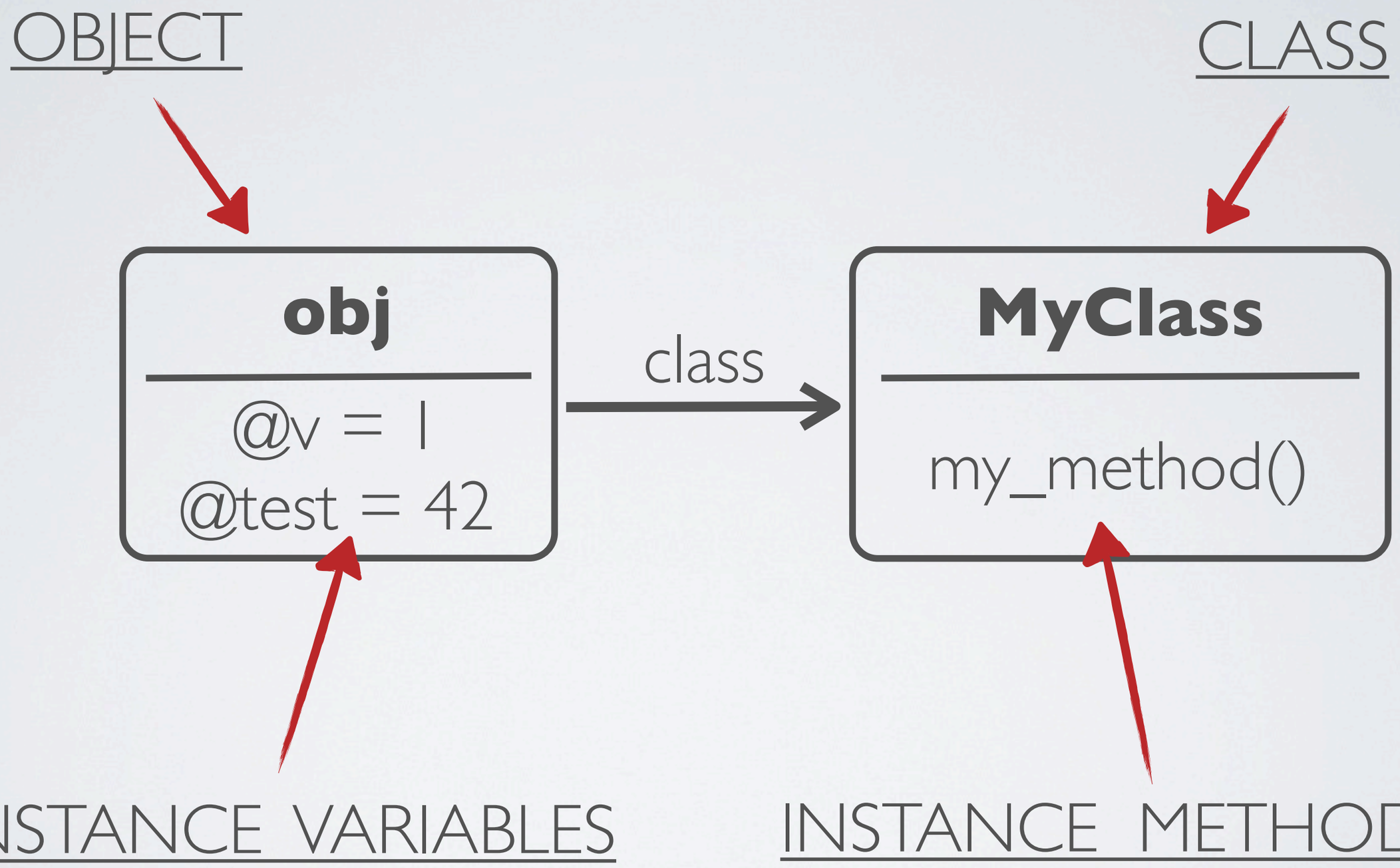
```ruby
obj.instance_variables
# => []

obj.my_method

obj.instance_variables
# => [:@v]

obj.instance_variable_set(:@test, 42)

obj.instance_variables
#  => [:@v, :@test]

obj.instance_variable_get(:@test)
# => 42
```


Ruby

Methoden

```ruby
obj.methods.grep(/my/)
# => [:my_method]

MyClass.methods == obj.methods
# => false

MyClass.instance_methods == obj.methods
# => true
```

Überblick

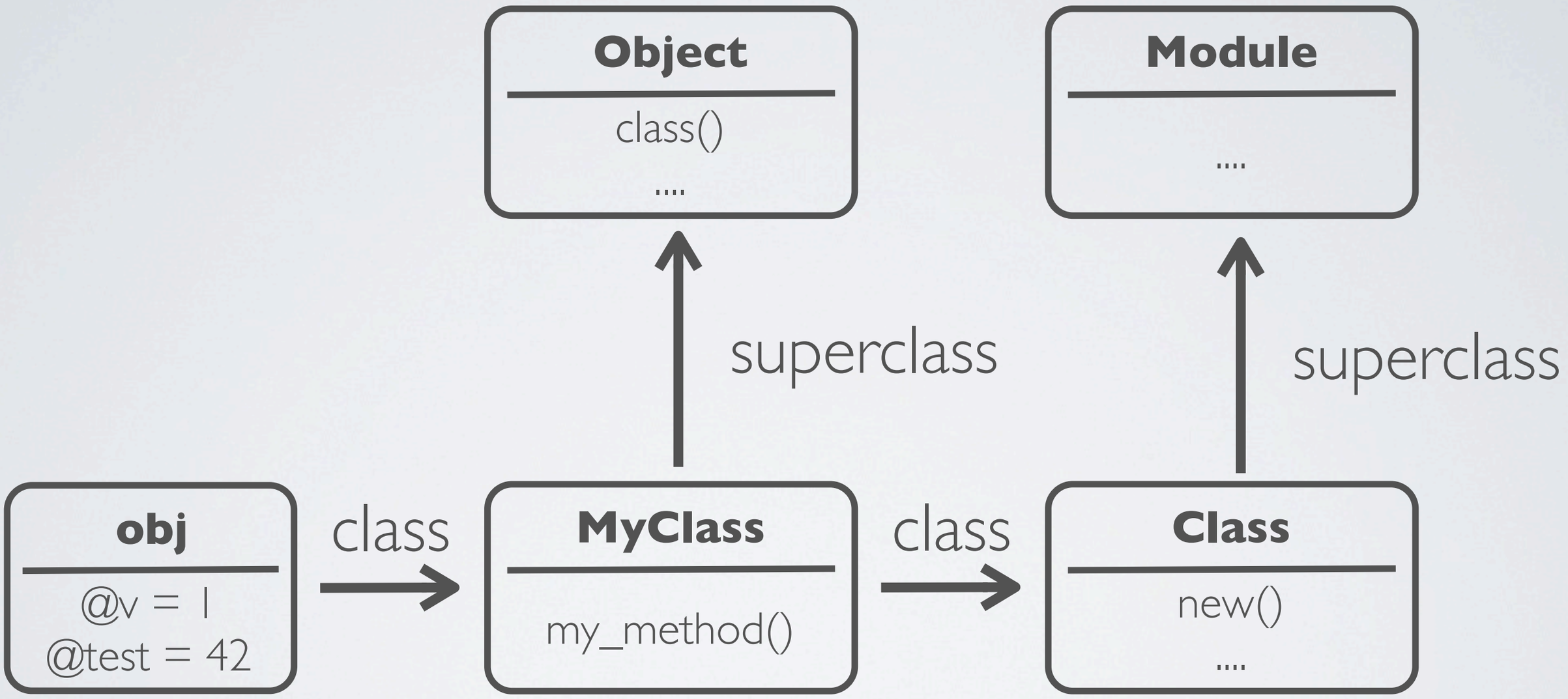

OBJECT
CLASS
obj
@v = 1
@test = 42
class
MyClass
my_method()
INSTANCE VARIABLES
INSTANCE METHODS

Klassenobjekte

```ruby
"hello".class # => String
String.class  # => Class

inherited = false
Class.instance_methods(inherited)
# => [:superclass, :allocate, :new]

String.superclass       # => Object
Object.superclass       # => BasicObject
BasicObject.superclass # => nil

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


Ruby

Überblick



Ruby

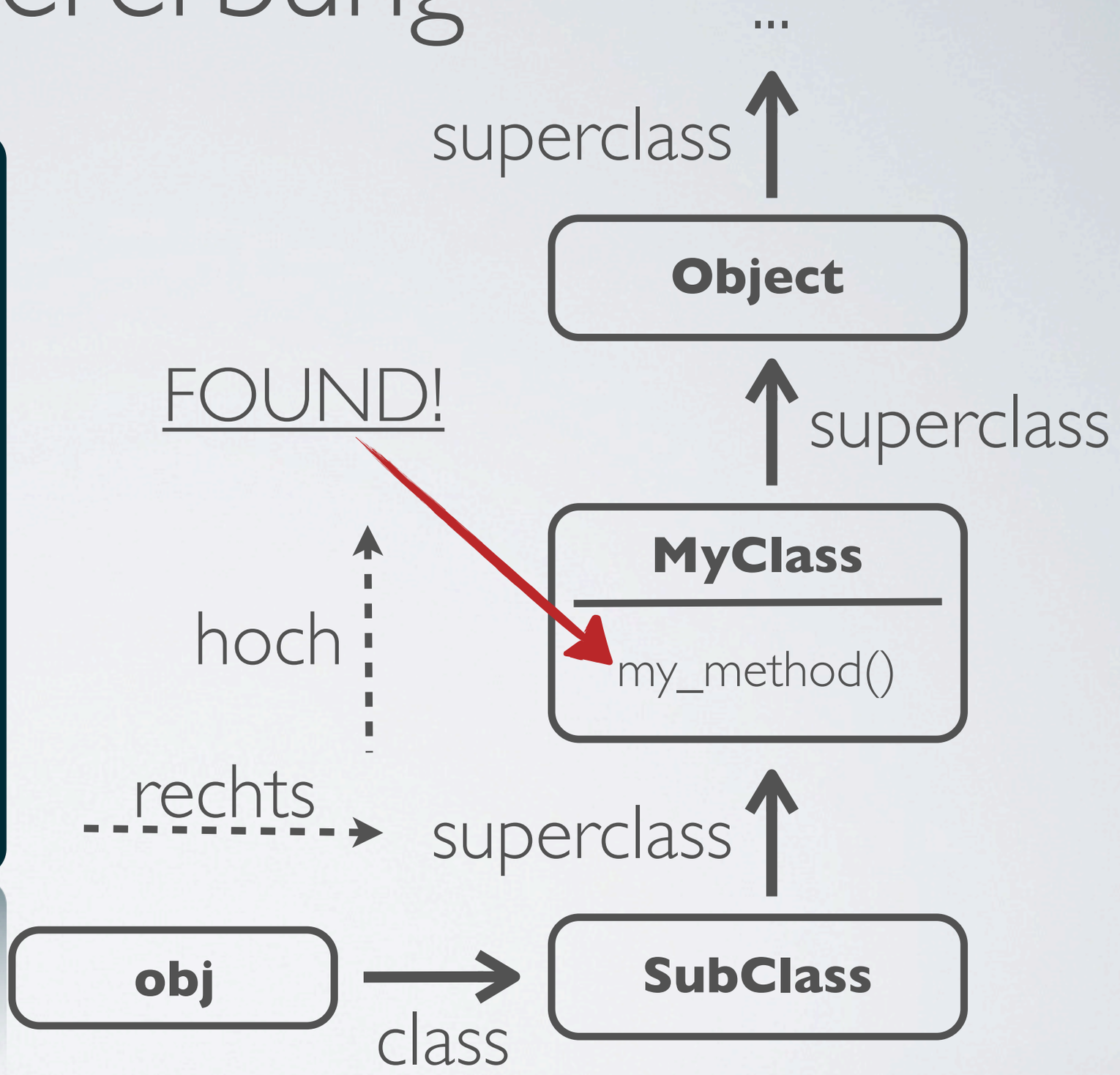
Method Lookup

Ruby

Object Model

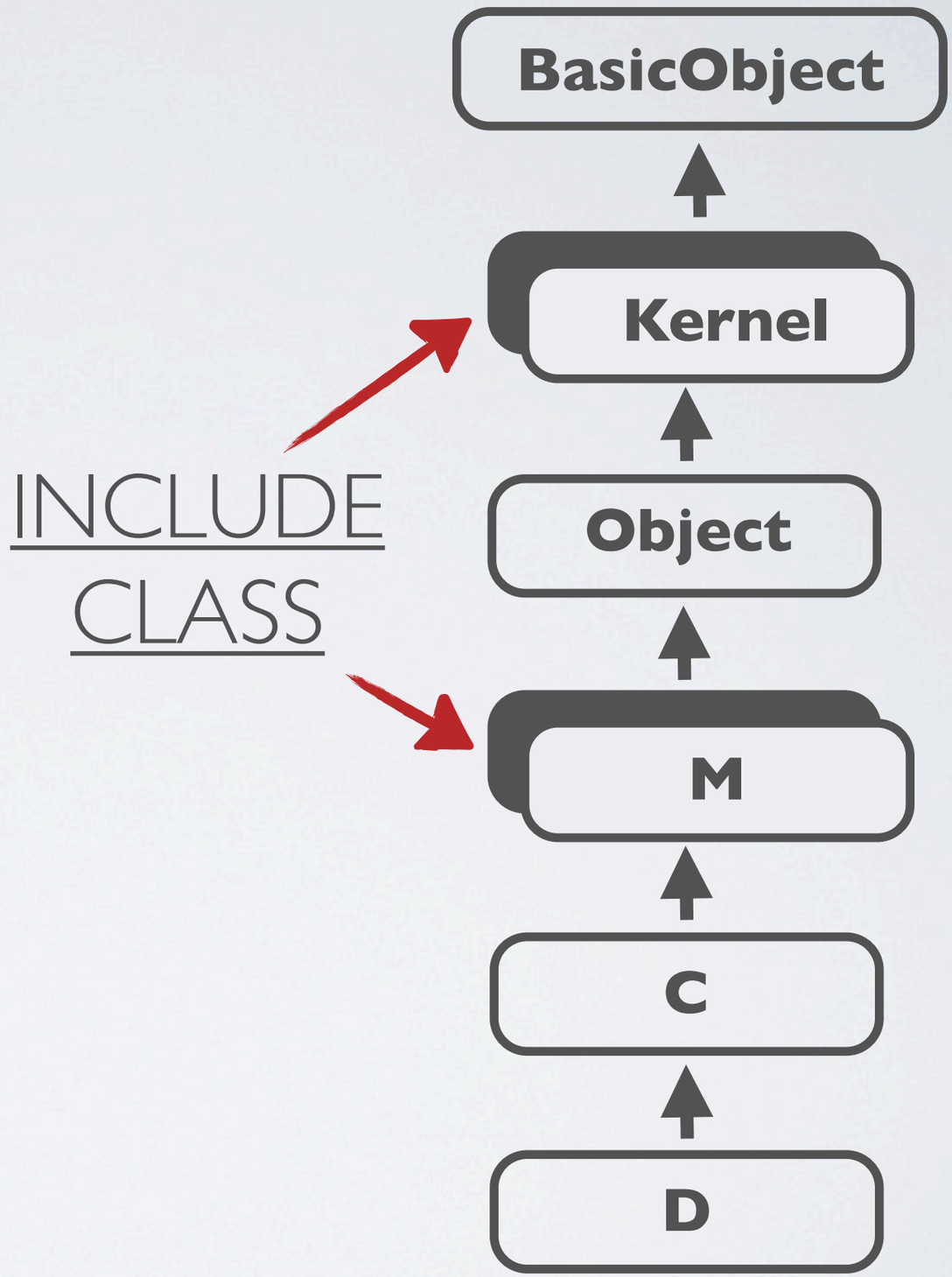
Vererbung

```ruby
class MyClass
  def my_method
    "my_method()"
  end
end

class SubClass < MyClass
end

obj = SubClass.new
obj.my_method
#  => "my_method()"
```



Mixins

```
module M
  def my_method
    "M#my_method()"
  end
end

class C
  include M
end

class D < C
end

D.new.my_method
# => "M#my_method"

D.ancestors
# => [ D, C, M,
#      Object, Kernel,
#      BasicObject ]
```



Self

Ruby

Object Model

Self

- alles wird immer im Kontext eines Objektes ausgewertet
  - *current object - self*
- zu jeder Zeit nimmt exakt ein Objekt die Rolle von self ein
- bei Methodenaufrufen wird der Empfänger zu self
  - Zugriff auf Instanzvariablen erfolgt immer auf die Variablen des aktuellen self
- Methodenaufrufe ohne expliziten Empfänger verwenden self als Empfänger

Ruby

Instanzen

```
class MyClass
  def testing_self
    @var = 10
    my_method
    self
  end

  def my_method
    @var = @var + 1
  end
end

obj = MyClass.new
obj.testing_self
# => #<MyClass:0x7fb0
#        @var = 11>
```

Aufruf an obj
=> self = obj

impliziter Empfänger:
self = obj

aufgerufen mit
self = obj

Zugriff auf aktuelles
Objekt (obj)

Main und Klassendefinitionen

```ruby
self          # => main
self.class # => Object

class MyClass
  self #  => MyClass
end

3.times do
  class MyClass
    puts self
  end
end

# MyClass
# MyClass
# MyClass
```

automatisch erzeugtes Main-Objekt

Klassendefinitionen ändern self

Klassendefinitonen sind gewöhnlicher Code

Testfragen

```ruby
module MyModule
  def my_method
    @var = 73
  end
end

class MyClass
  include MyModule
  attr_reader :var

  def initialize
    @var = 42
  end
end

mc = MyClass.new
mc.my_method

puts mc.var
```

Wer ist *self?*

Wo wird *MyModule* in Kette der Vorfahren platziert?

Wer ist *self?*

Ergebnis?
=> **73**

private und protected

Ruby

Object Model

Private

Aufruf nur mit implizitem Empfänger

```ruby
class MyClass
  def public_method
    self.private_method
  end

  private

  def private_method
    puts "private_method"
  end
end

MyClass.new.public_method
```

kein expliziter Empfänger erlaubt

```
NoMethodError: private method
'private_method' called [...]
```

Ruby

Private und Vererbung

```
class A
  private
  def my_method
    puts "my_method"
  end
end

class B < A
  def public_method
    my_method
  end
end

b = B.new
b.my_method

b.public_method
```

NoMethodError,...
(kein expliziter
Empfänger erlaubt)

"my_method"
(Empfänger ist implizit,
auch private Methoden
werden vererbt)

Ruby

Private und andere Instanzen

```ruby
class A
  def public_method
    a = A.new
    a.my_method
  end

  private
  def my_method
    puts "my_method"
  end
end

A.new.public_method
```

NoMethodError,...
(kein expliziter Empfänger erlaubt)

Protected

Expliziter Empfänger erlaubt, wenn gleiche Klasse wie *self*

```ruby
class A
  def public_method
    self.protected_method
    A.new.protected_method
  end

  protected
  def protected_method
    "protected_method"
  end
end

class B < A; end

B.new.public_method
B.new.protected_method
```

Unterklassen werden berücksichtigt

self.kind_of? self.class
=> true

self.kind_of? A
=> true

self.kind_of? B
=> false
=> NoMethodError

Ruby

Zusammenfassung

- ein Objekt besteht aus einer Menge von Instanzvariablen und einem Verweis zu seiner Klasse, welche die Instanzmethoden beinhaltet
- Klassen sind Instanzen der Klasse *Class*, die von *Module* erbt
- Module sind grundsätzlich nur eine Menge gruppierter Methoden
- jede Klasse hat eine Kette von Vorfahren, die mit ihr selbst beginnt und bei *BasicObject* endet
- Methoden werden gefunden, indem *rechts* in die Klasse geht und dann die Kette der Vorfahren *nach oben* durchsucht wird, bis die Methode gefunden wurde, oder man ans Ende gelangt

Zusammenfassung

- inkludierte Module werden oberhalb der eigenen Klasse in die Kette der Vorfahren eingefügt
- alles wird immer im Kontext eines durch *self* referenzierten, aktuellen Objektes ausgewertet
- bei Methodenaufrufen wird der Empfänger zu *self*
- bei Klassen- und Moduldefinition ist die Klasse/das Modul *self*
- Instanzvariablen beziehen sich immer auf *self*
- Methodenaufrufe ohne expliziten Empfänger werden immer an *self* gesendet

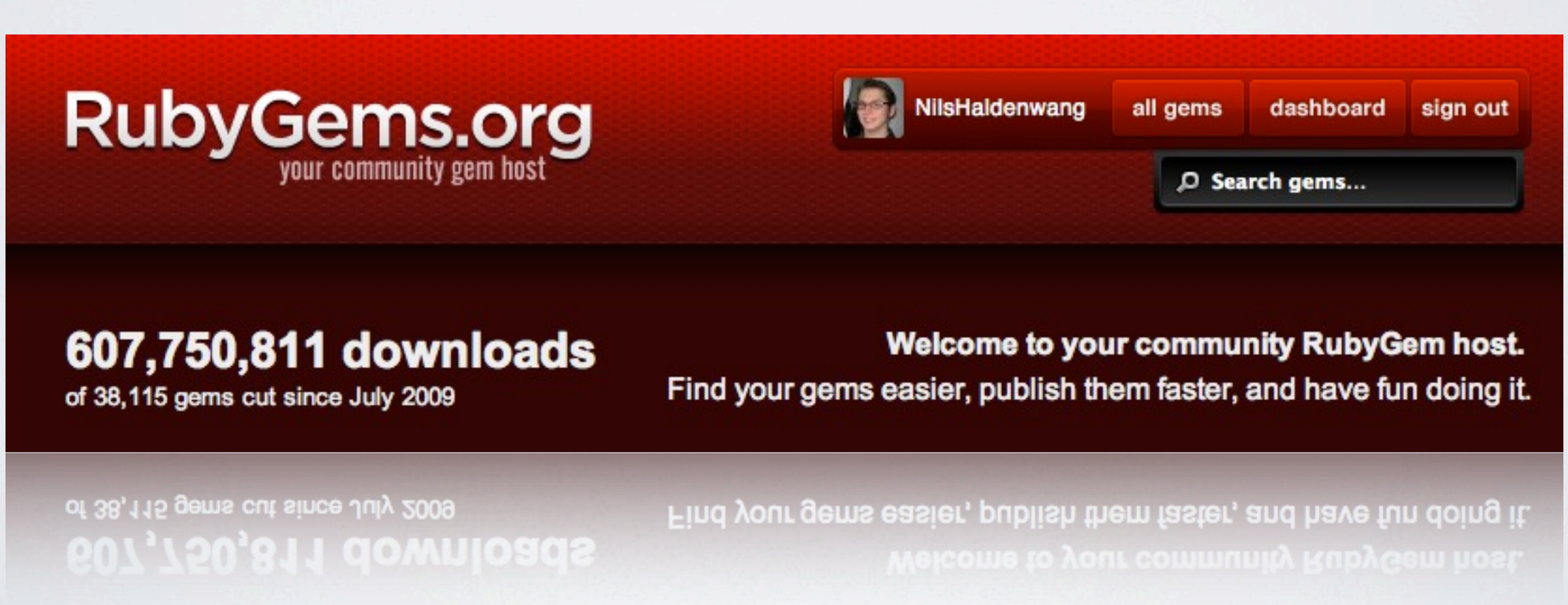
Ruby

RubyGems

RubyGems

https://rubygems.org/

"There is a gem for that."



Beispiel: Pry

http://pry.github.com/

Beispielkonfiguration:

https://github.com/skwp/dotfiles/tree/master/irb

Demo