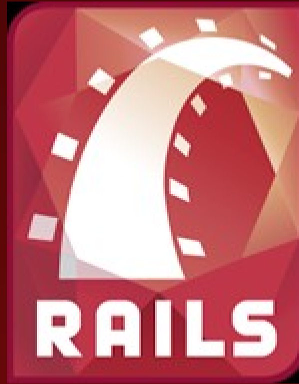


Fenomen Ruby'ego

Marek Kirejczyk

Część 1
Ruby

Sukces

Rzut okiem na historię...

- 1995 Yukihiro Matsumoto
  Ruby

- 2004 David Heinemeier Hansson
  Ruby on Rails
  (wydzielone z aplikacji BaseCamp)
  Stanie się KillerApp dla Ruby'iego

...sukcesu...

- 2005
  wg. statystyk księgarni Amazon
  w kategorii programowanie
  pierwsze dwa miejsca zajmują książki o
  Ruby i RoR
- Sierpień '2006
  Apple decyduje się dołączyć RoR do
  MacOS X Leopard

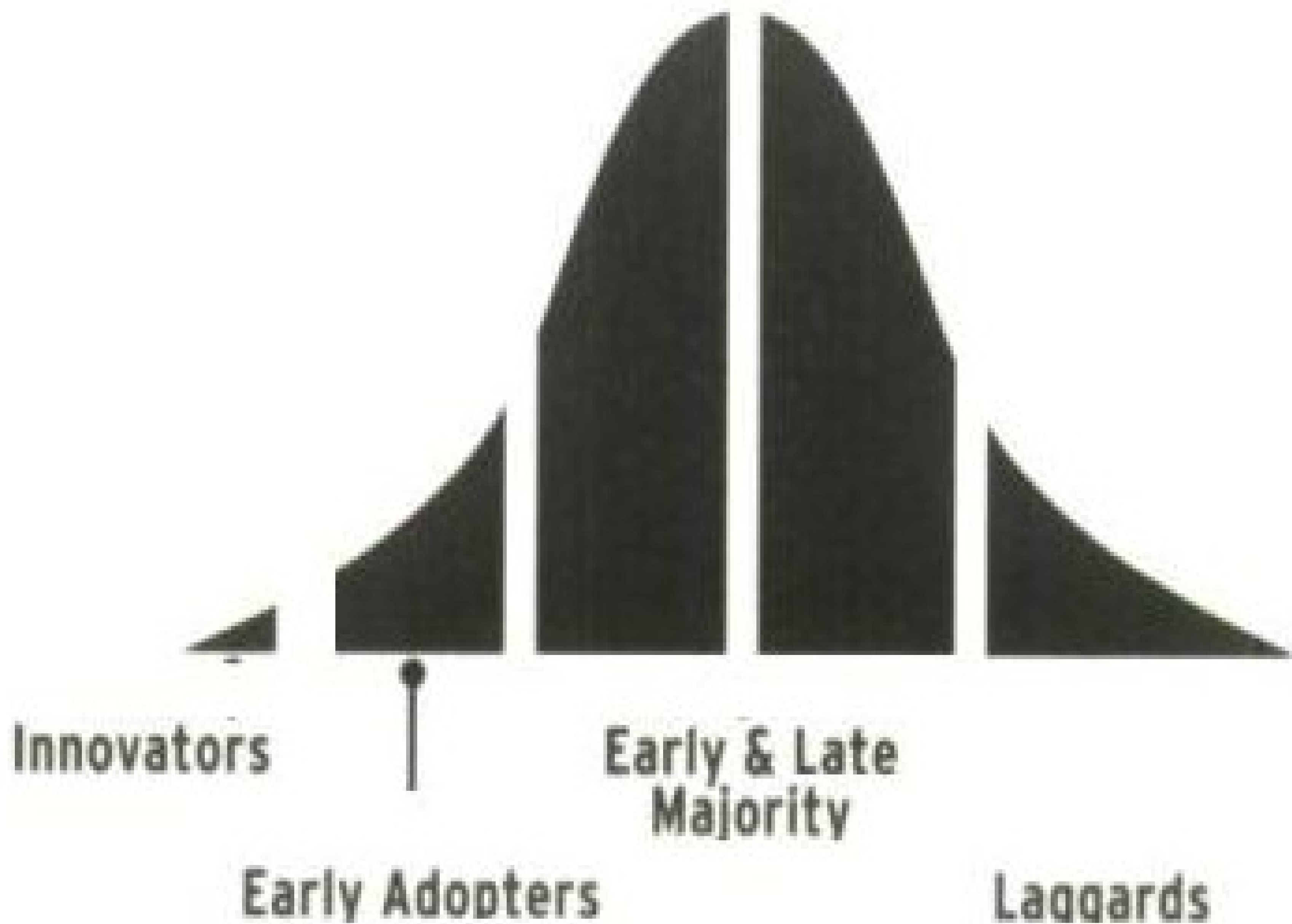
...który dopiero się zaczyna

- 2006
  Computerworld magazine
  Pierwsze miejsce w
  "The Top Five Technologies You Need to Know About in '07"
- 2007
  Powstaje Groovy on Grails 1.0 (JSR 241) i setki innych języków i frameworków
  - inspirowany elastycznością RoR

Źródło sukcesu?

Purpule Cow

- Nowa teoria marketingu
- Stworzona przez Setha Godina
- Koniec epoki massmediów
- Produkt jest niezwykły, wyróżniający się



Innovators
Early Adopters
Early & Late
Majority
Laggards

Fioletowy rubin?

- Niezwykła wygoda programisty
- Krótki kod = czytelny kod
- Tony lukru składniowego
- Czyta obiektowość
- Wysoki poziom abstrakcji i elastyczności
- Tworzenie języka zbliżonego do dziedziny
- RoR

Ruby i RoR jako nośnik ideologii

- Purple Cow
- Start-Up
- Web 2.0
- Human User Interface Design
- OpenId

- Google, Apple – podobne ideologie

Zagadka

Ile to jest 3^{300} ?

Język Ruby

Stare ludowe przysłowie

"If it walks like a duck and quacks like a duck, it must be a duck"

Składnia

- Entery znaczące, wcięcia nie
- Ostatnie wyrażenie to wynik metody
- Stringi
  - proste 'x'
  - złożone "x"

```
def say(msg)
    "Hello, #{msg} "
end
```

Konwencje nazewnicze

- lokalne
- parametry
- $globalne
- @instancyjne
- @@klasowe
- STALE
- NazwyKlas
- Moduły

Niektóre typy wbudowane

- FixNum i BigNum
  - Automatycznie podmieniane
- String
- Zakresy
- Wyrażenia regularne
- Symbole

```
num = 81
	6.times do
	puts "#{num.class}: #{num}"
	num *= num
end
```

Fixnum: 81
Fixnum: 6561
Fixnum: 43046721
Bignum: 1853020188851841
Bignum: 3433683820292512484657849089281
Bignum: 11790184577738583171520872861412518665678211592275841109096961

Tablice

- Wbudowane w język
- Ładne inicjalizacje
- Skrócony zapis inicjalizacji
- Nil to też obiekt

- a = [1, 'cat', nil]
- b = ['a', 'b', 'c']
- a = %w{a b c}
- a[0]
- nil - obiekt pusty

Hashe

- Wbudowane w język słowniki
- Składnia a'la PHP

```
inst_section = {
'clarinet' => 'woodwind',
'drum' => 'percussion',
'oboe' => 'woodwind',
}
inst_section[clarinet] ->
    'woodwind'
```

Instrukcja warunkowa

- puts "Danger, Will Robinson" if radiation > 3000

```
if count > 10
    puts "Try again"
elseif tries == 3
    puts "You lose"
else
    puts "Enter a number"
end
```

Pętle

```
square = 2
square = square*square
    while square < 1000
```

```
square = 2
while square < 1000
    square = square*square
end
```

Wyrażenia regularne

- Część języka
- Są obiektem
- Operator =~
- Metody w bibliotece standardowej

```
if line =~ /\d\d:\d\d:\d\d/
    puts "Hour: #{line}"
end
```

Zakresy

- Sekwencje
  - iterowanie
  - zawieranie
- Przedziały
  - Operator ===
- Wyrażenia warunkowe
  - Zwraca true w zadanym przedziale

```
1..10
'a'..'z'
my_array = [ 1, 2, 3 ]
0...my_array.length

(1..10) === 5

while line = gets
    puts line if line =~ /start/ ..
      line =~ /end/
end
```

Domknięcia

Domknięcia

- Metodzie można przekazać argument-blok
- Instrukcja yield

```
def call_block
    puts "Start of method"
    yield
    yield
    puts "End of method"
end

call_block { puts "In the
    block" }
```

Parametryzowanie domknięć

- Bardzo elegancka konkurencja klas anonimowych i delegatów
- Rozwiązanie wprowadzone do C# 3.0, planowane w Javie 7.0

```
def call_block
    yield("hello", 99)
end

call_block {|str, num| ...}
```

Propaganda
Akcje i User Interface

```
class MyButton < Button
    def initialize(label, &action)
        super(label)
        @action = action
    end
    def button_pressed
        @action.call(self)
    end
end

start_button = JukeboxButton.new("Start") { songlist.start }
pause_button = JukeboxButton.new("Pause") { songlist.pause }
```

- Dużo krócej niż w innych popularnych językach ;)

Iteracje

- Metoda each definiuje iterator

```
animals = %w( ant bee cat dog elk )
animals.each {|animal| puts animal }
[ 'cat', 'dog', 'horse' ].each {|name| print name, " " }
('a'..'e').each {|char| print char }
```

- Można oczywiście definiować inne „iteratory”

```
5.times { print "*" }
3.upto(6) {|i| print i }
```

Lambda

- Konwersja bloku na metodę:

```
def n_times(thing)
    return lambda {|n| thing * n }
end

p1 = n_times(23)
p1.call(3)              → 69
p1.call(4)              → 92
p2 = n_times("Hello ")
p2.call(3)              → "Hello Hello Hello "
```

Funkcje wyższych rzędów

```
def succ()
	return lambda {|x| x + 1 }
end

def compose(x, y)
	return lambda {|a| y.call(x.call(a)) }
end

puts compose(succ,succ).call(0)			2
```

Wyjątki

- Klasyczne rozwiązanie z pewnymi rozszerzeniami
- Trzy wersje raise
  - propaguje bierzący, lub RuntimeError
  - RuntimeError
  - nazwa klasy, msg i stacktrace (metoda Kernel.caller)

```
f = File.open("testfile")
begin
    # .. process
rescue
    # .. handle error
else
    puts „No errors;)"
ensure
    f.close unless f.nil?
end
```

```
raise
raise "bad mp3 encoding"
raise InterfaceException, "Keyboard
    failure", caller
```

Catch i throw

```
def routine(n)
    puts n
    throw :done if n <= 0
    routine(n1)
end
catch(:done) { routine(4) }

4, 3, 2, 1, 0
```

- Więcej niż goto
- Catch zwraca to co throw
- Element biblioteki,
nie języka

Obiekty

Model obiektu w Rubym

- Każdy obiekt posiada unikalny id
- Metoda inspekt - zwraca string z nazwą obiektu, id i zawartością
- Metod to_s - odpowiednik toString z Javy

```
song =
    Song.new("Bicylops",
    "Fleck", 260)
song.inspect !

#<Song:0x1c7ca8
    @name="Bicylops",
    @duration=260,
    @artist="Fleck">
```

Konstruktuory i inicjalizacja

- Statyczna metoda ClassName.new() – domyślny konstruktor – metoda statyczna, z dowolną liczbą parametrów, uzależnioną od initialize
- Instancyjna metoda initialize(), wywoływana przez konstruktor domyślny, z ustaloną liczbą parametrów
- Brak przeładowywania konstuktorów

Przykład initialize

```
class Song
    def initialize(name, artist, duration)
        @name = name
        @artist = artist
        @duration = duration
    end
end
Song.new("Hit me babe one more time", "Elthon
    John", 33)
```

Dziedziczenie i super w konstruktorach

```
class KaraokeSong < Song
  def initialize(name, artist, duration, lyrics)
      super(name, artist, duration)
      @lyrics = lyrics
  end
end
```

Super w metodach

```
def to_s
    super + " [#@lyrics]"
end
```

- Alias na metodę nie obiekt, jak w SmallTalku czy Javie

Aliasy

- Aliasy pozwalają tworzyć zdublowane nazwy na metody (nowe referencje na metody)
- Częściowe definicje klasy – klasa składa się z sumy wszystkich (załadowanych) definicji
- Aliasy są, przydatne jeśli chcemy przedefiniować znaczenie metody używając starej

```
class Fixnum
    alias old_plus +
    def +(other)
        old_plus(other).succ
    end
end
2 + 2 =  5
```

Akcesory

```
class Song
    ...
    def name
        @name
    end

    def duration=(new_duration)
        @duration = new_duration
    end

    attr_reader :artist, :duration
    attr_writer :artist, :name
end
```

```
song = Song.new("Bicylops", "Fleck",
    260)

song.artist -> "Fleck"
song.name -> "Bicylops"
song.duration -> 260
song.duration = 257
```

Operatory i przeładowanie

- W rubym można przeładowywać operatory
- Indexery: [] i []=
- Operatory porównywania <=>, <=, >=, ==, <, >
- Operatory arytmetyczne +,-,*,/ i automatyczne ich odpowiedniki += ....
- Podobnie ==, =~ i !=, !~
- Operator `` - wykonania polecenia powłoki

```
class SongList
    def [](index)
        @songs[index]
    end
end
```

Specyfikatory dostępu

```
class MyClass
    #To by była publiczna metoda, ale
    #będzie prywatna...
    def method1
    end

    protected
    def method2
    end

    #...o  czym zadecydowała ta linijka
    private :method1
end
```

- Domyślny dostęp publiczny
- Słowa kluczowe protected, public, private z klasycznym znaczeniem c++
- Można osobno dospecyfikować dostęp do metody

Moduły jako namespace'y

```
module Trig
    PI = 3.141592654
    def Trig.sin(x)
        # ..
    end
    def Trig.cos(x)
    end
end
```

```
module Moral
    VERY_BAD = 0
    BAD = 1
    def Moral.sin(badness)
        # ...
    end
end
```

```
require 'trig'
require 'moral'
y = Trig.sin(Trig::PI/4)
wrongdoing = Moral.sin(Moral::VERY_BAD)
```

Moduły i mixiny

- Moduły można włączać do klas
- Inne podejście do problemu wielodziedziczenia
- Modułu mogą zawierać metody
- Nie mają stanu, korzystają ze zmiennych mixinu
- Przykłady:
  - Comparable - definiuje pozostałe operatory porównywania, na podstawie operatora <=>
  - Enumerable - generuje map, include?, find_all?, inject, na podstawie each
- Zagadka: W jakim popularnym języku występują mixiny?

Moduły – przykład uzycia

```
class Song
    include Comparable
    def initialize(name, artist, duration)
        @name = name
        @artist = artist
        @duration = duration
    end
    def <=>(other)
        self.duration <=> other.duration
    end
end
```

```
song1 = Song.new("My Way",
    "Sinatra", 225)
song2 = Song.new("Bicylops",
    "Fleck", 260)
```

song1 <=> song2	1
song1 < song2	true
song1 == song1	true
song1 > song2	false

Etc...

Przykład - Singleton

```
class MyLogger

    private_class_method :new

    @@logger = nil

    def MyLogger.create
            @@logger = new unless

    @@logger
            @@logger
    end
ends
```

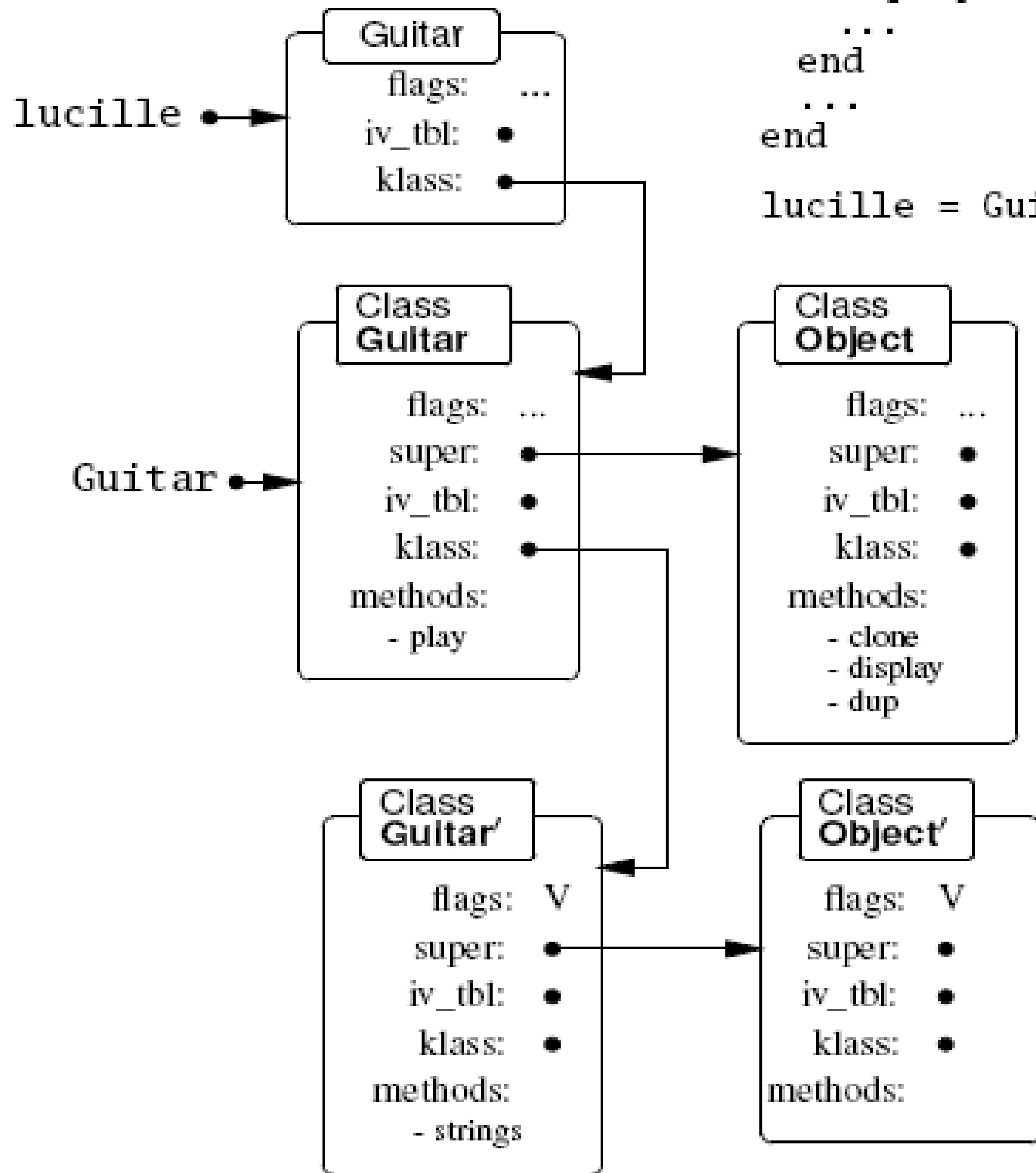
- Prywatny konstruktor
- Zmienna instancyjna trzymająca jedyna kopie obiektu
- Statyczna metoda dostępowa

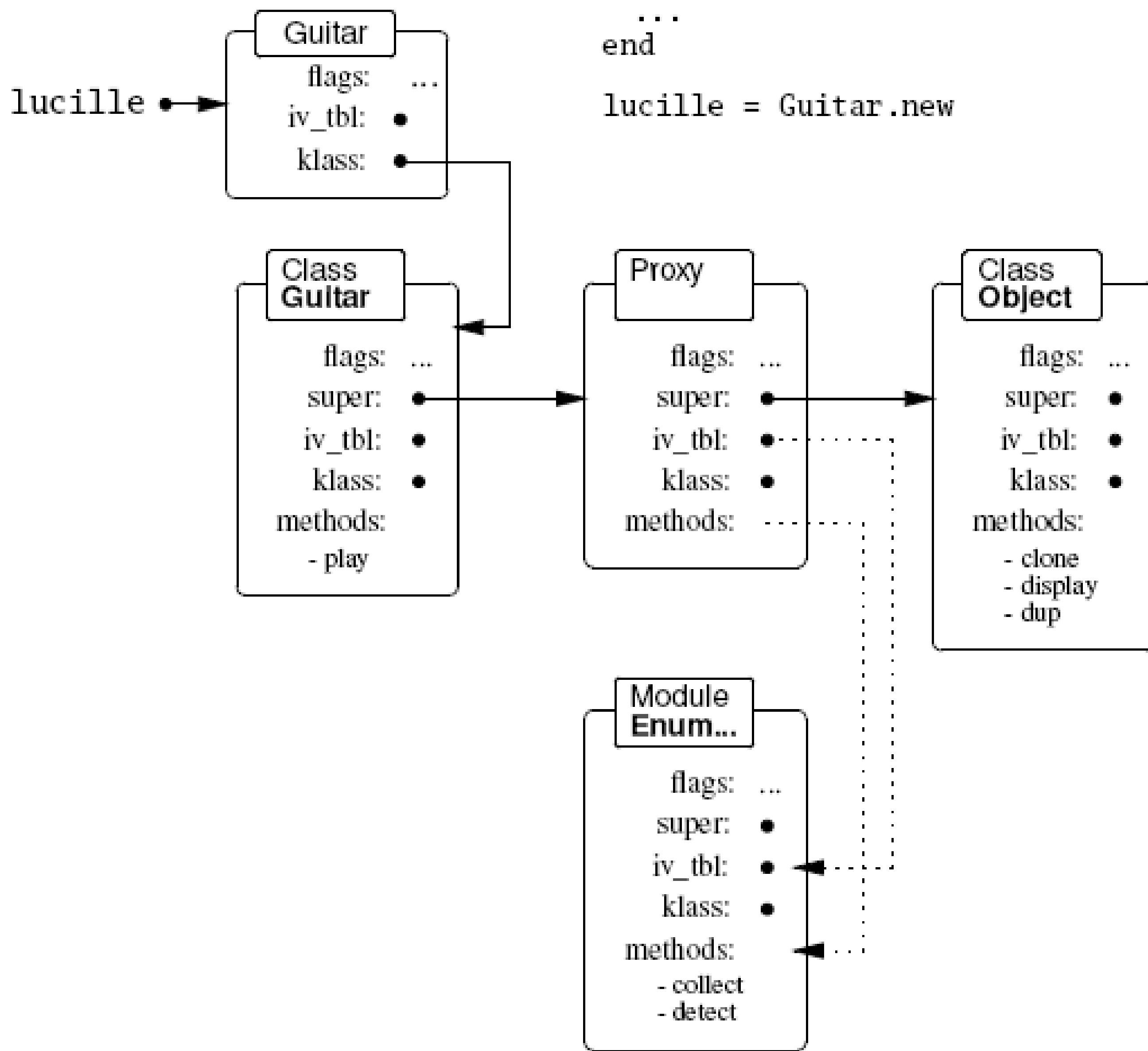
Przykład – Singleton Moduł

```
module MyLogger

    private_class_method :new

    @@logger = nil

    def MyLogger.create
            @@logger = new unless

    @@logger
            @@logger
    end
ends
```

- Zenkapsułowane pojęcie
- Tylko jedna zmiana
- Nie do wykonania w C++, C#, Java

```
class Guitar
  def Guitar.strings()
    return 6
  end
  def play()
    ...
  end
  ...
end

lucille = Guitar.new
```




class Guitar
include Enumerable
def play()
...
end
...
end
lucille = Guitar.new
lucille
Guitar
flags: ...
iv_tbl:
klass:
Class
Guitar
flags: ...
super:
iv_tbl:
klass:
methods:
- play
Proxy
flags: ...
super:
iv_tbl:
klass:
methods:
Class
Object
flags: ...
super:
iv_tbl:
klass:
methods:
- clone
- display
- dup
Module
Enum...
flags: ...
super:
iv_tbl:
klass:
methods:
- collect
- detect

Metaprogramowanie

Metaprogamowanie

– Dynamizm i refleksja
– Otwartość na zmiany
– Brak makr, naturalna składnia
– Domknięcia pozwalają na budownie nowych instrukcji sterujących (np.: throw, catch)
– Kierunek: DSL

attr_reader

```
class Module
    def attr_reader (*syms)
        syms.each do { |sym|
        class_eval %{def #{sym}
                    @#{sym}
                end}
        }
    end
end
```

Refleksja

- Każdy obiekt posiada składowe
  - methods – kolekcja obiektów „odbijających” metody
  - responds_to? – sprawdza czy istnieje metoda
  - kind_of – czy jest nadklasą argumentu
  - instance_of – czy jest instancją argumentu
  - klass – reprezentuje bieżącą klasę
  - klass.superclass – reprezentuje nadklasę
  - klass.ancestors – lista „odbić” modułów i nadklasy

Refleksja C.D.

- Odpytywanie ze względu na typy:
  - private_instance_methods
  - protected_instance_methods
  - public_instance_methods
  - singleton_methods
  - class_variables
  - constants

Dynamiczne wołanie metod

- Metoda send

```
"John Coltrane".send(:length)
```

- Obiekt Method

```
trane = "John Coltrane".method(:length)
miles = "Miles Davis".method("sub")
trane.call                          13
miles.call(/iles/, '.')             "M. Davis"
```

- Metoda eval

```
val = 13
eval("val")
```

Bindownie

- Bindownie metod

```
unbound_length = String.instance_method(:length)
class String
    def length
        99
    end
end

str = "cat"
str.length                                      99
bound_length = unbound_length.bind(str)
bound_length.call                               3
```

System Hooks

- Technika pozwalająca łapać zdarzenia rubiego

```
class Object
    attr_accessor :timestamp
end

class Class
    alias_method :old_new, :new
    def new(*args)
      result = old_new(*args)
      result.timestamp = Time.now
      result
    end
end
```

ObjectSpace

- Daje dostęp do GC
- ObjectSpace.each_object(Numeric) {|x| p x }
  - Enumeruje po wszystkich obiektach Numeric
- Daje możliwość definiowania finalizerów
- garbage_collect
- _id2ref

Kontynuacje

```
def strange
    callcc {|continuation|
        return continuation}
    print "Back in method, "
end

print "Before method. "
continuation = strange()
print "After method. "
continuation.call if continuation
```

Before method. After method. Back in method, After method.

- Definiuje rozwidlenie w programie
- Callcc wykonuje kod z bloku, po czym kontynuuje instrukcje następujące
- Argumentem bloku jest zapisana kontynuacja (save game)
- Metoda call kontynuacji powraca do momentu, za blokiem
- Kontynuacja trzyma stan wątku

Serwery kontynuacje

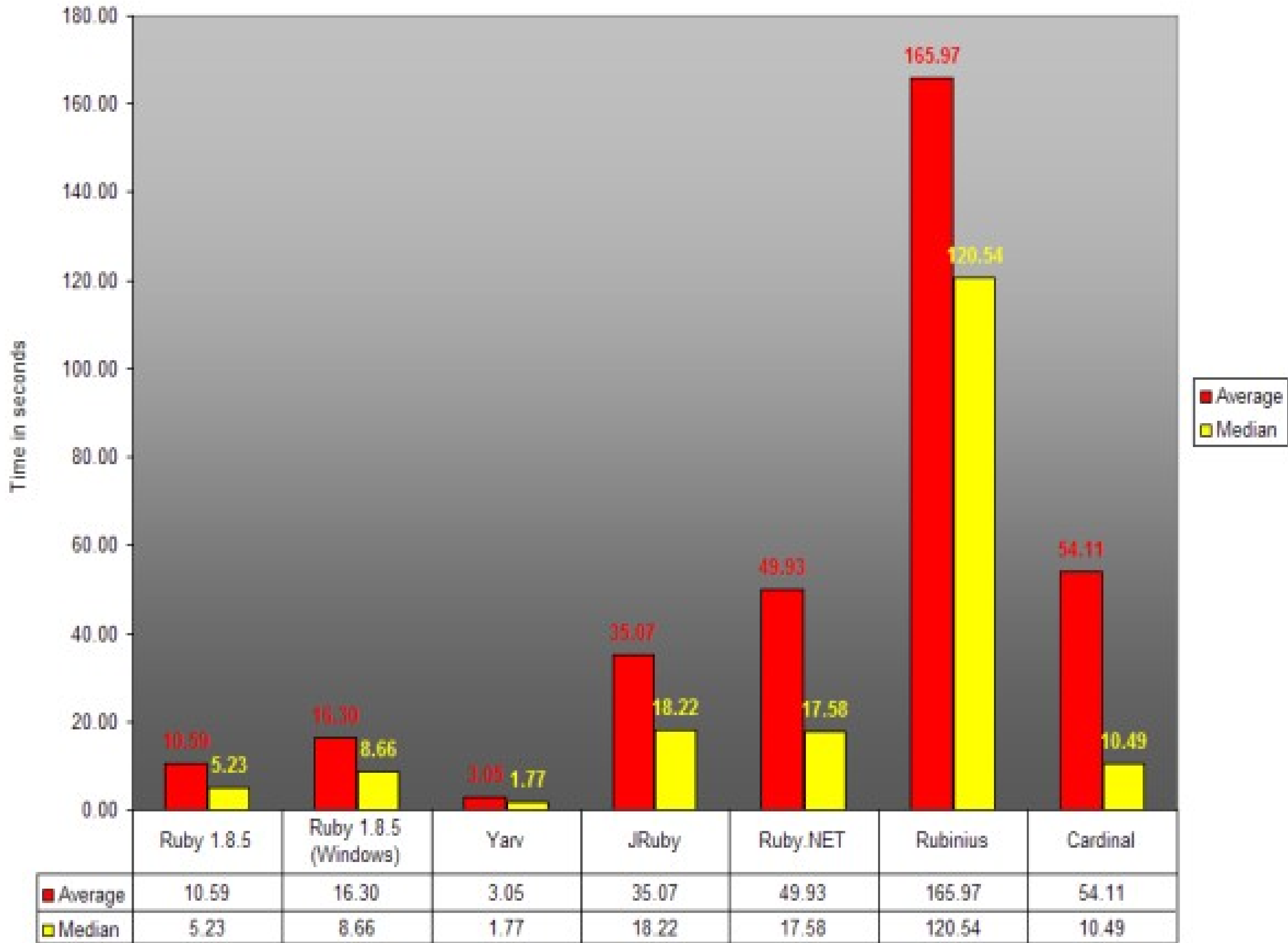
- Nie RoR
- SmallTalk SeaSide
- Koniec z sesją pisaniem, bez stanowym, podziałem między requestami
- GUI lokalne i WWW pisane identycznie
- Nowe możliwości dla przycisków wstecz i w przód

Biblioteka standardowa

- Wsparcie dla wątków z poziomu języka
- Marshaling (serializacja)
- Mechanizm poziomów bezpieczeństwa
- Server WebRick
- Framework testujący
- Zip, xml, rpc, SOAP
- Rinda
- Inne

Ruby - podsumowanie

- Wydaje się, że to kolejny krok po Java, C#
- Odwrót od statycznego typowania
    - Trade-off: elastyczność vs. werfikacja w czasie kompilacji
- Enkapsułowanie bardziej złożonych idei, niż dane w obiektach
- Odciążenie programisty, skrócenie czasu pracy
- Tworzenie języków zbliżonych do dziedziny



	Ruby 1.8.5	Ruby 1.8.5 (Windows)	Yarv	JRuby	Ruby.NET	Rubinius	Cardinal
Average	10.59	16.30	3.05	35.07	49.93	165.97	54.11
Median	5.23	8.66	1.77	18.22	17.58	120.54	10.49

Czemu ruby przetrwa?

- Jest świetnym językiem DSL
- Java i C# natrafią na duże bariery już nie długo, jako języki ogólnego zastosowania:
  - Staną się zbyt skomplikowane
  - Nie poradzą sobie z przejściem na massive multicore

Czy słabe typowanie jest słabe?

- Wtyka do Idei
- Wtyka do NetBeansa
- Zintegrowane testy jednostkowe

Inne znane rozwiązania oparte o ruby

- Gems – menadżer pakietów rubiego
- raven – „Budowanie javy za pomoca rubiego”
- JRuby - Ruby na Virtualnej Maszynie Javy, współpraca z bibliotekami Javy
- RadRails - przerobione środowisko eclipse (więcej niż wtyka;) dla rubiego

Literatura

- „Beyond Java” - Bruce Tate O'REILLY
  - „Więcej niż Java” – Helion 2005
- „Programming Ruby - The Pragmatic Programmers's Guide” - Dave Thomas
- http://tryruby.hobix.com/
  - Online live tutorial