The future of Ruby is faster

Euruko 2013

Dirkjan Bussink

@dbussink

Rubinius

Use Ruby


TM
Engine
yard

“Everything you heard about Ruby being slow is not true. It’s about twice as slow as that”

The lure of the hash table

```ruby
class MethodTable

  def initialize
    @methods = {}
  end

  def store_method(name, code)
    @methods[name] = code
  end

  def find_method(name)
    @methods[name]
  end

end
```

```ruby
class Address
  def initialize
    @instance_variables[:street] = "Pantheon"
    @instance_variables[:number] = 1
    @instance_variables[:city]   = "Athens"
  end
end
```

Hash tables everywhere!

Inline caching

```
p = Person.new
```

```
p.name
```

```
p = Person.new
p.name
```

<receiver_class_id, code>

"There are 2 hard problems in computer science: caching, naming, and off-by-1 errors"

Global serial number

https://charlie.bz/blog/things-that-clear-rubys-method-cache

http://jamesgolick.com/2013/4/14/mris-method-caches.html

Per class serial number

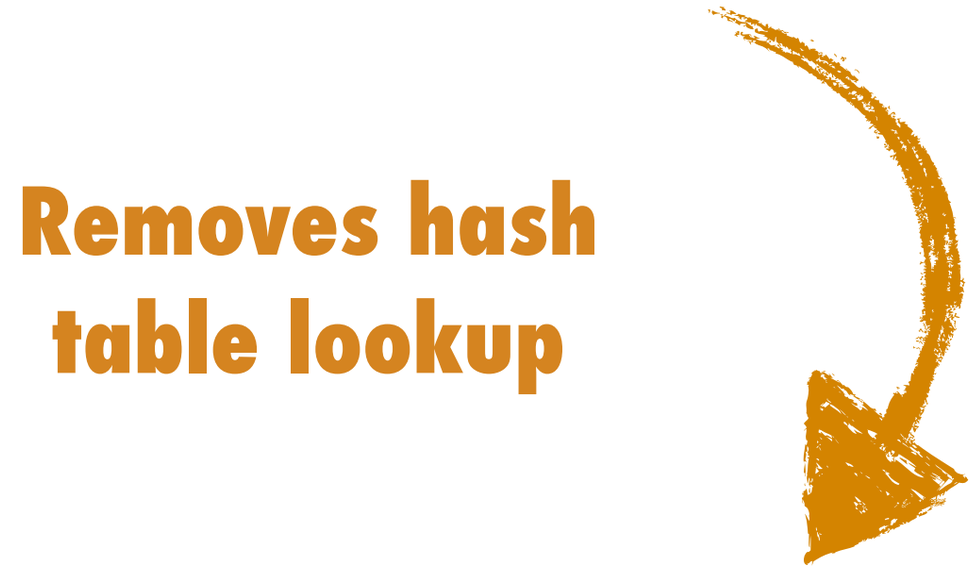
Probably in 2.1

Instance variable packing

```ruby
class Address
  def initialize
    @street = "Pantheon"
    @number = 1
    @city   = "Athens"
  end
end

Address.instance_variable_get("@seen_ivars")
=> [:@street, :@number, :@city]
```

```
0001: push_ivar :@number
```




```
self[1]
```

Just in time compilation

```ruby
def method1
  1 + method2
end

def method2
  2 + 1
end

100000.times do
  method1
end
```

```ruby
def method1
  1 + method2
end

def method2
  2 + 1
end

100000.times do
  method1
end
```

```
...
0x110ed90e7   mov $0x9, %eax
0x110ed90ec   jmp 0x119                    ; 0x110ed9129
...
0x110ed9129   addq $0xa0, %rsp
0x110ed9130   pop %rbp
0x110ed9131   ret
```

```ruby
def method1
  1 + method2
end

def method2
  2 + 1
end

100000.times do
  method1
end
```

```
...
0x110ed90e7   mov $0x9, %eax
0x110ed90ec   jmp 0x119                       ; 0x110ed9129
...
0x110ed9129   addq $0xa0, %rsp
0x110ed9130   pop %rbp
0x110ed9131   ret
```

b0100 (4)
b1000 (8)
b1001 (9)

```ruby
def method1
  1 + method2
end

def method2
  2 + 1
end

100000.times do
  method1
end
```

```
...
0x110ed90e7   mov $0x9, %eax
0x110ed90ec   jmp 0x119                    ; 0x110ed9129
...
0x110ed9129   addq $0xa0, %rsp
0x110ed9130   pop %rbp
0x110ed9131   ret
```

b0100 (4)
b1000 (8)
b1001 (9)


RenoNorge AS
i oppdrag for Stavanger Kommune


Waste
Less
Living

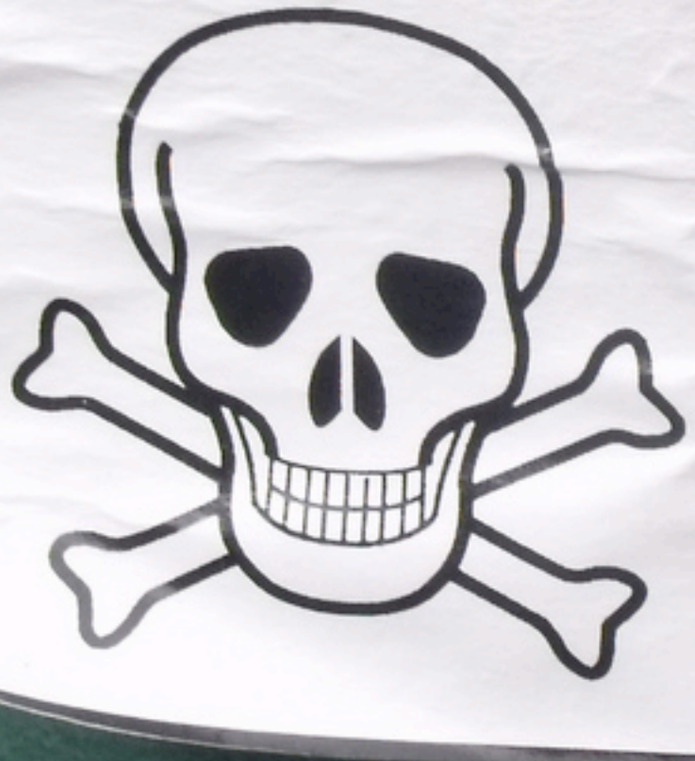
```ruby
a        = Address.new
a.street = "Pantheon"
a.number = "1"
a.city   = "Athens"
```

```ruby
Rubinius.memory_size(a) => 56
```

VS

```ruby
Rubinius.memory_size(a) => 160
```

WARNING! WARNING! WARNING! WARNING! WARNING!



Fast Workers Die Young!

Take a break

IWW - a Union for ALL Workers

Rubinius

Use Ruby™

Ruby

A Programmer's Best Friend

2.1

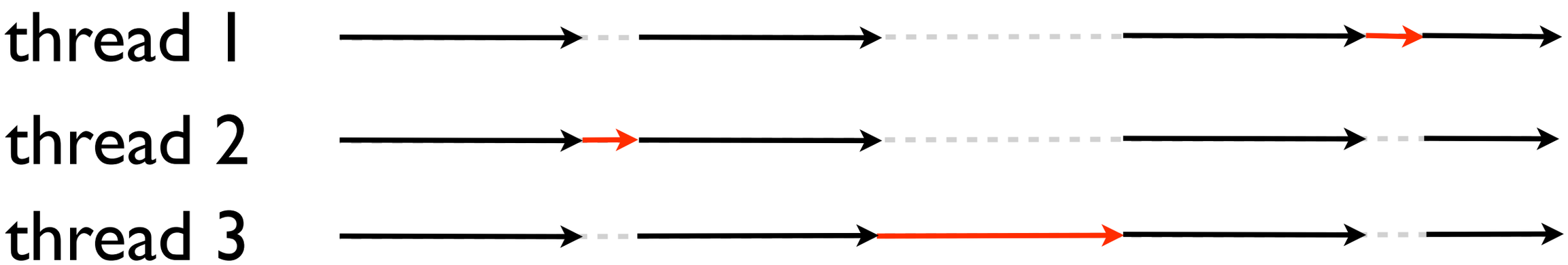
Auto tuning

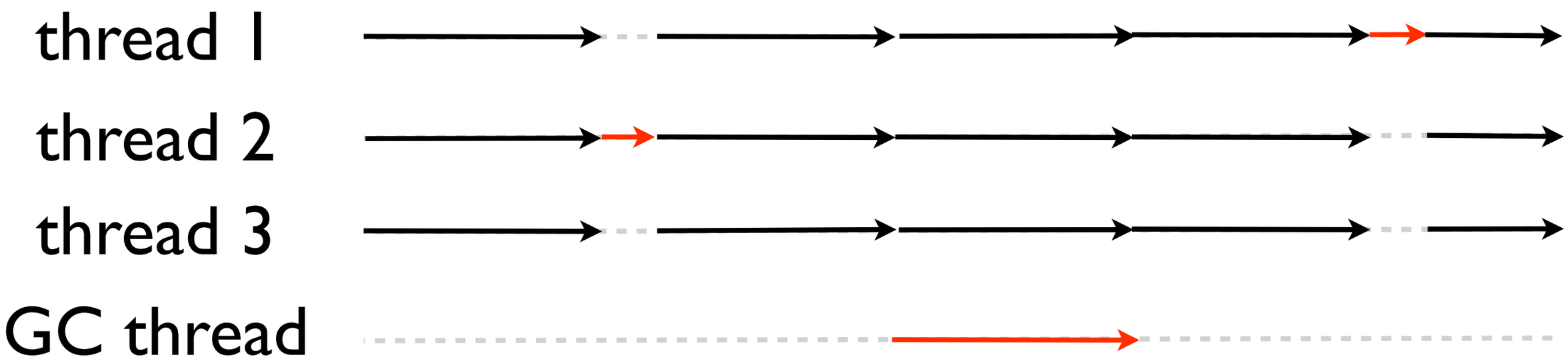
Parallelism

Higher throughput

Concurrency

Shorter stop the world


thread 1
thread 2
thread 3

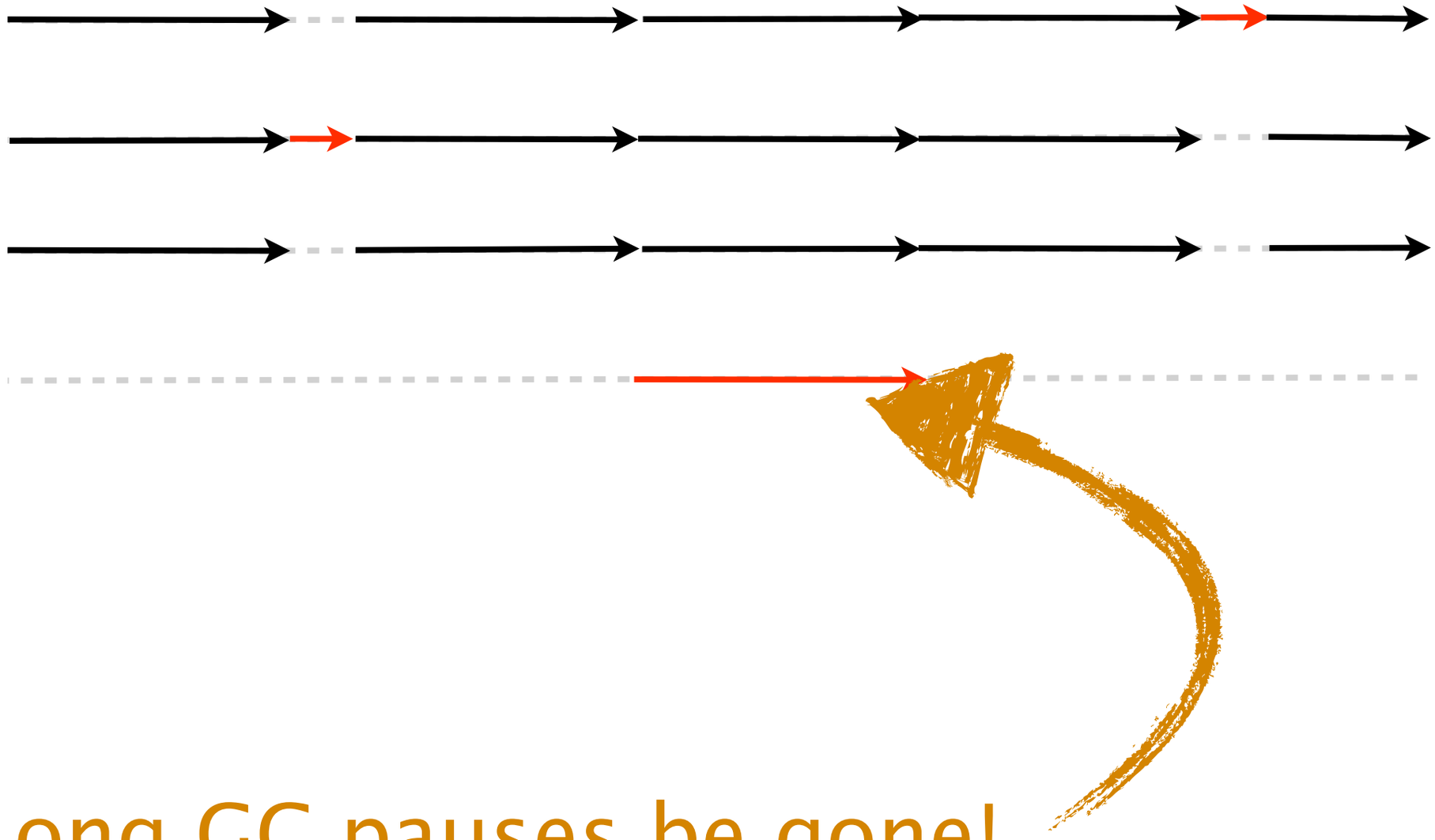

thread 1
thread 2
thread 3
GC thread

thread 1

thread 2

thread 3

GC thread



Long GC pauses be gone!

http://rubini.us/2013/06/22/concurrent-garbage-collection/

How much does it help?

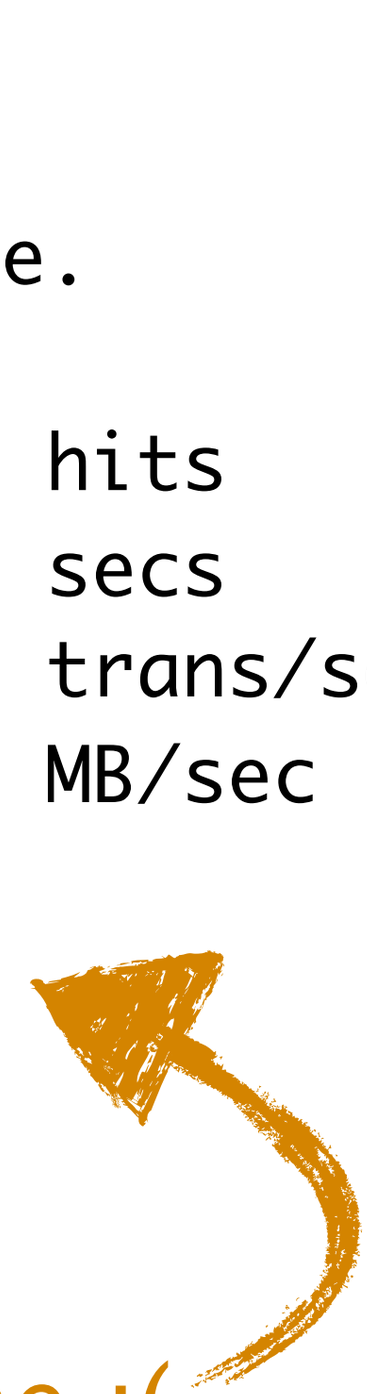
Before

```
Lifting the server siege...      done.

Transactions:                    501 hits
Response time:                  0.02 secs
Transaction rate:              51.70 trans/sec
Throughput:                     1.46 MB/sec
Concurrency:                    1.00
Longest transaction:            0.15
Shortest transaction:           0.01
```

```
Lifting the server siege...      done.

Transactions:                    501 hits
Response time:                  0.02 secs
Transaction rate:              51.70 trans/sec
Throughput:                     1.46 MB/sec
Concurrency:                    1.00
Longest transaction:            0.15
Shortest transaction:           0.01
```



Sad long wait time :(

```
Lifting the server siege...      done.

Transactions:                   1032 hits
Response time:                     0.04 secs
Transaction rate:                102.79 trans/sec
Throughput:                        2.91 MB/sec
Concurrency:                       4.00
Longest transaction:               0.28
Shortest transaction:              0.02
```

```
Lifting the server siege...      done.

Transactions:                   1032 hits
Response time:                     0.04 secs
Transaction rate:                102.79 trans/sec
Throughput:                        2.91 MB/sec
Concurrency:                       4.00
Longest transaction:               0.28
Shortest transaction:              0.02
```

Not so good concurrency

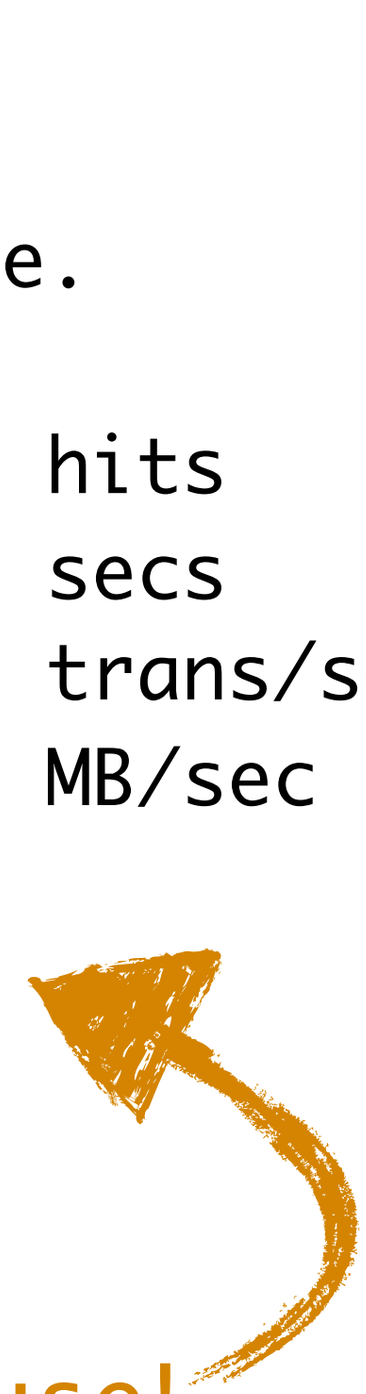
After

```
Lifting the server siege...      done.

Transactions:                    599 hits
Response time:                  0.02 secs
Transaction rate:              61.06 trans/sec
Throughput:                     1.73 MB/sec
Concurrency:                    1.00
Longest transaction:            0.03
Shortest transaction:           0.01
```

```
Lifting the server siege...      done.

Transactions:                    599 hits
Response time:                  0.02 secs
Transaction rate:              61.06 trans/sec
Throughput:                     1.73 MB/sec
Concurrency:                    1.00
Longest transaction:            0.03
Shortest transaction:           0.01
```



No more long pause!

```
Lifting the server siege...      done.

Transactions:                   1374 hits
Response time:                     0.02 secs
Transaction rate:                176.38 trans/sec
Throughput:                        5.00 MB/sec
Concurrency:                       3.99
Longest transaction:               0.04
Shortest transaction:              0.01
```

```
Lifting the server siege...      done.

Transactions:                   1374 hits
Response time:                     0.02 secs
Transaction rate:                176.38 trans/sec
Throughput:                        5.00 MB/sec
Concurrency:                       3.99
Longest transaction:               0.04
Shortest transaction:              0.01
```

More req/s!

1 client: 51 => 61 req/s

4 clients: 102 => 176 req/s

Concurrency & parallelism

No GIL/GVL!

Rubinius

Use Ruby™

Future is multi-core

We need new API's

Parallel each

Thread safe collections

What about you as a developer?

Be nice

Write type stable code

Small and simple methods

Benchmark!