EventMachine
Что делать, если вы соскучились по callback-ам?

Николай Норкин, 7Pikes

DevConf

http://www.devconf.ru

Что такое асинхронность?

Наша жизнь синхронна и однопоточна

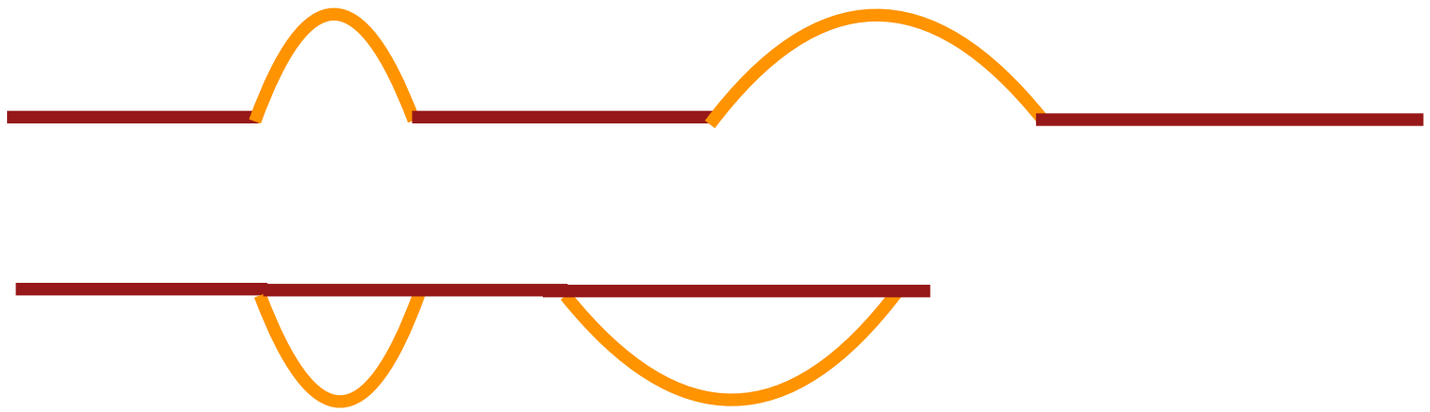
Наша жизнь синхронна и однопоточна

Асинхронность в вычислительной технике

Работа Ожидание

Асинхронность в вычислительной технике

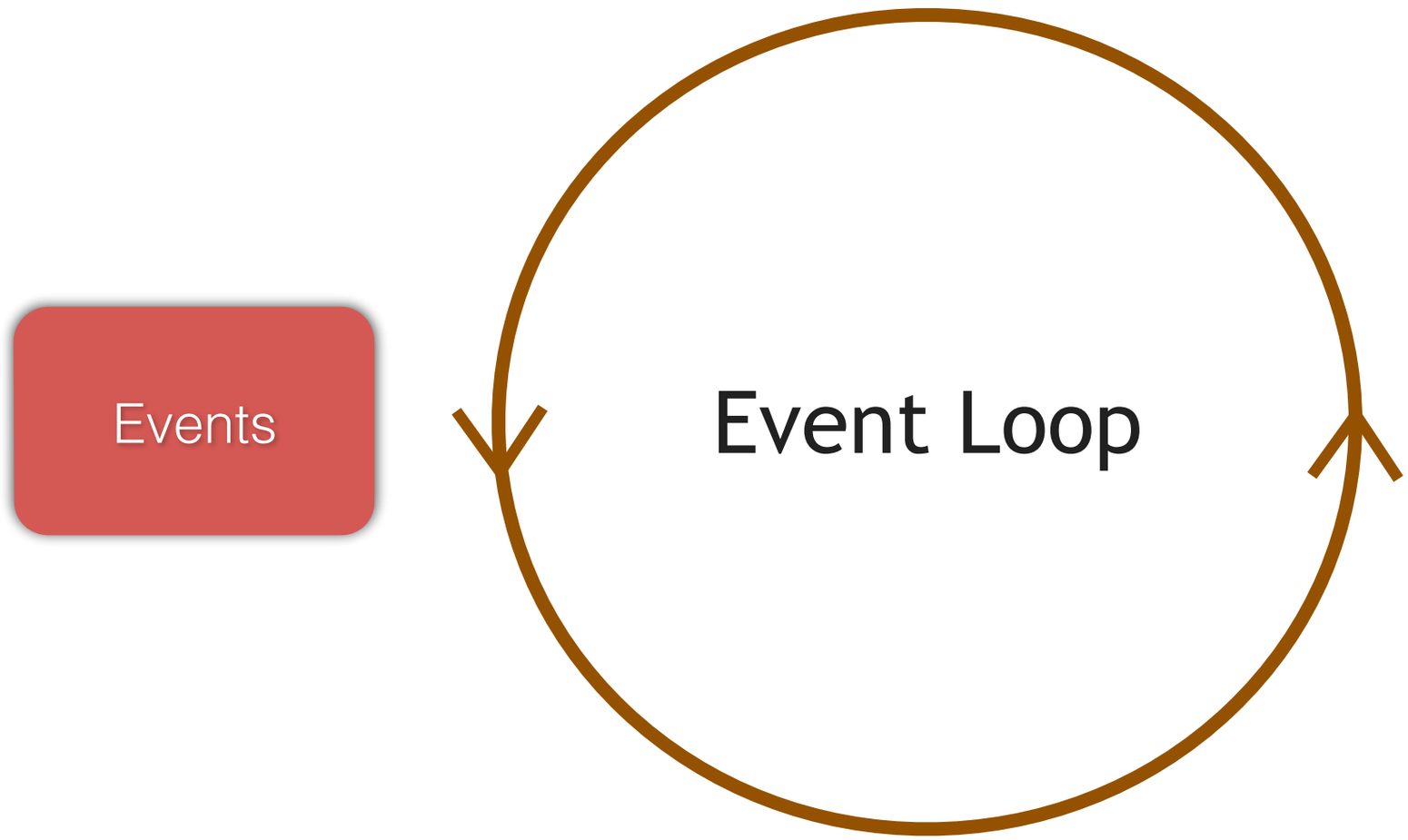
Работа Ожидание



Reactor

Reactor

Ожидание событий



Обработка событий

EventMachine

Когда нам нужен EventMachine?

Когда нам нужен EventMachine?

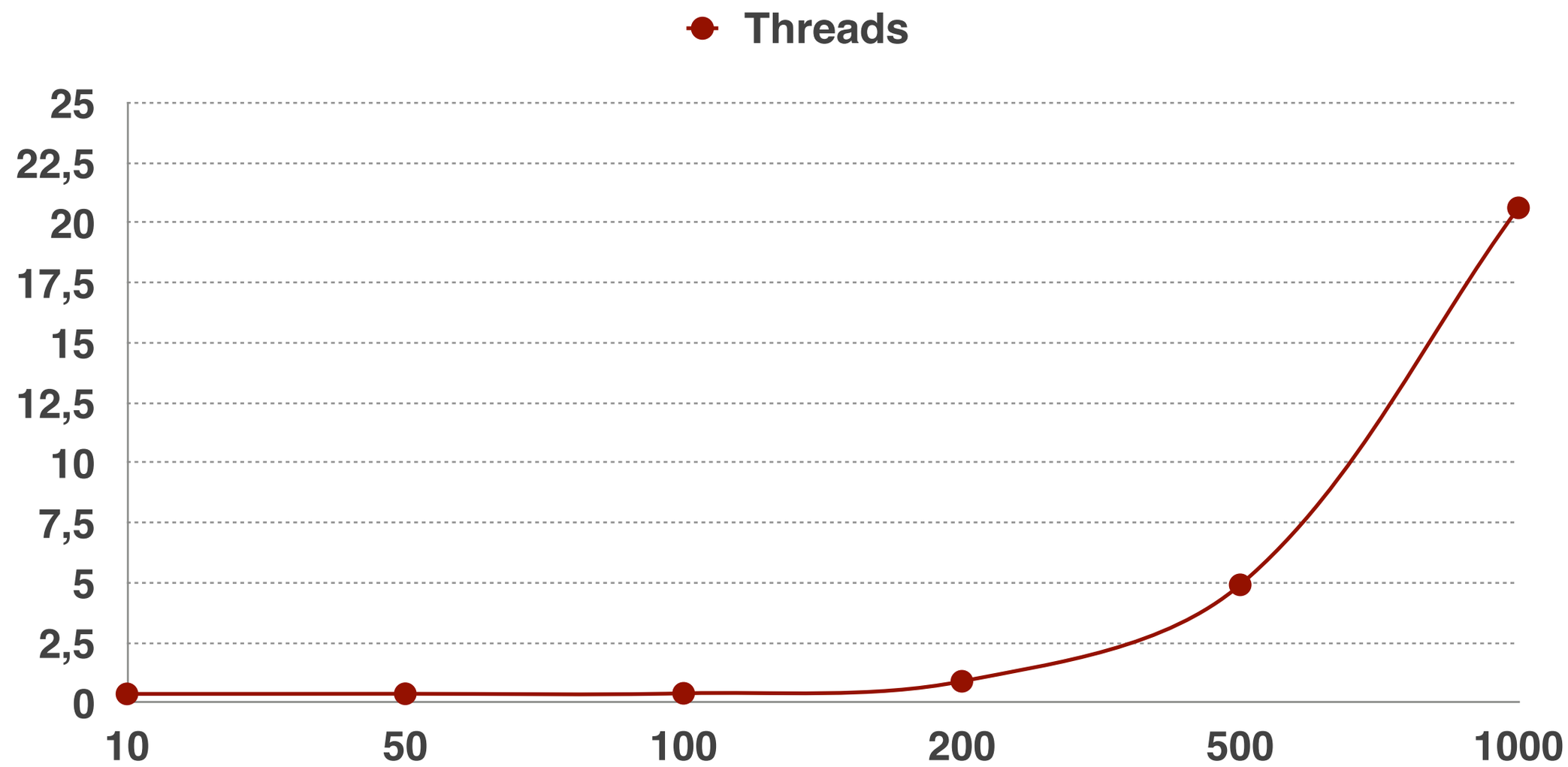
- Работа с сетью (HTTP, TCP, e.t.c)
- Работа с ФС
- Запросы к БД
- любые другие операции, вынуждающие процесс ждать

Параллельные запросы

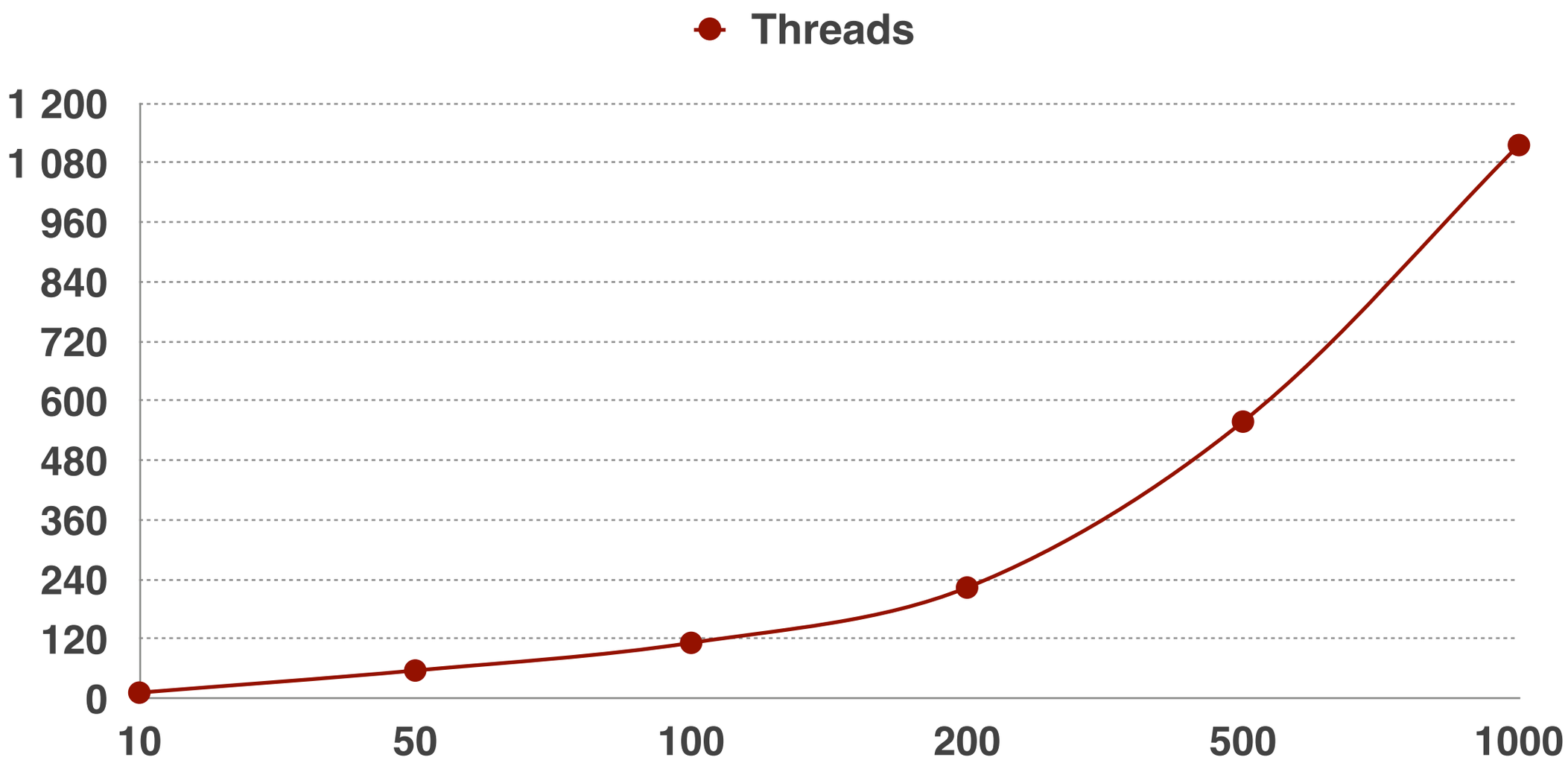
Threads

```
threads = []
responses = []
responses_mutex = Mutex.new

request_count.times do
  threads << Thread.new(responses) do |responses|
    response = RestClient.get URL
    responses_mutex.synchronize { responses << response }
  end
end

threads.each(&:join)
```

Время



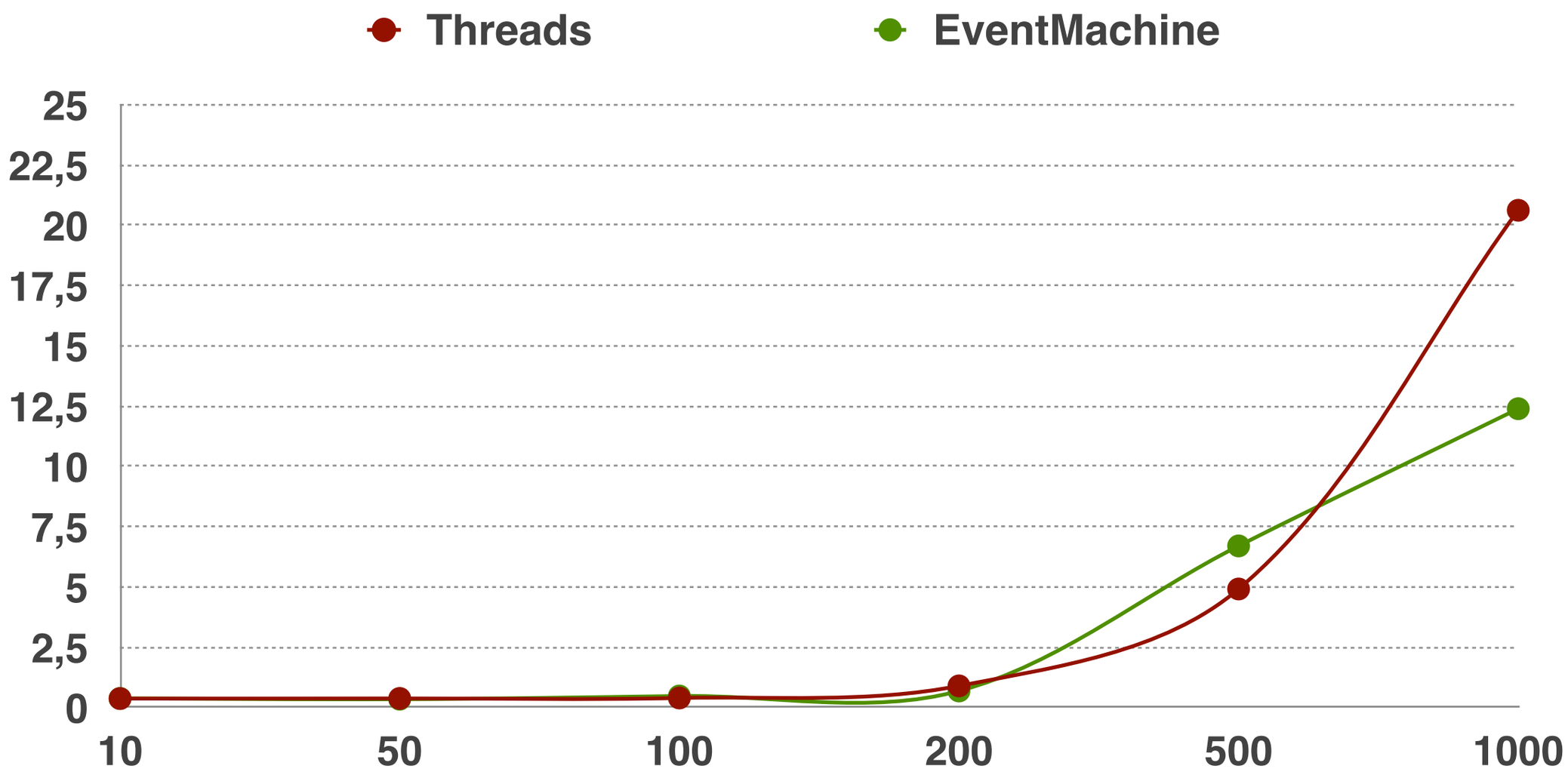
Память


Threads
1 200
1 080
960
840
720
600
480
360
240
120
0
10
50
100
200
500
1000

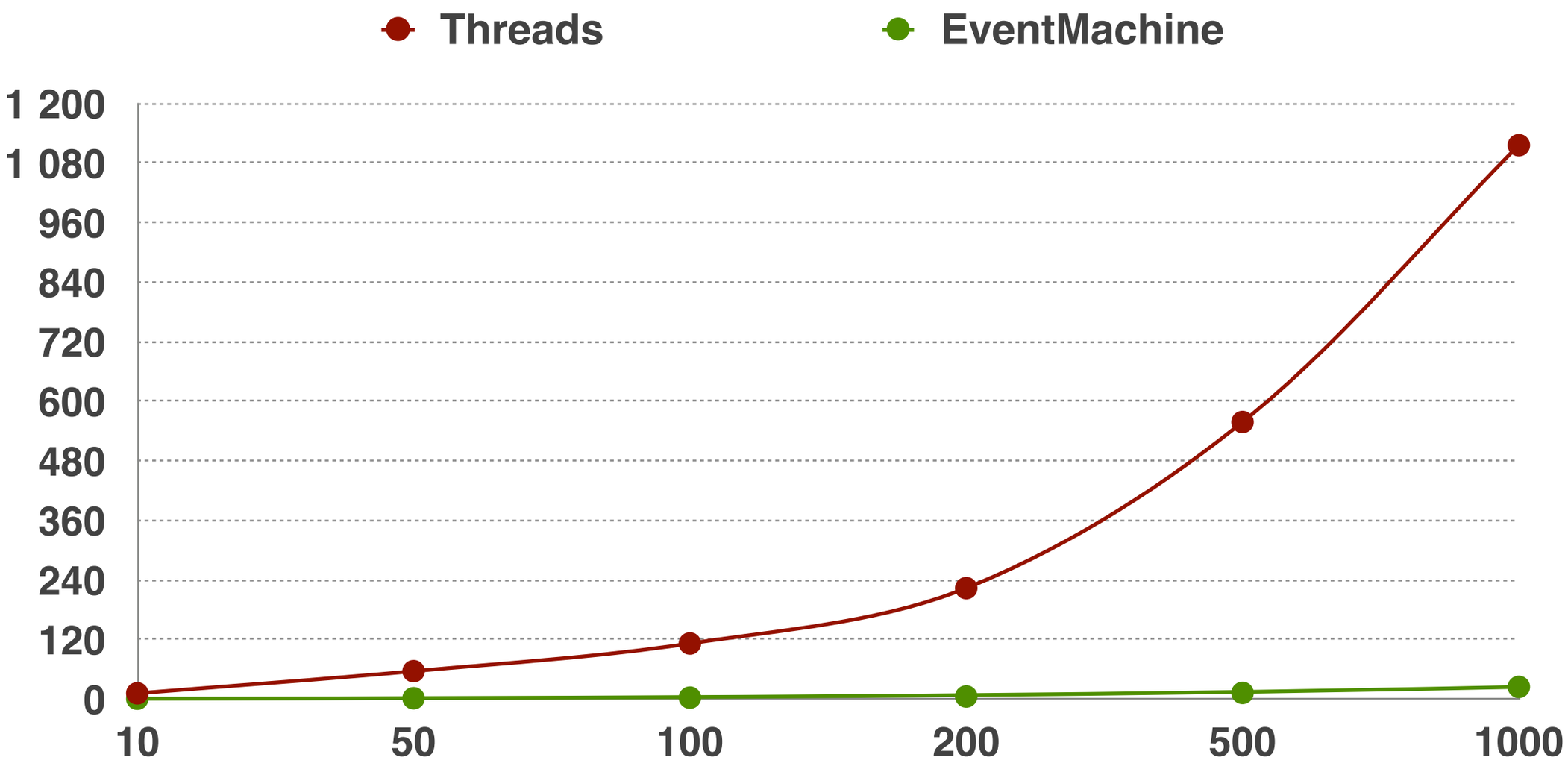
EventMachine

```
responses = []

EventMachine.run do
  multi = EventMachine::MultiRequest.new

  request_count.times { |i| multi.add i,
EM::HttpRequest.new(URL).get }

  multi.callback do
    responses = multi.responses[:callback].values.map(&:response)
    EventMachine.stop
  end
end
```

Время


Threads
EventMachine
25
22,5
20
17,5
15
12,5
10
7,5
5
2,5
0
10
50
100
200
500
1000

Память


Threads
EventMachine
1 200
1 080
960
840
720
600
480
360
240
120
0
10
50
100
200
500
1000

Быстродействие

rack

```ruby
require 'rubygems'
require 'rack'

class HelloWorld
  def call(env)
    [200, { 'Content-Type' => 'text/plain' }, ["Hi! I'm Rack!"]]
  end
end

Rack::Handler::WEBrick.run HelloWorld.new, Port: 8000
```

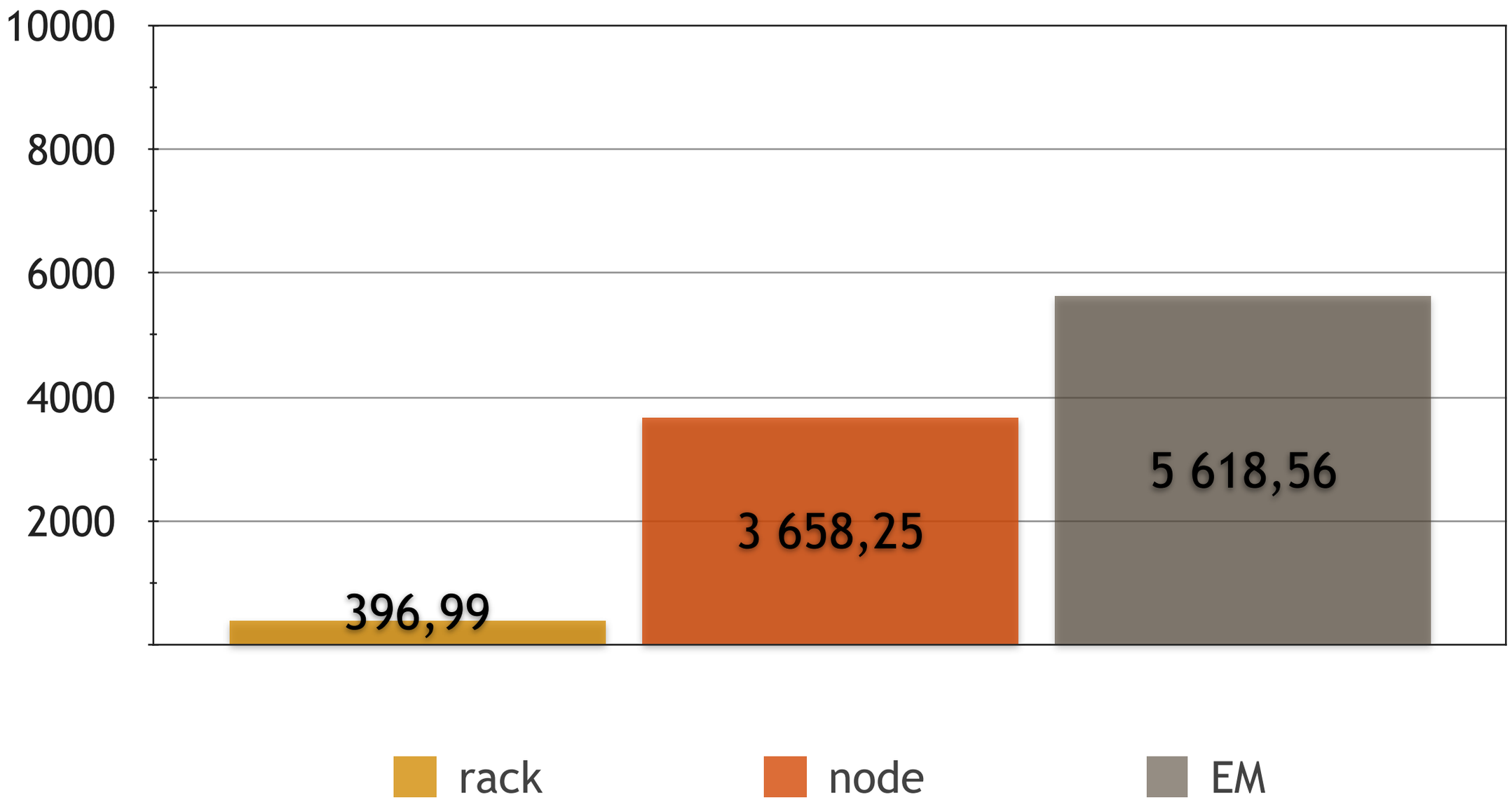
node

```
var http = require('http');

http.createServer(function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end("Hi! I'm Node.JS\n");
}).listen(8000, '127.0.0.1');
```

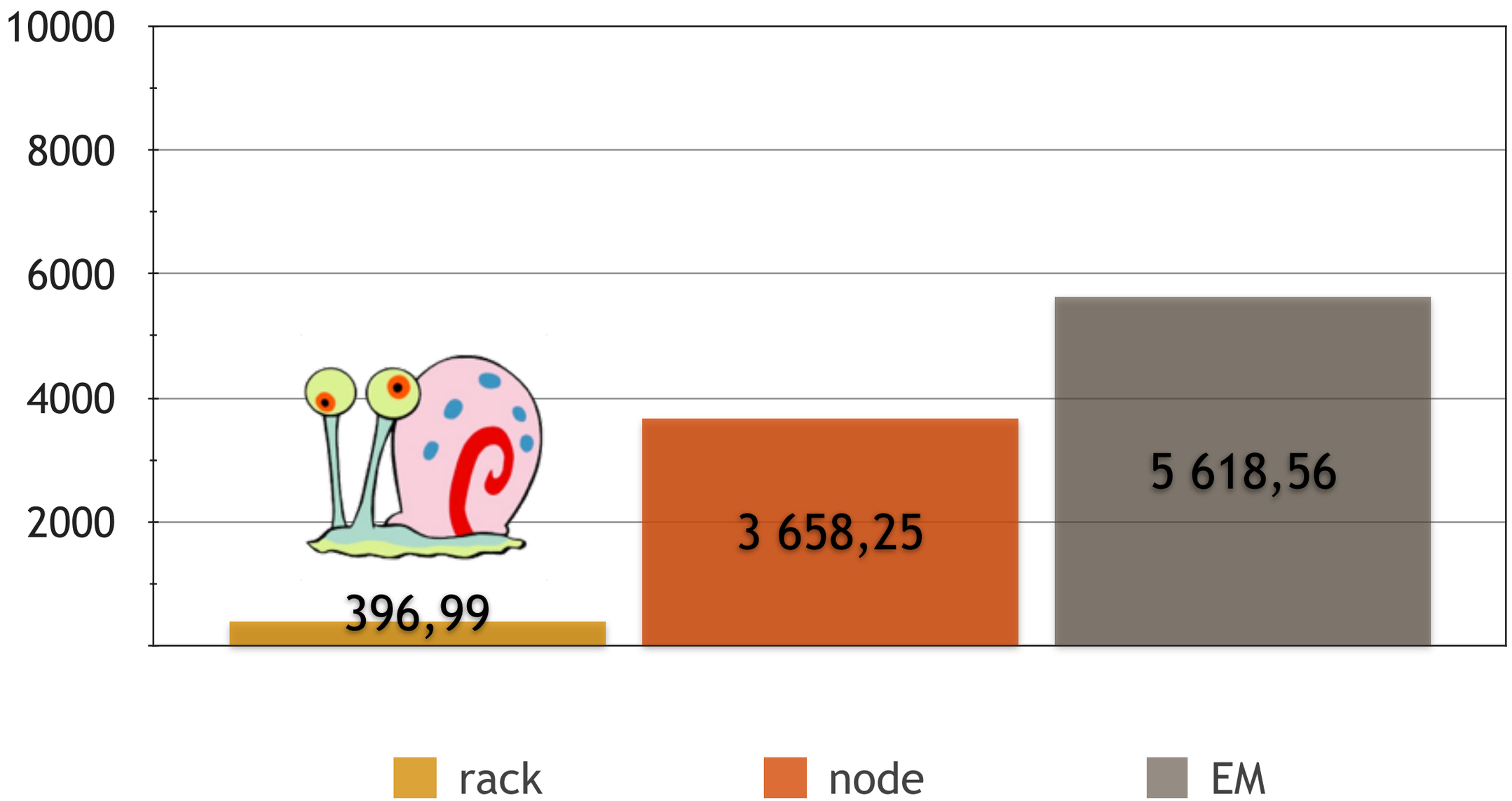
EventMachine

```
require 'rubygems'
require 'eventmachine'

class RequestHandler < EM::Connection
  def receive_data(_)
    data = "Hi! I'm EventMachine!"
    send_data("HTTP/1.1 200 OK\r\nContent-Type: text/plain\r
\n#{data}\n")
    close_connection_after_writing
  end
end

EM.run do
  EM.start_server('127.0.0.1', 8000, RequestHandler)
end
```

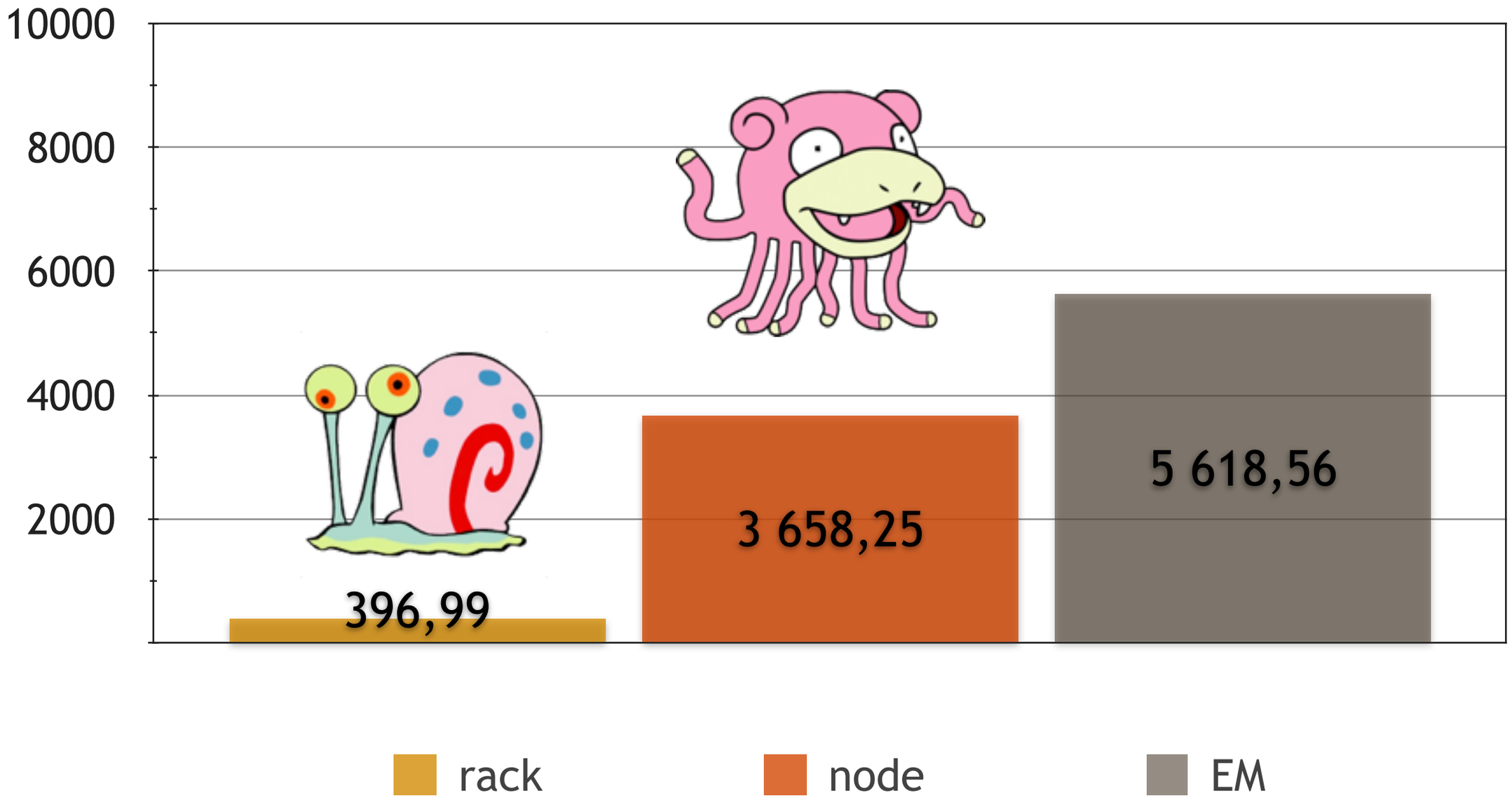
Запросов в секунду



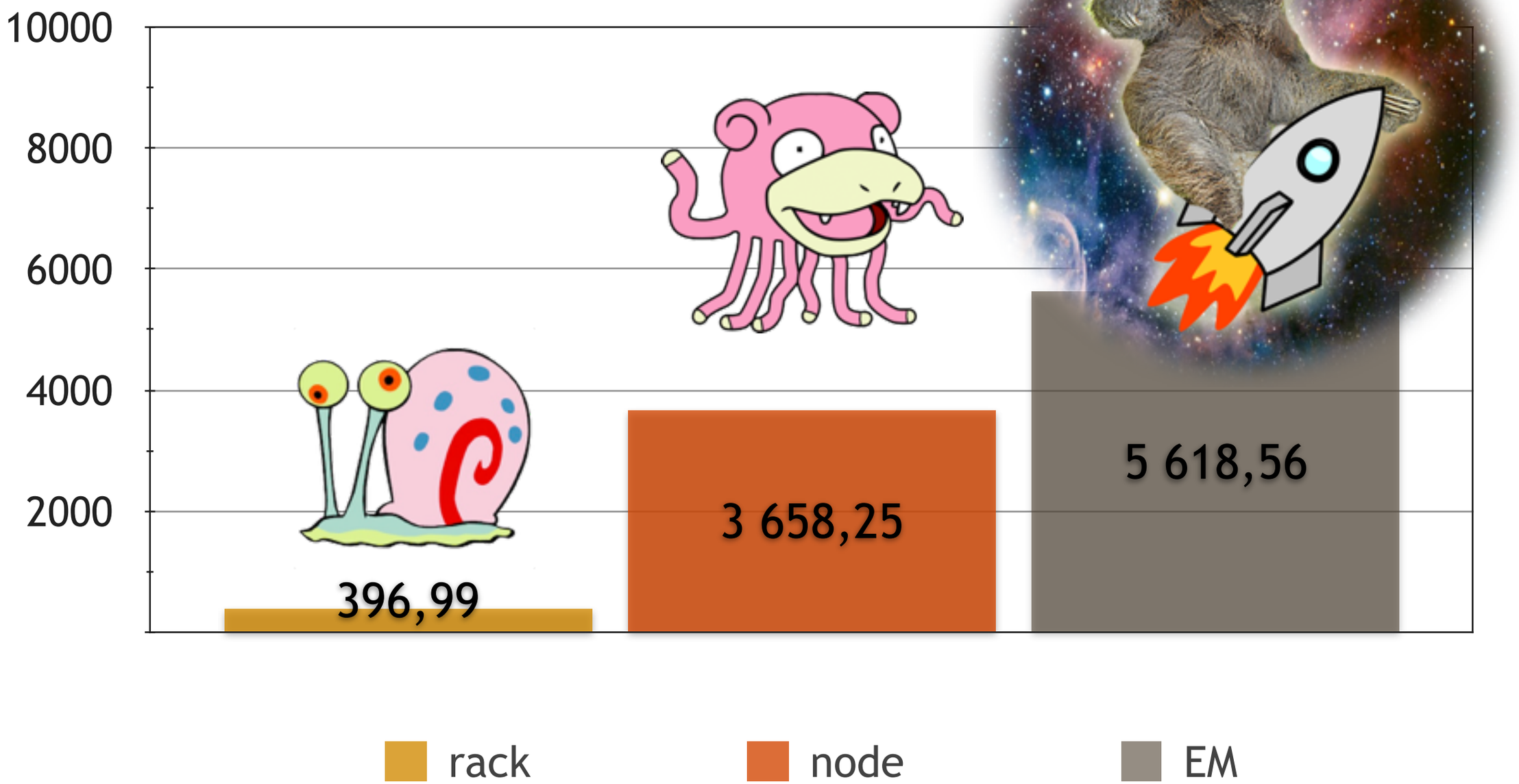
Запросов в секунду



Запросов в секунду



Запросов в секунду



Установка

```
gem install eventmachine
```

```
require "eventmachine"
```

```
EventMachine == EM
```

EventMachine API

EM.run

- Запускает EventLoop
- Принимает блок
- Запускается в текущем потоке

```ruby
EM.run do
  puts "Will be executed"
end
puts "Will be never reached"
```

EM.stop

```ruby
EM.run do
  puts "Will be executed"
  if EM.reactor_running?
    puts "Yeap. I'm running right now"
  end
  EM.stop
end
puts "Hoorayyy! I've been called too!"
```

Threads

Threads

```
thread = Thread.current
Thread.new do
  EM.run do
    thread.wakeup
    # do whatever you want
  end
end

Thread.stop
```

Threads

```
thread = Thread.current
Thread.new do
  EM.run do
    thread.wakeup
    # do whatever you want
  end
end

Thread.stop
```

- Не блокирует основной поток
- Печальная ситуация с тредами в ruby
- Появляются проблемы с thread-safety

EM.schedule

- Нужно использовать, если тред EM отличен от основного треда
- Бережет от threaded-проблем
- Обертка вокруг EM.next_tick

EM.next_tick

Ставит блок на следующую итерацию реактора

EM.next_tick

Ставит блок на следующую итерацию реактора

```
1000.times do
  long_processing
end
```

EM.next_tick

Ставит блок на следующую итерацию реактора

```
1000.times do
  long_processing
end
```

```
n = 0
iter = proc do
  if n < 1000
    long_processing
    n += 1
    EM.next_tick(&iter)
  end
end
EM.next_tick(&iter)
```

Таймеры

```ruby
EM.run do
  puts "It's time to take a nap"
  sleep(60*60*24*7)
  puts "Gosh... How long have I been sleeping?"
  EM.stop
end
```

```
EM.ru
  puts
  sle
  puts                                    ng?"
  EM.s
end
```

```ruby
EM::Timer.new(60*60*24*7) do
  puts "See ya one week later"
end
```

```ruby
EM::Timer.new(60*60*24*7) do
  puts "See ya one week later"
end
```

```ruby
timer = EM::PeriodicTimer.new(1) do
  puts "Second passed"
end
```

```ruby
EM::Timer.new(60*60*24*7) do
  puts "See ya one week later"
end
```

```ruby
timer = EM::PeriodicTimer.new(1) do
  puts "Second passed"
end
puts "...or not" if timer.cancel
```

EM.defer

Запускает блок параллельно основному потоку

EM.defer

Запускает блок параллельно основному потоку

В отдельном потоке

В основном потоке

```
EM.defer(
  proc { long_operation },
  proc { |result| process(result) }
)
```

EM::Deferrable

EM::Deferrable

- Это не defer
- Модуль
- Добавляется к вашим классам
- Коллбэки

```ruby
class DeepThought
  include EM::Deferrable

  attr_reader :answer

  def search
    puts 'Looking up for a meaning of life'
    EM.add_timer(60*60*24*365*7_500_000) do
      @answer = 42
      succeed(@answer)
    end
  rescue
    raise 'Oops... Something wrong'
  end
end

greatest_computer = DeepThought.new
greatest_computer.callback do |answer|
  puts 'Answer to the Ultimate Question' \
       ' of Life, the Universe,' \
       " and Everything is #{answer}"
end
greatest_computer.errback do |error|
  puts "Oops... #{error}"
end
greatest_computer.search
```

```ruby
class DeepThought
  include EM::Deferrable

  attr_reader :answer

  def search
    puts 'Looking up for a meaning of life'
    EM.add_timer(60*60*24*365*7_500_000) do
      @answer = 42
      succeed(@answer)
    end
  rescue
    raise 'Oops... Something wrong'
  end
end

greatest_computer = DeepThought.new
greatest_computer.callback do |answer|
  puts 'Answer to the Ultimate Question' \
       ' of Life, the Universe,' \
       " and Everything is #{answer}"
end
greatest_computer.errback do |error|
  puts "Oops... #{error}"
end
greatest_computer.search
```

Функциональность Deferrable

```ruby
class DeepThought
  include EM::Deferrable

  attr_reader :answer

  def search
    puts 'Looking up for a meaning of life'
    EM.add_timer(60*60*24*365*7_500_000) do
      @answer = 42
      succeed(@answer)
    end
  rescue
    raise 'Oops... Something wrong'
  end
end

greatest_computer = DeepThought.new
greatest_computer.callback do |answer|
  puts 'Answer to the Ultimate Question' \
       ' of Life, the Universe,' \
       " and Everything is #{answer}"
end
greatest_computer.errback do |error|
  puts "Oops... #{error}"
end
greatest_computer.search
```

Функциональность Deferrable

Успешное завершение

```ruby
class DeepThought
  include EM::Deferrable

  attr_reader :answer

  def search
    puts 'Looking up for a meaning of life'
    EM.add_timer(60*60*24*365*7_500_000) do
      @answer = 42
      succeed(@answer)
    end
  rescue
    raise 'Oops... Something wrong'
  end
end

greatest_computer = DeepThought.new
greatest_computer.callback do |answer|
  puts 'Answer to the Ultimate Question' \
       ' of Life, the Universe,' \
       " and Everything is #{answer}"
end
greatest_computer.errback do |error|
  puts "Oops... #{error}"
end
greatest_computer.search
```

Функциональность Deferrable

Успешное завершение

Ошибка

EM::Channel

Удобный thread-safe способ обмена данными внутри приложения

```ruby
channel = EM::Channel.new
sid = channel.subscribe do |msg|
  puts "Got #{msg}»
end
channel.push("message")
channel.unsubscribe(sid)
```

Работа с сетью

- EM.start_server (stop_server)
- EM.connect

```
server_signature =
  EM.start_server("0.0.0.0", 8080, ClientConnection)
EM.stop_server(server_signature)

EM.connect("/sockets/mysql.sock", MySqlConnection)
```

Работа с сетью

- EM.start_server (stop_server)
- EM.connect

```
server_signature =
  EM.start_server("0.0.0.0", 8080, ClientConnection)
EM.stop_server(server_signature)

EM.connect("/sockets/mysql.sock", MySqlConnection)
```

Наследует от `EM::Connection`

EM::Connection

События (вызывает EventMachine)

- post_init
- connection_completed
- receive_data
- unbind
- ssl_verify_peer
- ssl_handshake_completed

Методы (вызывает пользователь)

- send_data
- close_connection
- close_connection_after_writing
- get_peername
- pause/resume
- e.t.c

EM::Protocols

aka EM::P

Традиционный способ получения данных

Традиционный способ получения данных

```
send_data("Rudy")
send_data("A message to you Rudy")
```

Сеть

```
receive_data(«Rudy\n")
receive_data("A messa")
receive_data("ge to you Rudy\n")
```

С использование EM::Protocols

```
class Client < EM::Connection
  include EM::P::LineText2

  def receive_line(text)
    puts text
  end
end
```

С использование EM::Protocols

```
class Client < EM::Connection
  include EM::P::LineText2

  def receive_line(text)
    puts text
  end
end
```

EM::P::ObjectProtocol

```ruby
class Client < EM::Connection
  include EM::P::ObjectProtocol

  def receive_object(object)
    puts "Finally received an amazing #{object.class}"
    send_object(thank: "you!")
  end
end
```

Выведет «Finally received an amazing Hash»

Другие протоколы

- HttpClient and HttpClient2
- Stomp
- Memcache
- SmtpClient and SmtpServer
- SASLauth and SASLauthclient
- LineProtocol, LineAndTextProtocol and LineText2
- HeaderAndContentProtocol
- Postgres3
- ObjectProtocol

\+ Можно писать свои и есть куча сторонних

Пример

```
module EventMachine
  module Protocols
    module ApostoreMessage
      def receive_data(data)
        (@buf ||= '') << data
        while @buf.slice!(/#{Apostore::Config.message_beggining}(?
<msg>.*?)#{Apostore::Config.message_ending}/)
          message = Regexp.last_match[:msg]
          next if message.nil?
          parsed_message = Apostore::Message.new(message)
          receive_message(parsed_message)
        end
      end

      def receive_message(_message)
        # stub
      end
    end
  end
end
```

DevConf

C10K

select/poll и epoll

- select и poll медленные. Они вынужденны передавать все дескрипторы и возвращать состояние для всех
- Аналогично не только для файлов, но и для соединений и т.п.
- epoll/kqueue срабатывает при готовности дескриптора
- EM по умолчанию работает с select

Как включить epoll?

```
EM.epoll
```

Как включить epoll?

```
EM.epoll
```

Другие фичи

- system - системные вызовы
- watch - мониторинг дескриптора
- attach - подключение объектов ввода-вывода
- watch_file - мониторинг файлов
- watch_process - мониторинг процессов
- popen - работа с внешними процессами

e.t.c

Недостатки

Недостатки

- Колбэки

```
question = some_operation
response = get_http(question)
answer = build_answer(response)
send_answer(answer)
log("done")
```

```
question = some_operation
response = get_http(question)
answer = build_answer(response)
send_answer(answer)
log("done")
```

```
some_operation do |question|
  get_http(question) do |response|
    build_answer(response) do |answer|
      send_answer(answer) do
        log("done")
      end
    end
  end
end
```

Недостатки

- Колбэки
- thread-safety

```ruby
x = 0

10.times.map do |i|
  EM.defer do
    puts "before (#{ i }): #{ x }"
    x += 1
    puts "after (#{ i }): #{ x }"
  end
end
```

```
before (0): 0
after (0): 1
before (2): 1
before (4): 1
after (4): 3
before (7): 1
after (7): 4
after (2): 2before (5): 1
after (5): 5
before (9): 1
before (3): 1
before (6): 1
after (6): 8
after (3): 7
before (8): 1
after (8): 9
after (9): 6
before (1): 1
```

```ruby
x, mutex = 0, Mutex.new

10.times.map do |i|
  EM.defer do
    mutex.synchronize do
      puts "before (#{ i }): #{ x }"
      x += 1
      puts "after (#{ i }): #{ x }"
    end
  end
end
```

```
before (0): 0
after (0): 1
before (1): 1
after (1): 2
before (2): 2
after (2): 3
before (3): 3
after (3): 4
before (4): 4
after (4): 5
before (5): 5
after (5): 6
before (6): 6
after (6): 7
before (7): 7
after (7): 8
before (8): 8
after (8): 9
before (9): 9
after (9): 10
```

Недостатки

- Колбэки
- thread-safety
- Утечки и разбухание памяти

PHP

- По процессу на запрос
- Память вычищается после каждого запроса

EM

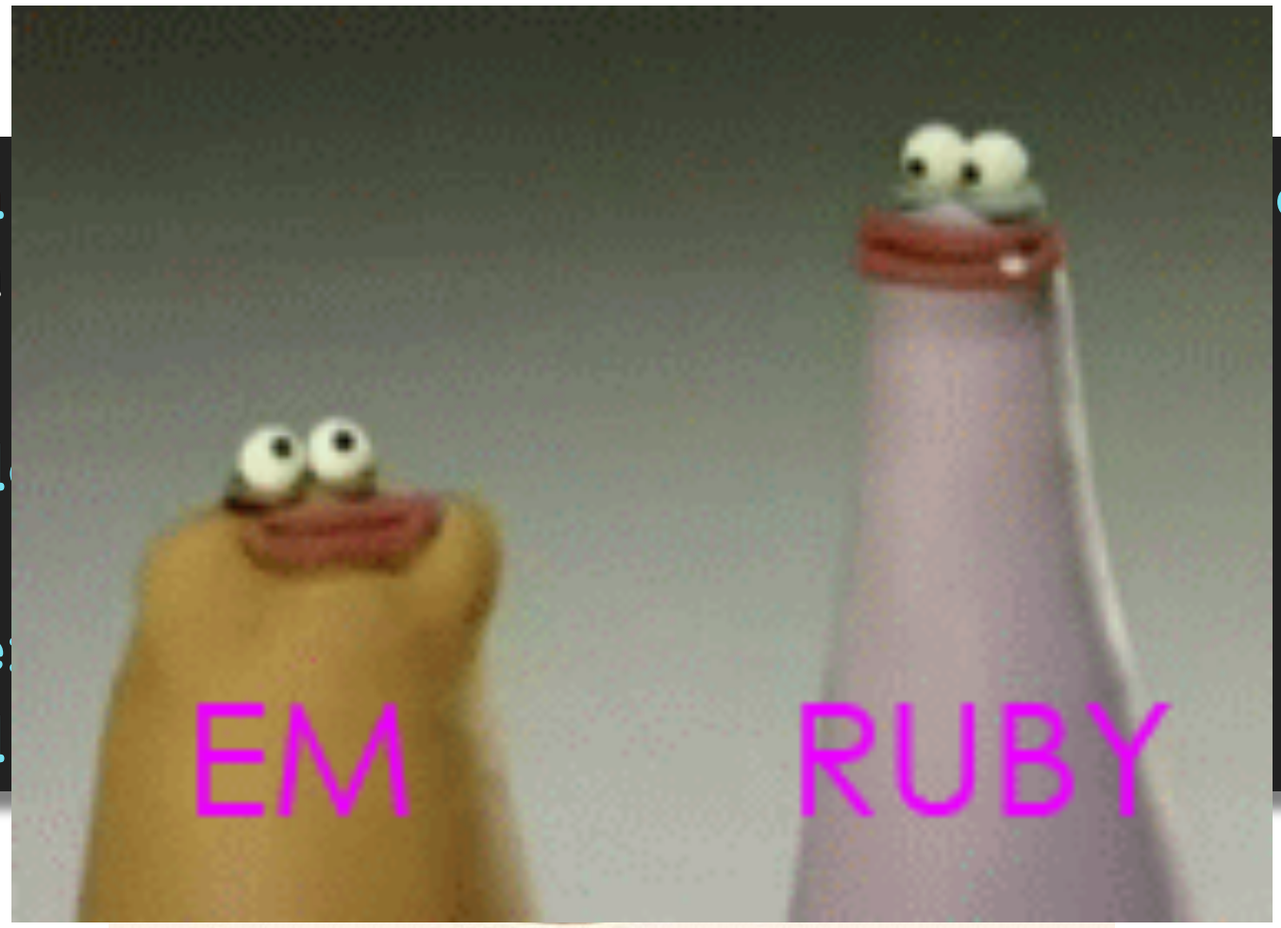
- Все выполняется в одном процессе
- Следить за памятью нужно самому

```ruby
class Connection < EM::Connection
  @@connections = []

  def post_init
    @@connections << self
  end
end
```

```
class
  @@c
  def
    @
  end
end
```


```
ection
```

```
cla                                    on
  @

  d

    e
end
```



```
cla                                    on
  @

  d

  e
end
```

Real-life

```
Process 'robot'
  status                              running
  monitoring status                   monitored
  pid                                 2153
  parent pid                          1
  uptime                              105d 5h 50m
  children                            0
  memory kilobytes                    33352
  memory kilobytes total              33352
  memory percent                      0.8%
  memory percent total                0.8%
  cpu percent                         0.0%
  cpu percent total                   0.0%
```

Недостатки

- Колбэки
- thread-safety
- Утечки и разбухание памяти
- Блокирование неблокирующего ввода-вывода

```ruby
require 'eventmachine'

EM.run do
  start_time = Time.now

  EM.next_tick do
    puts Time.now - start_time
    sleep(2)
  end

  EM::Timer.new(1) do
    puts Time.now - start_time
    EM.stop
  end

  sleep(2)
end
```

```ruby
require 'eventmachine'

EM.run do
  start_time = Time.now

  EM.next_tick do
    puts Time.now - start_time
    sleep(2)
  end

  EM::Timer.new(1) do
    puts Time.now - start_time
    EM.stop
  end

  sleep(2)
end
```

```
2.004218
4.004673
```

Недостатки

- Колбэки
- thread-safety
- Утечки и разбухание памяти
- Блокирование неблокируемого ввода-вывода
- MRI, GIL

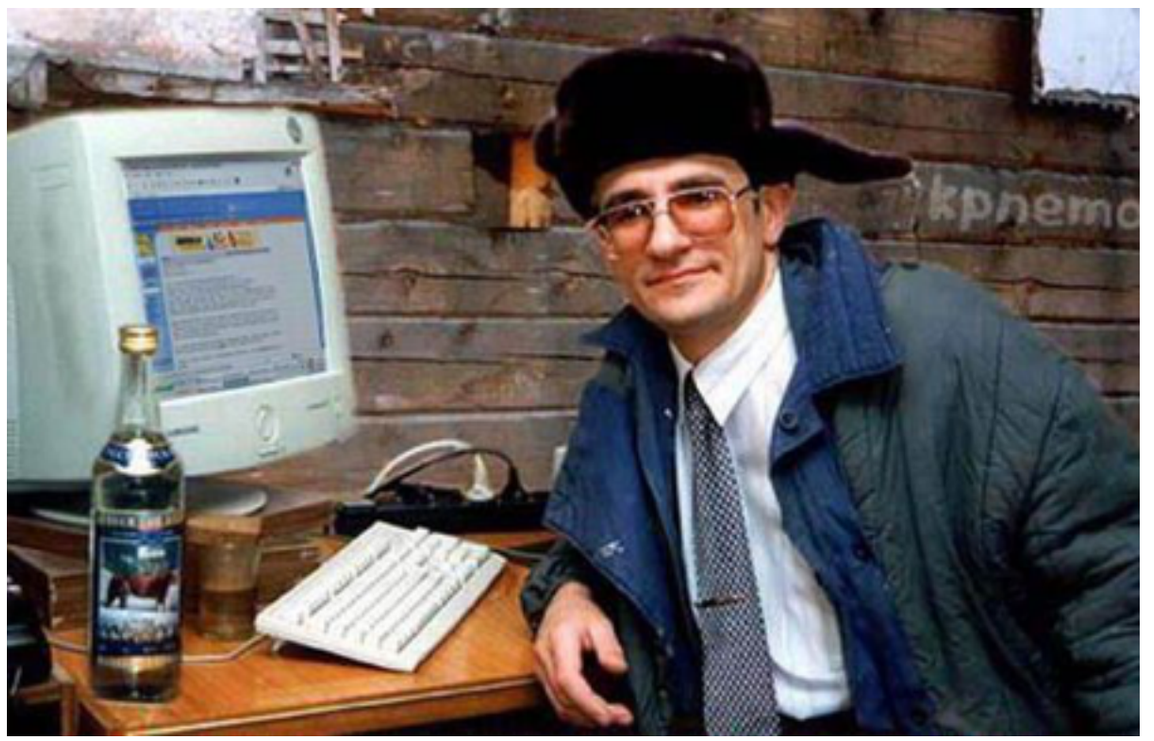
Concurency vs Parallelism

Concurency vs Parallelism

Concurency vs Parallelism

Concurency vs Parallelism

Concurency vs Parallelism



Concurency vs Parallelism

Concurency vs Parallelism

Concurency vs Parallelism

- MRI
- Rubinius 1
- MagLev

- Rubinius 2
- JRuby
- MacRuby

Недостатки

- Колбэки
- thread-safety
- Утечки и разбухание памяти
- Блокирование неблокируемого ввода-вывода
- MRI, GIL
- Бедноватая документашка

Инструменты на основе EventMachine

- Thin
- Goliath
- amqp
- logstash
- websocket-rails

Библиотеки

- https://github.com/igrigorik/async-rails
- https://github.com/kyledrake/sinatra-synchrony
- https://github.com/stevegraham/em-rspec
- https://github.com/jcoglan/rspec-eventmachine
- **https://github.com/igrigorik/em-synchrony**

em-synchrony

- Делает EM «синхронным»
- Основан на Fiber
- Содержит много синхронизированных частей EventMachine
- Возможность синхронизировать любую асинхронную функцию

Как работает?

```
def http_get(url)
  f = Fiber.current
  http = EventMachine::HttpRequest.new(url).get
  http.callback { f.resume(http) }
  return Fiber.yield
end

EventMachine.run do
  Fiber.new{
    page = http_get('http://www.google.com/')
    puts "Fetched page: #{page.response_header.status}"
  }.resume
end
```

Выводы

- EventMachine добавляет скорости
- EventMachine добавляет проблем

Куда дальше?

- Homepage http://rubyeventmachine.com/
- GitHub https://github.com/eventmachine/eventmachine
- Wiki https://github.com/eventmachine/eventmachine/wiki

Вопросы?

n.e.norkin@gmail.com

https://github.com/duderman