Scaling Rails

a system-wide approach to performance optimization

MSc thesis presentation

July 2010
MIEIC / FEUP

Structure

Context

Problem

Approach

Results

Conclusions

Context

Problem

Approach

Results

Conclusions

Context

Problem

Approach

Results

Conclusions

Context

Problem

Approach

Results

Conclusions

Context

Problem

Approach

Results

Conclusions

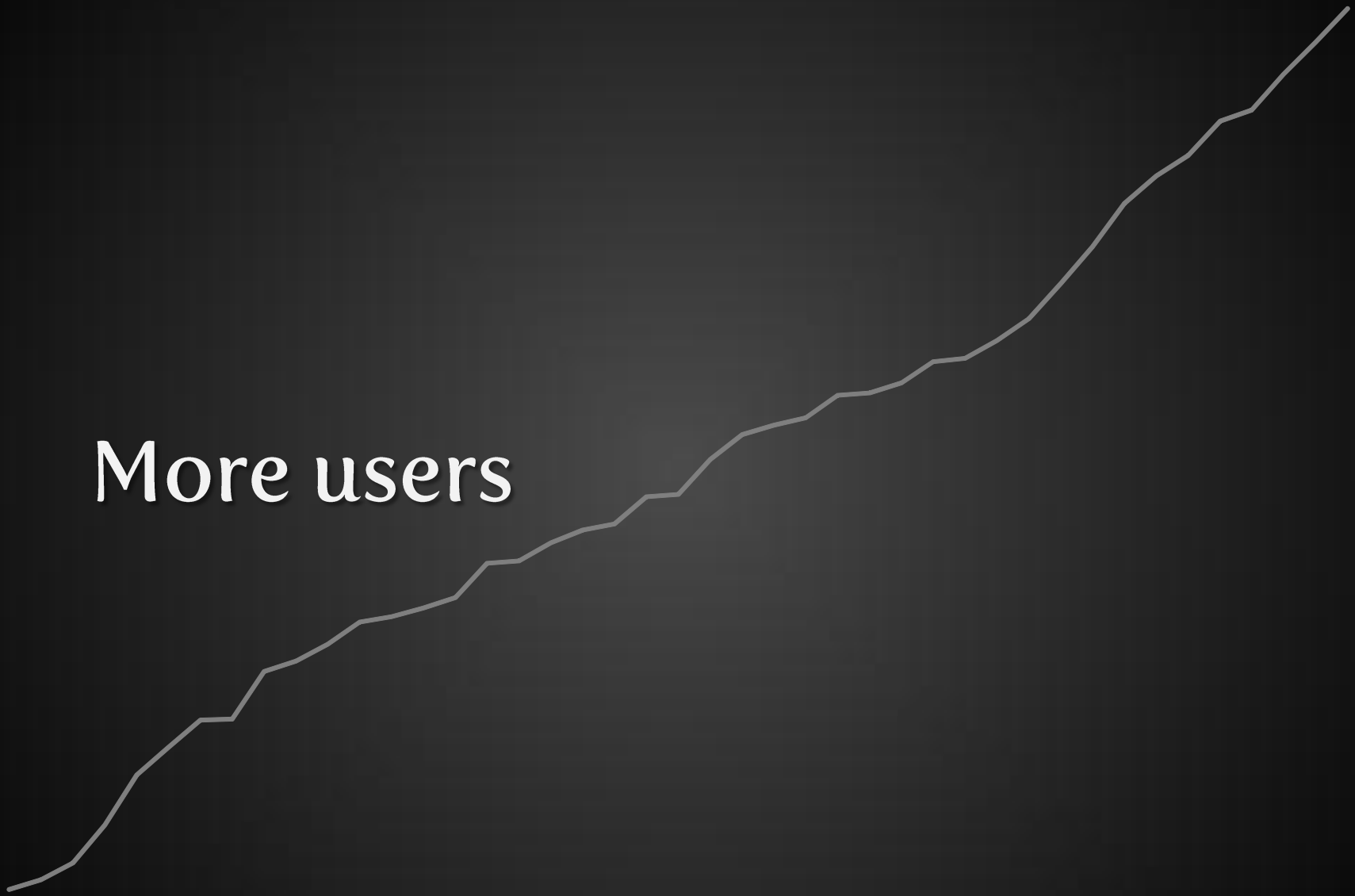
Context

Problem
Approach
Results
Conclusions

Web

More importance

More users



Evolution

Web 2.0

Better UX

Better response times

Better web, better tools

Ruby on Rails

Fast development

Full-featured applications

Popularity

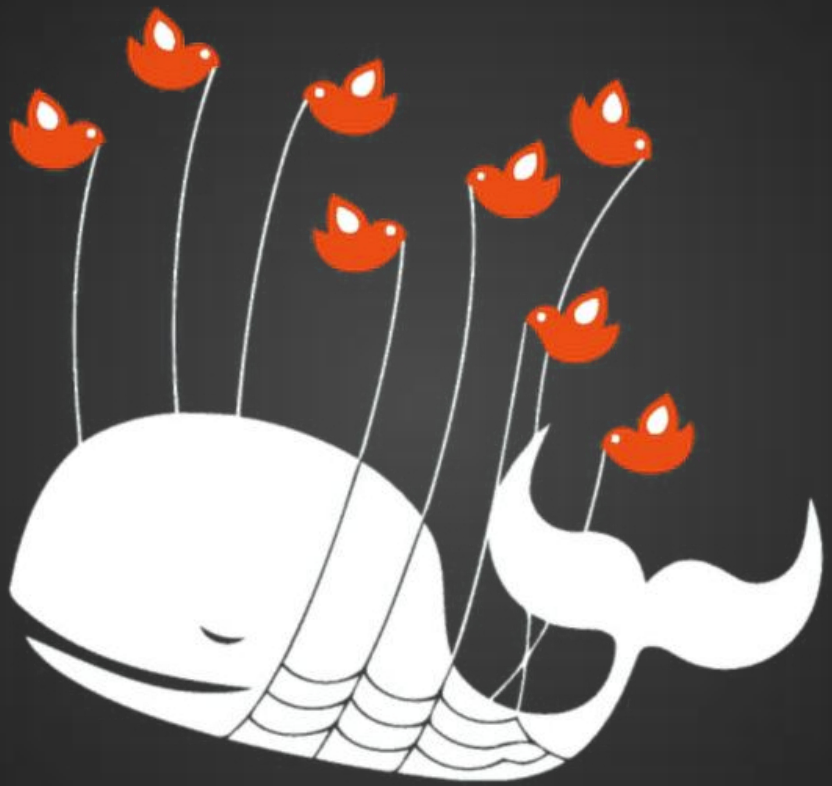
Scalability

Performance

Popular websites

Rails doesn't scale

Twitter

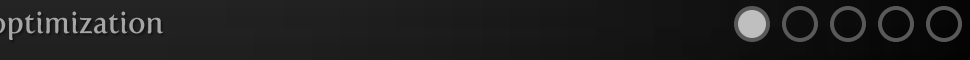


YellowPages

"Just throw hardware at it"

Rails doesn't scale

Hulu



Huge hardware infrastructure

Basecamp

David Heinemeier Hansson*

*Rails creator

What's missing here?

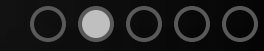
Context

Problem

Approach
Results
Conclusions

Focus on

Performance?

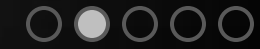


Scalability?

Complexity

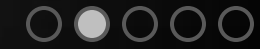
Rails

Ruby



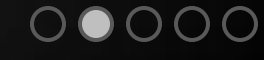
Web server

Database



Operating System

Not much, actually



Targeting 3 core issues

Information

There are benchmarks

There are configurations

There is information

Scattered

Outdated

Specific

Awareness

Ruby 1.9

Rails 3

2008

They are not commonly used

Applications

Plugins

Gems

Profiling

Native tools

Rails

Ruby

Which are broken

Don't integrate with each other

Text-based

Context
Problem

Approach

Results
Conclusions

Information

General guide

Centralized information

Publicly accessible

Awareness

General guide

Advantages of updating

Show off

Profiling

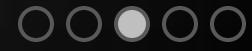
Native tools

Integrated

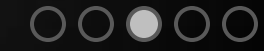
More powerful

More intuitive

How?



Target all system components



Unified perspective

Rails!

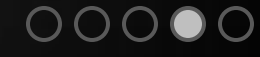
Context
Problem
Approach

Results

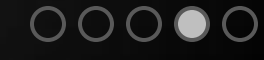
Conclusions

Operating System

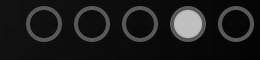
Development



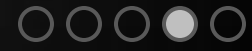
General benchmarking tool



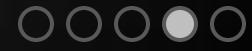
CPU-intensive



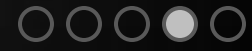
I/O



Workload



Benchmarking



General benchmark

Ubuntu, Debian, CentOS, Gentoo

Ubuntu – CPU usage

Debian – Workload

Gentoo – CPU + I/O

CentOS – inconsistent

Ubuntu – CPU usage

Debian – Workload

Gentoo – CPU + I/O

~~CentOS – inconsistent~~

Not really conclusive

Web server benchmark

> 30 seconds = failed request

Requests / Concurrency

10000 / 1000

100000 / 1000

100000 / 10000

Ubuntu, Debian, Gentoo

Apache / Nginx

Default configurations

Low or medium concurrency: OK

High concurrency: FAIL

Except…

Gentoo

Very stable

High scalability

What about BSD?

Gentoo, FreeBSD

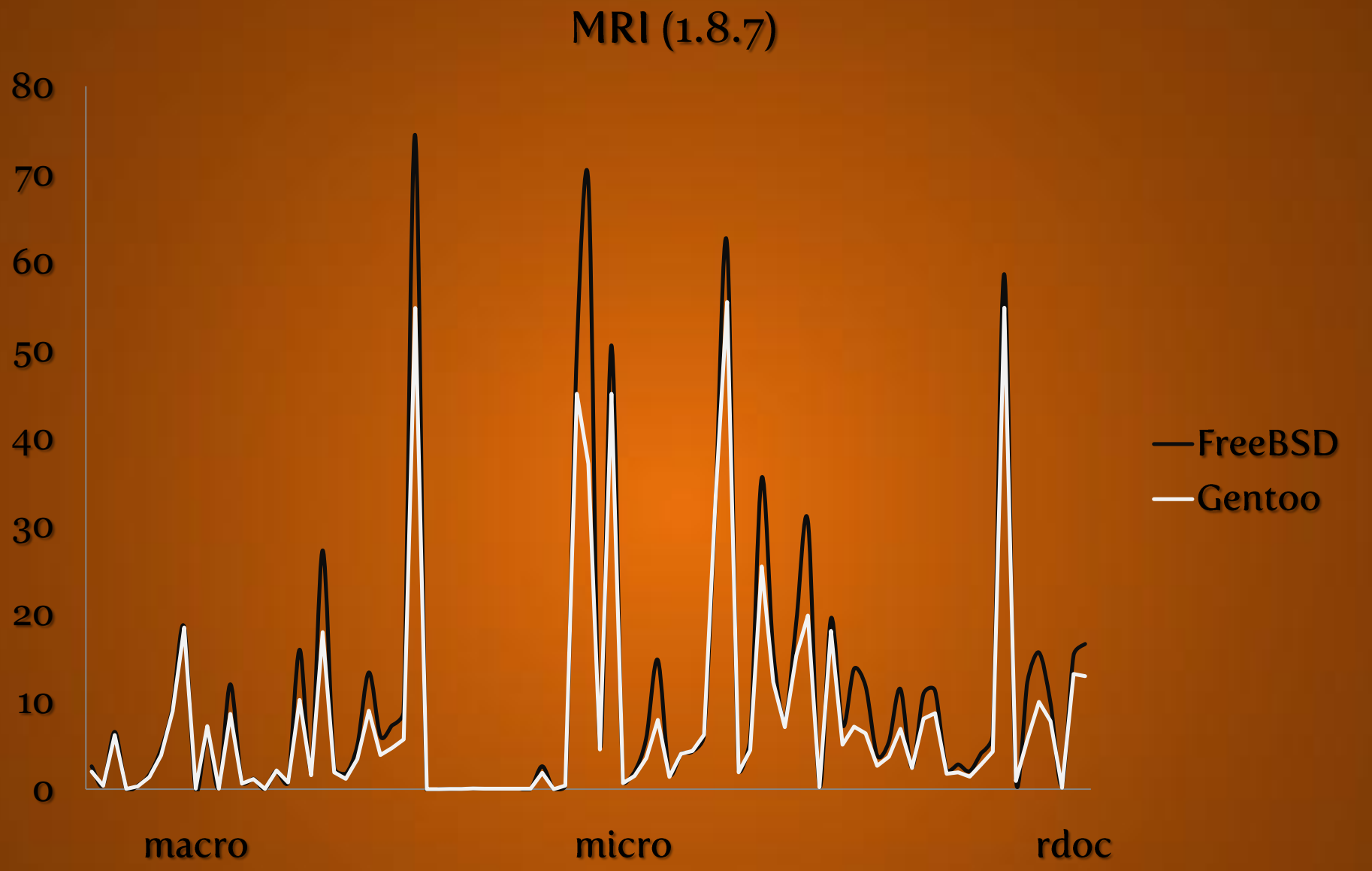
Ruby benchmarking suite

8 macro benchmarks

62 micro benchmarks

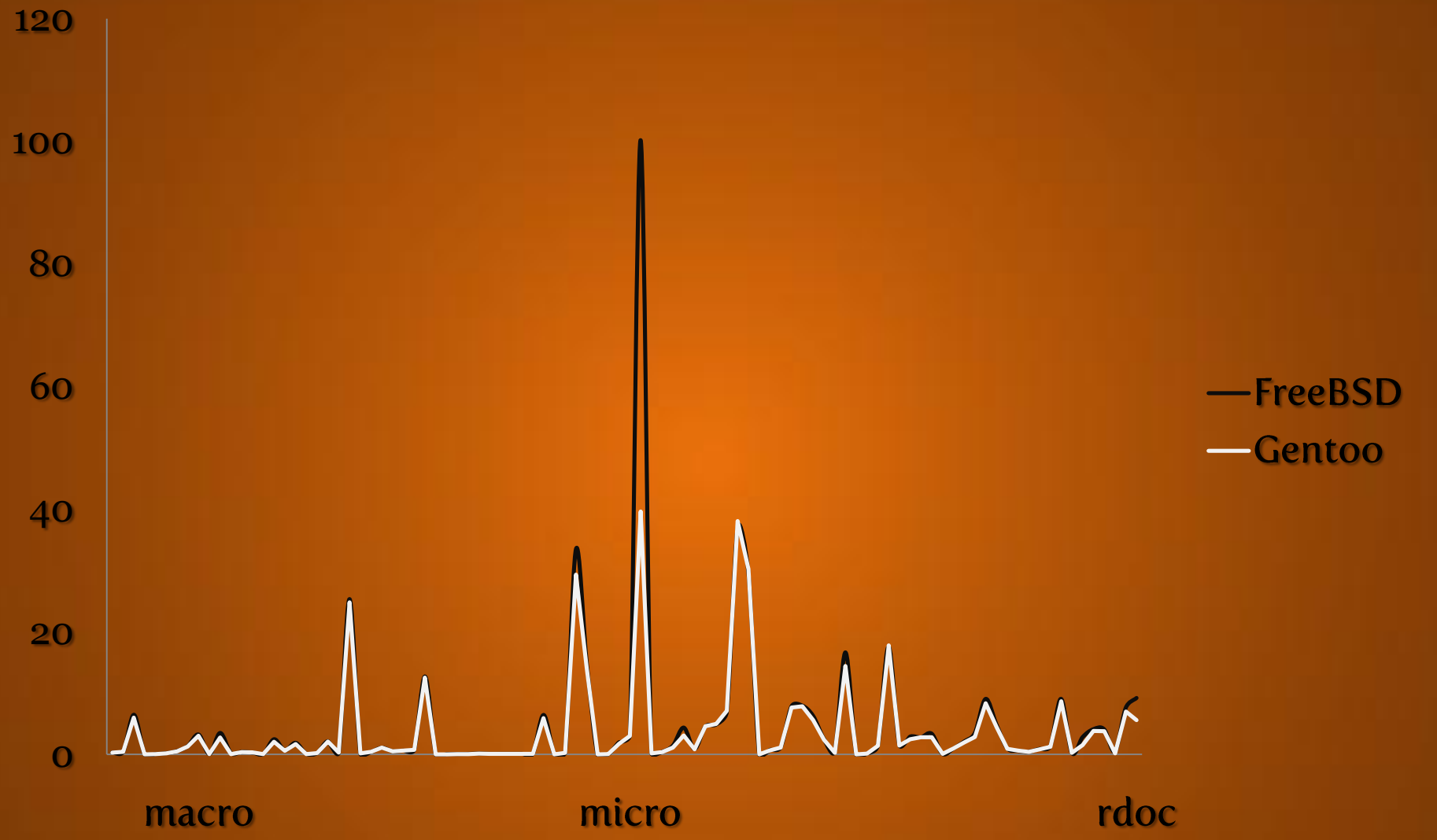
3 rdoc-related benchmarks

Time – less is better


MRI (1.8.7)
80
70
60
50
40
30
20
10
0
macro
micro
rdoc
FreeBSD
Gentoo

More ~29% time on FreeBSD overall

YARV (1.9.1)



More ~22% time on FreeBSD overall

Gentoo

Great performance

High configurability

Will be used in the remaining work

Tweaking

Deadline I/O Scheduler

Non-preemptive configuration

100Hz timer interrupt

TCP tweaks

Kernel buffer and limit tweaks

Operating System

Development ->

- Benchmarking tool

Benchmark ->

- General benchmark
- Web server benchmark
- Ruby benchmark

Tweaking ->

- Kernel fine-tunning

Ruby

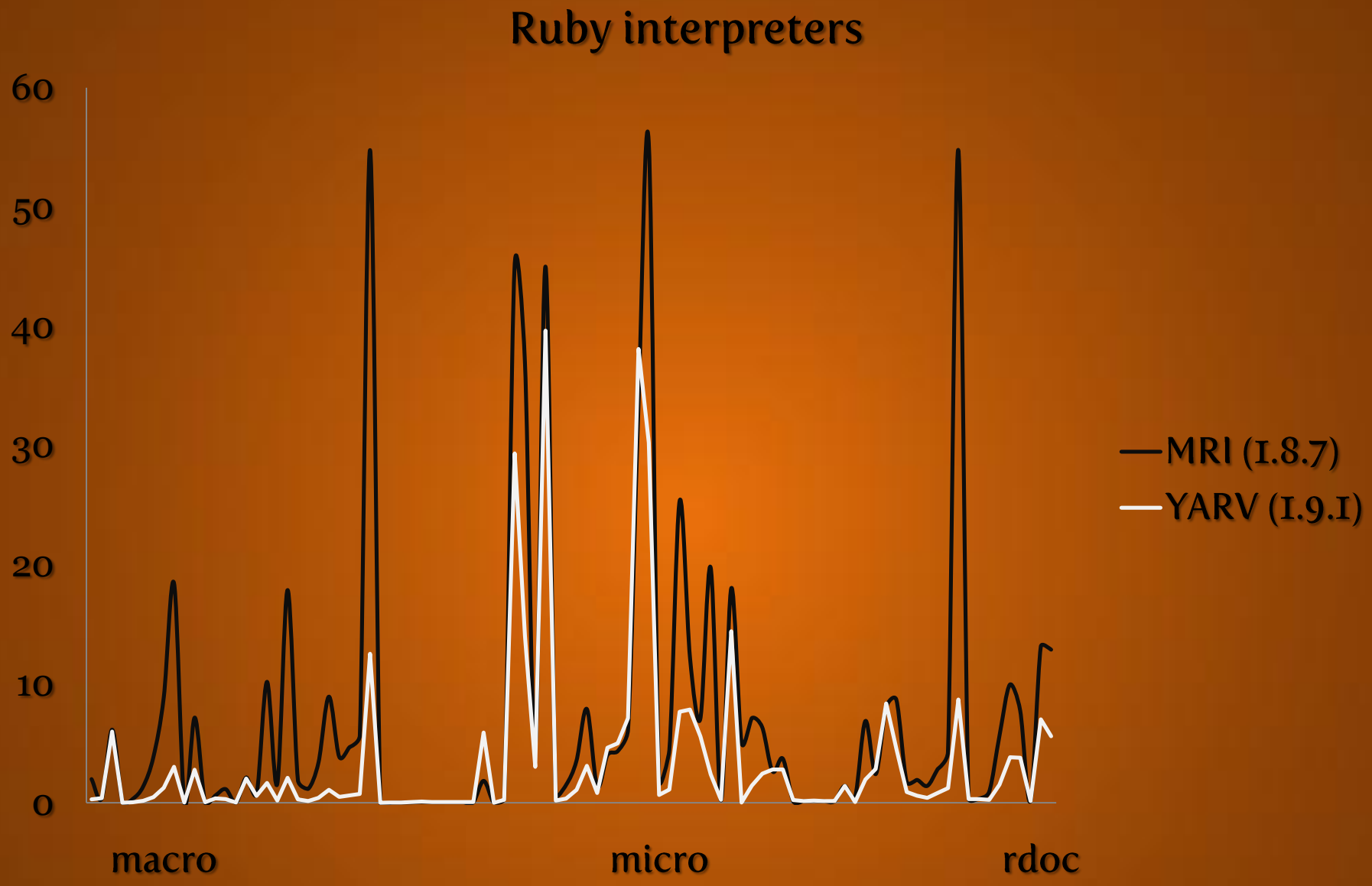
Benchmarking

MRI (1.8)

YARV (1.9)

Worth updating?

Ruby benchmarking suite


Ruby interpreters
60
50
40
30
20
10
0
macro
micro
rdoc
MRI (1.8.7)
YARV (1.9.1)

Less ~108% time on YARV overall

YARV's performance is outstanding

What does this mean, for Rails?

Web server analysis will tell

Development

Porting *escolinhas.pt* to Ruby 1.9

escolinhas.pt

> 70 models

> 130000 lines of code

> 40 plugins/gems

> 10 development branches

Rails 2.3

Porting

Straightforward

Fast

Issues

Encoding

Fix?

A small patch for Rails

Natively fixed in Rails 3

More?

Rare functionality changes

Rare syntax changes

Less than 0.1% of the time

Increasing YARV's flexibility

For adaptive performance

Tweaked MRI gave Twitter +30%

Same principle -> YARV

Configurable GC

5 configuration options

Ruby benchmarking suite

~15% less time needed overall

Slightly better memory usage

With a sample configuration which was not fine-tunned!

Improving YARV's profiler

Storing more data

Storing correct data

Retrieval

In a formatted string

Not enough

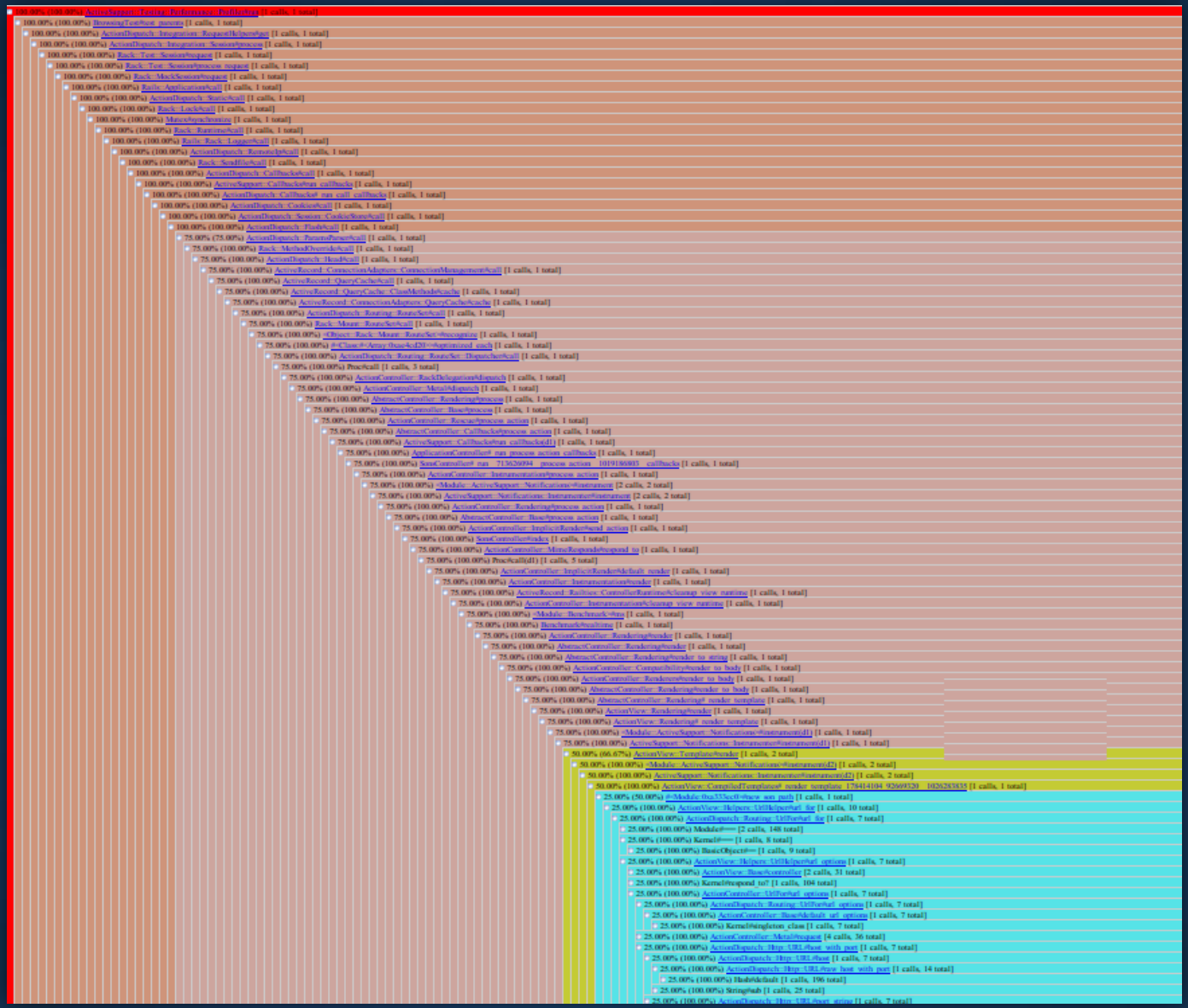
Hash

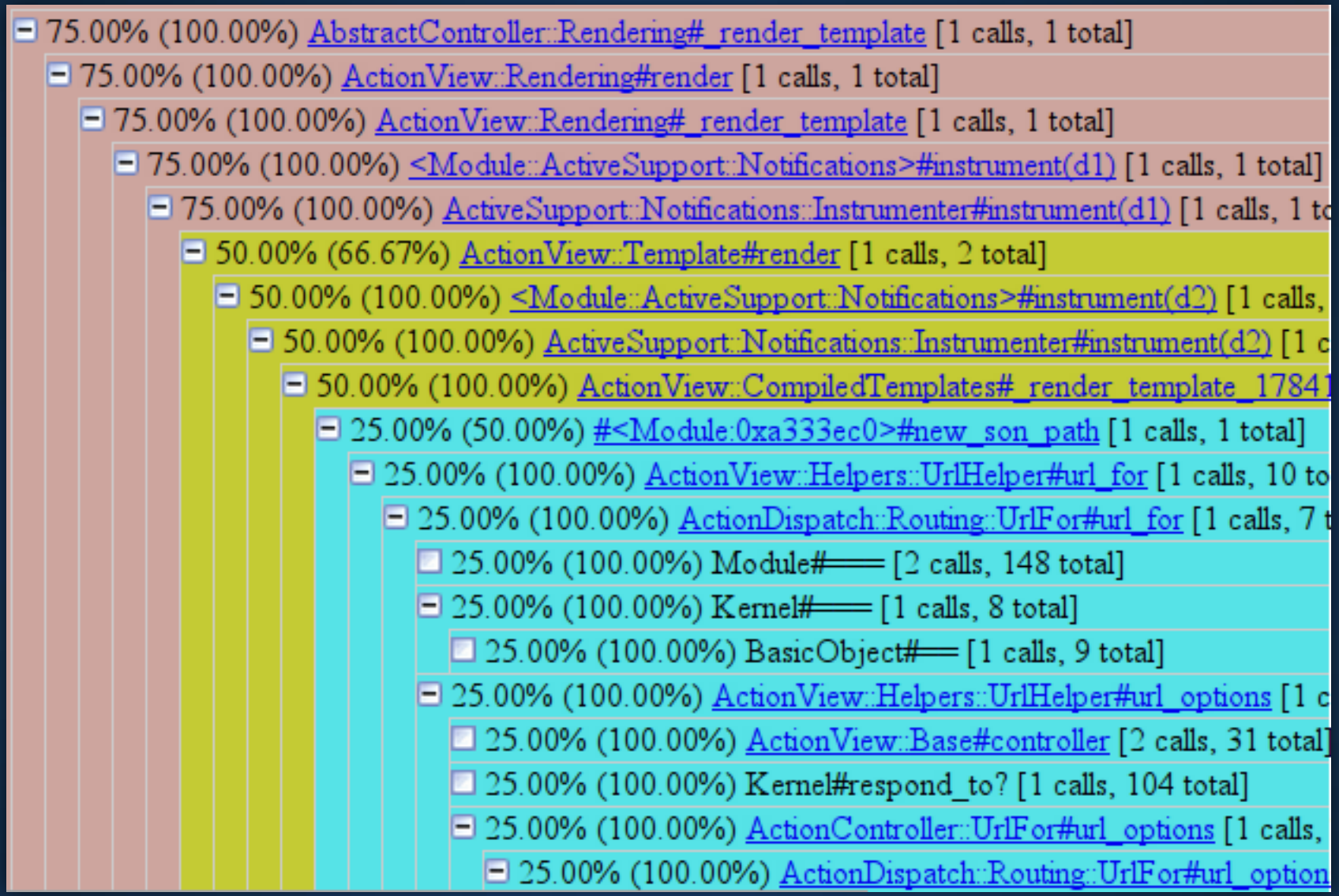
For automated processing

Hierarchical HTML viewer

For human processing

And sanity




75.00% (100.00%) AbstractController::Rendering#_render_template [1 calls, 1 total]
75.00% (100.00%) ActionView::Rendering#render [1 calls, 1 total]
75.00% (100.00%) ActionView::Rendering#_render_template [1 calls, 1 total]
75.00% (100.00%) <Module::ActiveSupport::Notifications>#instrument(d1) [1 calls, 1 total]
75.00% (100.00%) ActiveSupport::Notifications::Instrumenter#instrument(d1) [1 calls, 1 to
50.00% (66.67%) ActionView::Template#render [1 calls, 2 total]
50.00% (100.00%) <Module::ActiveSupport::Notifications>#instrument(d2) [1 calls,
50.00% (100.00%) ActiveSupport::Notifications::Instrumenter#instrument(d2) [1 c
50.00% (100.00%) ActionView::CompiledTemplates#_render_template_1784
25.00% (50.00%) #<Module:0xa333ec0>#new_son_path [1 calls, 1 total]
25.00% (100.00%) ActionView::Helpers::UrlHelper#url_for [1 calls, 10 to
25.00% (100.00%) ActionDispatch::Routing::UrlFor#url_for [1 calls, 7 t
25.00% (100.00%) Module#=== [2 calls, 148 total]
25.00% (100.00%) Kernel#=== [1 calls, 8 total]
25.00% (100.00%) BasicObject#== [1 calls, 9 total]
25.00% (100.00%) ActionView::Helpers::UrlHelper#url_options [1 c
25.00% (100.00%) ActionView::Base#controller [2 calls, 31 total]
25.00% (100.00%) Kernel#respond_to? [1 calls, 104 total]
25.00% (100.00%) ActionController::UrlFor#url_options [1 calls,
25.00% (100.00%) ActionDispatch::Routing::UrlFor#url_option

Ruby

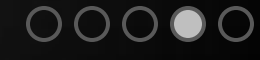
Benchmark ->

- YARV and MRI

Development ->

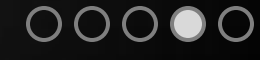
- Porting *escolinhas.pt* to Ruby 1.9
- Increasing YARV's flexibility
- Fixed and enhanced internal profiler

Rails Web Servers



Development

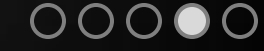
Monitoring memory usage script



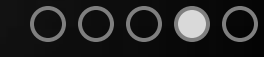
Measuring consumption over time

Very light, configurable

Proper output format (CSV)



Benchmarking



Many small benchmarks

One complete analysis

3 pages: heavy, medium, light

Proxy performance

Apache, Nginx, Cherokee

Nginx slightly faster

Nginx lighter

Passenger performance

Apache, Nginx

With Nginx it yields better results

Thin and Unicorn comparison

Similar performance

Thin is slightly lighter

Threaded Thin performance

Worse than non-threaded Thin

In-depth analysis

Passenger, Thin, Unicorn, Nginx

30 and 60 workers

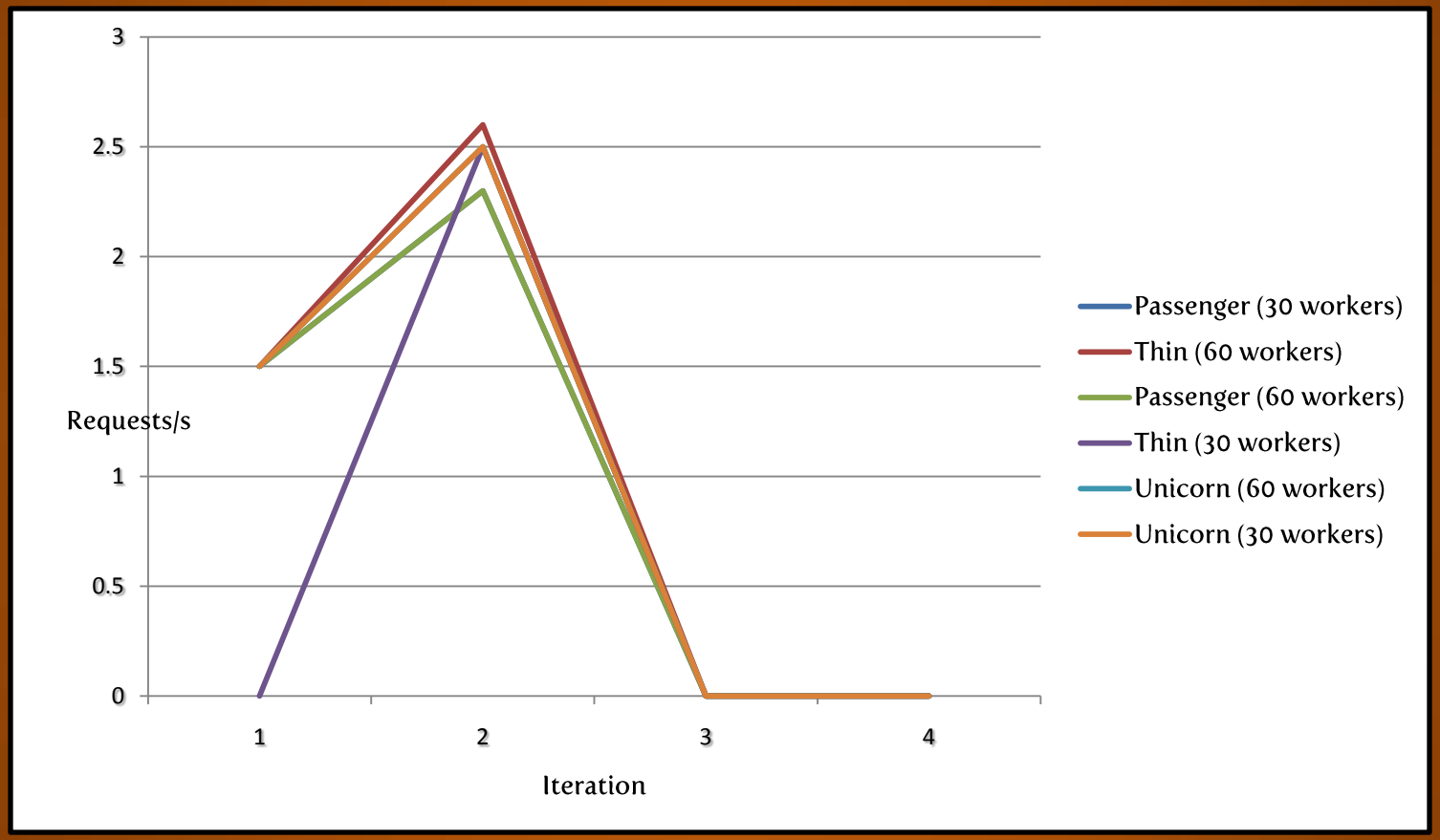
MRI and YARV

Using autobench

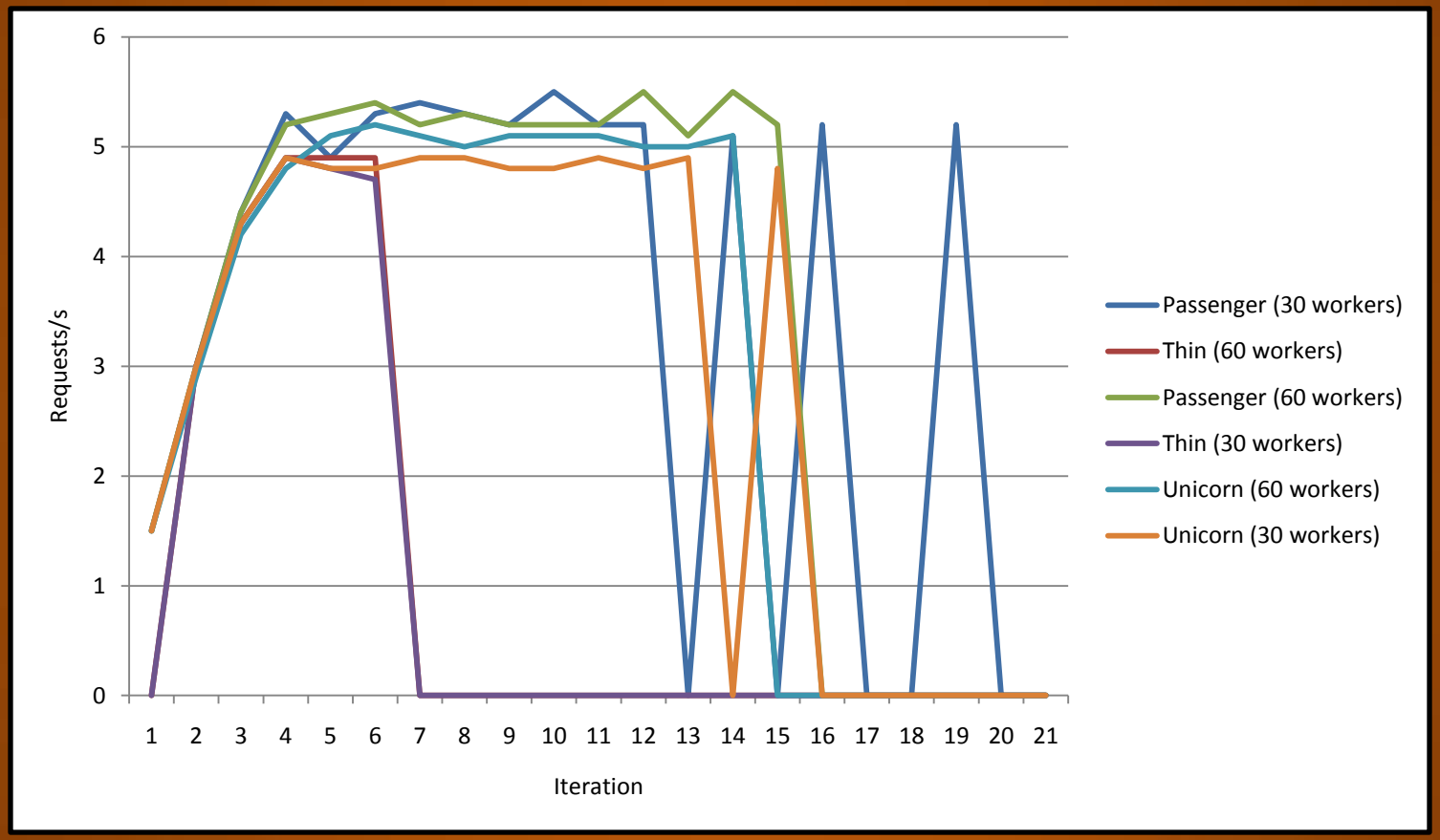
Similar performance among web servers

But...

MRI (1.8)


3
2.5
2
1.5
1
0.5
0
Requests/s
1
2
3
4
Iteration
Passenger (30 workers)
Thin (60 workers)
Passenger (60 workers)
Thin (30 workers)
Unicorn (60 workers)
Unicorn (30 workers)

YARV (1.9)


Requests/s
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
Iteration
Passenger (30 workers)
Thin (60 workers)
Passenger (60 workers)
Thin (30 workers)
Unicorn (60 workers)
Unicorn (30 workers)

1 iteration -> 15/16 interations

2,5 requests/s -> 5 requests/s

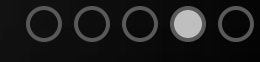
Huge improvement!

Rails Web Servers

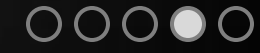
Benchmark ->

- Reverse proxies
- Passenger on Apache and Nginx
- Thin and Unicorn
- Threaded Thin
- Thin, Unicorn, Passenger and Nginx

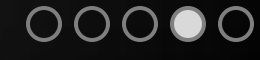
Databases



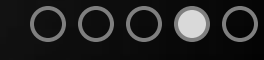
Development



mysql2 adapter

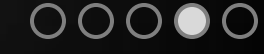


Type conversions in C



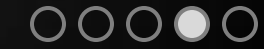
Average 400% improvement

Likely to replace mysql adapter



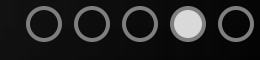
Caveat

More fields fetched than those used



Solution?

Lazy type casting

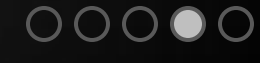


Databases

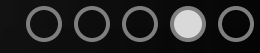
Development ->

Adding lazy type-casting to mysql2

Rails



Benchmarking



Various Rails features

Eager loading

Transactions

Magic finders

Fetching in batches

Hypothetical situations

Eager loading – 93%

Transactions – 93%

Magic finders – 68%

Fetching in batches – 10%

Development

Rails profiling tools

Working

Seamless integration with YARV

Porting renowned Rails plugins

Used by *escolinhas.pt*

acts_as_list

permalink_fu

acts_as_paranoid

Compatible with Rails 3

Compatible with Ruby 1.9

Easing the porting effort

Porting Redmine

Compatible with Rails 3

Compatible with Ruby 1.9

Highly increasing the visibility of the benefits

Ruby Summer of Code

Benchmarking Continuous Integration

Rails core team

Full-stack benchmarking suite

Automated

Smashing performance regressions

Community blog

Performance-oriented article series

Rails Magazine

Ruby on Rails

Benchmark->

Eager loading, transactions, magic finders, fetching in batches

Development ->

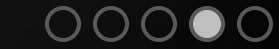
Integrating profiling tools with YARV's

Porting Redmine and plugins to Rails 3 and Ruby 1.9

Benchmarking CI for Rails

Community blog

Performance-oriented article series for Rails Magazine



Context
Problem
Approach
Results

Conclusions

General guide was created

Information from benchmarking and analyzing

Covering all system components

Publicly accessible

Profiling tools are functional

Enhanced

Seamlessly integrate

Increased global awareness

General guide

Flexible GC in YARV

Porting Redmine & Plugins

Public blog

Article series

Official benchmarking CI

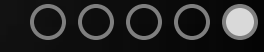
Direct improvement

YARV

mysql2

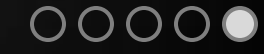
Future research

Non-relational databases



Native caching for Rails

Rewriting web servers in C

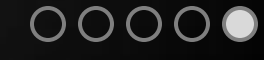


Ongoing work

Blog

Performance-oriented article series

Benchmarking CI



Thank you

Gonçalo S. Silva
Supervisor: Ademar Aguiar (PhD.)

http://goncalossilva.com
http://twitter.com/goncalossilva