Rails Performance Best Practices

http://ihower.tw
2010/3

About Me

- 張文鈿 a.k.a. ihower
  - http://ihower.tw
  - http://twitter.com/ihower
  - http://github.com/ihower
- Ruby on Rails Developer since 2006
- Ruby Taiwan Community
  - http://ruby.tw

Performance Worse Practices

- Premature Optimization
- Guessing
- Caching everything
- Fighting the framework

Performance Guidelines

- Algorithmic improvements always beat code tweaks
- As a general rule, maintainability beats performance
- Only optimize what matters (80/20 rule)
- Measure twice, cut once
- Must be balanced with flexibility

How to improve performance?

- Find the target baseline
- Know where you are now
- Profile to find bottlenecks
- Remove bottlenecks
- Repeat

Agenda

- Analysis and Measurement
- Write Efficient Ruby code
- Use REE 1.8.7 or Ruby 1.9
- Use faster Ruby Library
- Caching
- SQL and ActiveRecord
- Consider NoSQL storage
- Rack and Rails Metal
- Use HTTP server for static files
- Front-end web performance
- Use external programs or write inline C code

Finding the Problems: Analysis

Server Log Analysis

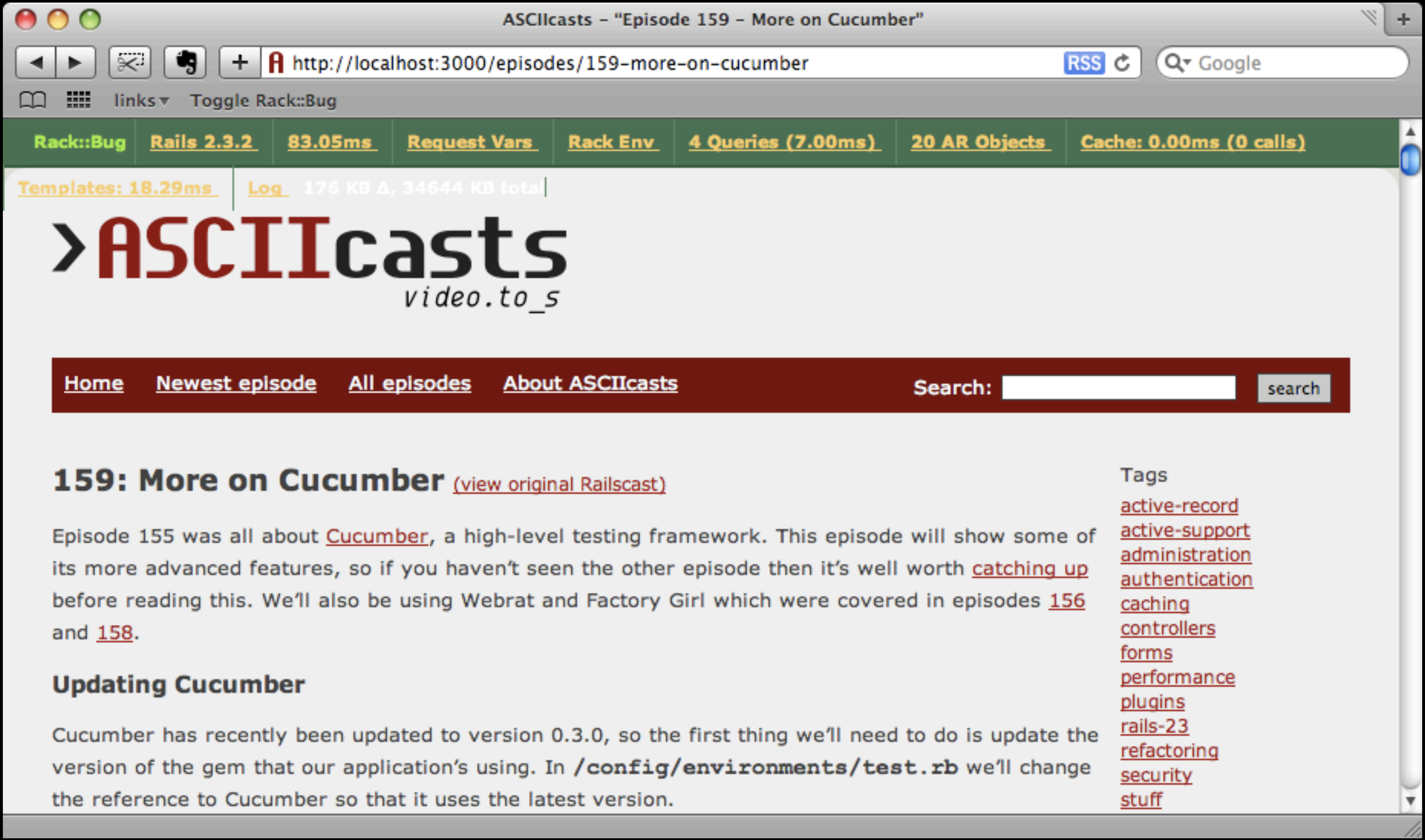
- http://github.com/wvanbergen/request-log-analyzer
  - Request distribution per hour
  - Most requested
  - HTTP methods
  - HTTP statuses returned
  - Rails action cache hits
  - Request duration
  - View rendering time
  - Database time
  - Process blockers
  - Failed requests

Commercial Monitor Products

- New Relic
- Scout

Rack::Bug

Rails middleware which gives you an informative toolbar in your browser



ASCIIcasts - "Episode 159 - More on Cucumber"

http://localhost:3000/episodes/159-more-on-cucumber

links ▾ Toggle Rack::Bug

Rack::Bug | Rails 2.3.2 | 83.05ms | Request Vars | Rack Env | 4 Queries (7.00ms) | 20 AR Objects | Cache: 0.00ms (0 calls)

Templates: 18.29ms | Log 176 KB Δ, 34644 KB total

›ASCIIcasts video.to_s

Home Newest episode All episodes About ASCIIcasts Search: search

159: More on Cucumber (view original Railscast)

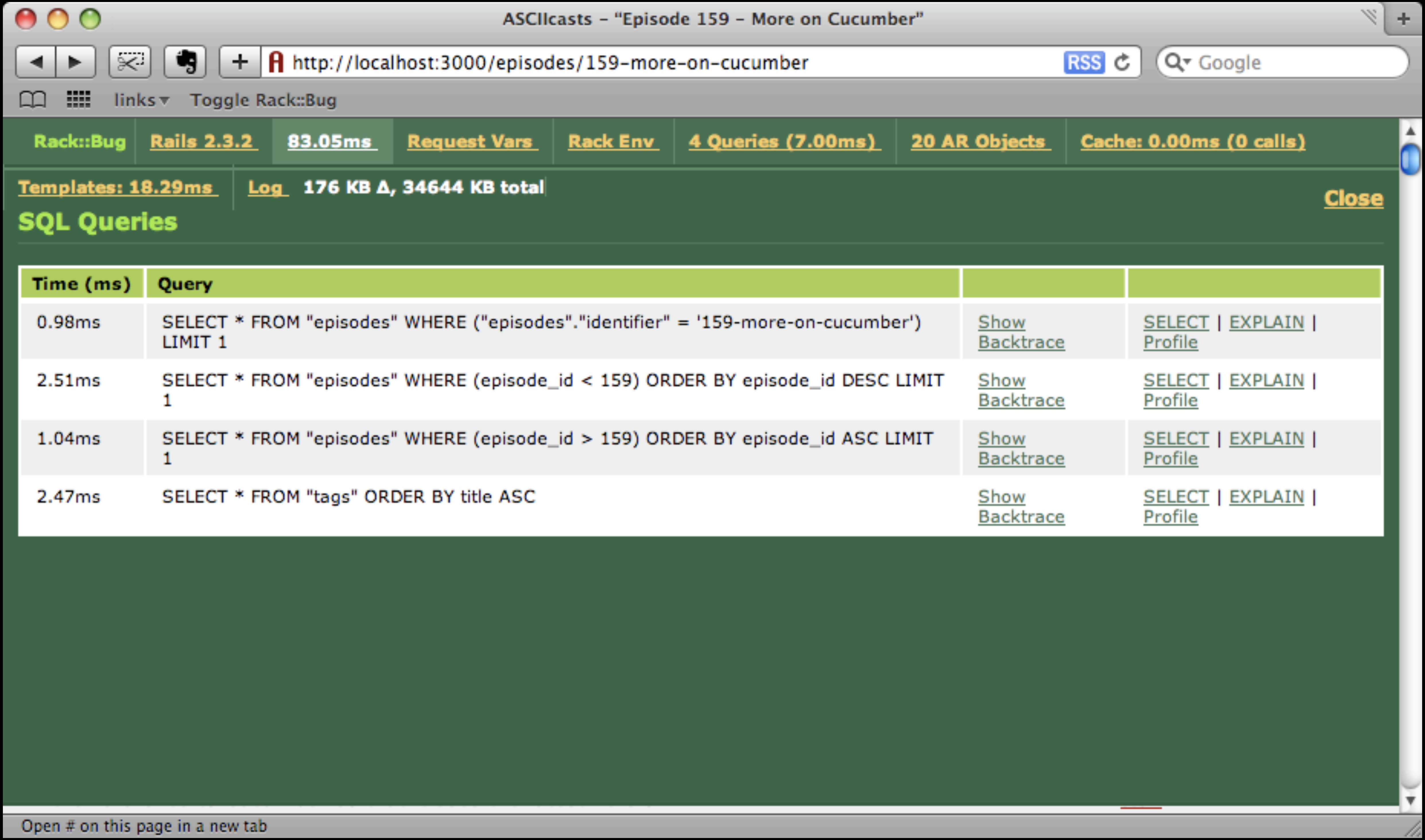
Episode 155 was all about Cucumber, a high-level testing framework. This episode will show some of its more advanced features, so if you haven't seen the other episode then it's well worth catching up before reading this. We'll also be using Webrat and Factory Girl which were covered in episodes 156 and 158.

Updating Cucumber

Cucumber has recently been updated to version 0.3.0, so the first thing we'll need to do is update the version of the gem that our application's using. In `/config/environments/test.rb` we'll change the reference to Cucumber so that it uses the latest version.

Tags

- active-record
- active-support
- administration
- authentication
- caching
- controllers
- forms
- performance
- plugins
- rails-23
- refactoring
- security
- stuff



SQL Queries

Time (ms)	Query		
0.98ms	SELECT * FROM "episodes" WHERE ("episodes"."identifier" = '159-more-on-cucumber') LIMIT 1	Show Backtrace	SELECT | EXPLAIN | Profile
2.51ms	SELECT * FROM "episodes" WHERE (episode_id < 159) ORDER BY episode_id DESC LIMIT 1	Show Backtrace	SELECT | EXPLAIN | Profile
1.04ms	SELECT * FROM "episodes" WHERE (episode_id > 159) ORDER BY episode_id ASC LIMIT 1	Show Backtrace	SELECT | EXPLAIN | Profile
2.47ms	SELECT * FROM "tags" ORDER BY title ASC	Show Backtrace	SELECT | EXPLAIN | Profile

MemoryLogic

Adds in proccess id and memory usage in your rails logs, great for tracking down memory leaks

- http://github.com/binarylogic/memorylogic

```
Processing WelcomeController#index (for 127.0.0.1 at 2010-02-26 01:51:54) [GET] (mem 104792)
  Parameters: {"action"=>"index", "controller"=>"welcome"} (mem 104792)
  ...
Memory usage: 108888 | PID: 6170 (mem 108888)
Completed in 3570ms (View: 1861, DB: 1659) | 200 OK [http://localhost/] (mem 108888)
```

oink

Log parser to identify actions which significantly increase VM heap size

- http://github.com/noahd1/oink

```
script/oink -t 0 ~/rails-app/log/development.log

---- MEMORY THRESHOLD ----
THRESHOLD: 0 MB

-- SUMMARY --
Worst Requests:
1. Feb 26 02:12:45, 37360 KB, SessionsController#new
2. Feb 26 02:12:41, 37352 KB, BooksController#hot
3. Feb 26 02:12:21, 16824 KB, BooksController#hot
4. Feb 26 02:12:25, 11632 KB, BooksController#hot
5. Feb 26 02:12:19, 11120 KB, BooksController#hot
6. Feb 26 02:12:51, 9888 KB, WelcomeController#index
7. Feb 26 02:12:28, 7548 KB, WelcomeController#index
8. Feb 26 02:12:23, 5120 KB, ArticlesController#index

Worst Actions:
4, BooksController#hot
2, WelcomeController#index
1, ArticlesController#index
1, SessionsController#new
```

ruby-prof gem

- a fast code profiler for Ruby. Its features include:
  - Speed - it is a C extension
  - Modes - call times, memory usage and object allocations.
  - Reports - can generate text and cross-referenced html reports
  - Threads - supports profiling multiple threads simultaneously
  - Recursive calls - supports profiling recursive method calls

ruby-prof example (1)

```
require 'ruby-prof'

# Profile the code
RubyProf.start
...
# code to profile
100.times { puts "blah" }

result = RubyProf.stop

# Print a flat profile to text
printer = RubyProf::FlatPrinter.new(result)
printer.print(STDOUT, 0)
```

ruby-prof example (2)

```
Thread ID: 2148368700
Total: 0.022092

 %self     total     self     wait    child    calls  name
 13.19      0.00     0.00     0.00     0.00        2  Readline#readline (ruby_runtime:0}
 12.16      0.01     0.00     0.00     0.00       79  RubyLex#getc (/opt/ruby-enterprise/lib/ruby/1.8/irb/ruby-lex.rb:101}
  4.87      0.00     0.00     0.00     0.00       30  RubyLex#ungetc (/opt/ruby-enterprise/lib/ruby/1.8/irb/ruby-lex.rb:144}
  4.52      0.00     0.00     0.00     0.00        5  RubyLex#identify_identifier (/opt/ruby-enterprise/lib/ruby/1.8/irb/ruby-lex.rb:770}
  4.41      0.00     0.00     0.00     0.00      202  IO#write (ruby_runtime:0}
  4.03      0.02     0.00     0.00     0.02       20  IRB::SLex::Node#match_io (/opt/ruby-enterprise/lib/ruby/1.8/irb/slex.rb:204}
  4.00      0.01     0.00     0.00     0.01       28  Proc#call (ruby_runtime:0}
  2.92      0.00     0.00     0.00     0.00       20  RubyToken#Token (/opt/ruby-enterprise/lib/ruby/1.8/irb/ruby-token.rb:84}
  2.85      0.00     0.00     0.00     0.00      271  String#== (ruby_runtime:0}
  2.65      0.01     0.00     0.00     0.01       14  IRB::SLex::Node#match_io(d1) (/opt/ruby-enterprise/lib/ruby/1.8/irb/slex.rb:204}
  2.55      0.02     0.00     0.00     0.02       20  RubyLex#token (/opt/ruby-enterprise/lib/ruby/1.8/irb/ruby-lex.rb:279}
  2.26      0.00     0.00     0.00     0.00      100  Kernel#puts (ruby_runtime:0}
  2.23      0.00     0.00     0.00     0.00       20  Array#& (ruby_runtime:0}
  1.72      0.02     0.00     0.00     0.02        2  RubyLex#lex (/opt/ruby-enterprise/lib/ruby/1.8/irb/ruby-lex.rb:262}
  1.71      0.02     0.00     0.00     0.02       20  IRB::SLex#match (/opt/ruby-enterprise/lib/ruby/1.8/irb/slex.rb:70}
  1.65      0.00     0.00     0.00     0.00        1  Integer#times (ruby_runtime:0}
  1.60      0.00     0.00     0.00     0.00      163  Fixnum#+ (ruby_runtime:0}
  1.56      0.00     0.00     0.00     0.00      167  Kernel#hash (ruby_runtime:0}
  1.53      0.00     0.00     0.00     0.00      147  Array#empty? (ruby_runtime:0}
  1.24      0.00     0.00     0.00     0.00       20  RubyLex#peek (/opt/ruby-enterprise/lib/ruby/1.8/irb/ruby-lex.rb:180}
  1.24      0.00     0.00     0.00     0.00       44  <Class::RubyLex>#debug? (/opt/ruby-enterprise/lib/ruby/1.8/irb/ruby-lex.rb:34}
```

Rails command line

```
# USAGE
script/performance/profiler 'Person.expensive_method(10)' [times] [flat|graph|graph_html]

# EXAMPLE
script/performance/profiler 'Item.all'
```

Performance Testing: Measurement

Benchmark standard library

```
require 'benchmark'

puts Benchmark.measure { "a"*1_000_000 }
# 1.166667   0.050000   1.216667 (  0.571355)

n = 50000
Benchmark.bm do |x|
  x.report { for i in 1..n; a = "1"; end }
  x.report { n.times do   ; a = "1"; end }
  x.report { 1.upto(n) do ; a = "1"; end }
end

#        user     system      total        real
#    1.033333   0.016667   1.016667 (  0.492106)
#    1.483333   0.000000   1.483333 (  0.694605)
#    1.516667   0.000000   1.516667 (  0.711077)
```

Rails command line

```
# USAGE
script/performance/benchmarker [times] 'Person.expensive_way' 'Person.another_expensive_way' ...

# EXAMPLE
script/performance/benchmarker 10 'Item.all' 'CouchItem.all'
```

Rails helper methods

Creating report in your log file

```
# Model
Project.benchmark("Creating project") do
  project = Project.create("name" => "stuff")
  project.create_manager("name" => "David")
  project.milestones << Milestone.find(:all)
end

# Controller
def process_projects
  self.class.benchmark("Processing projects") do
    Project.process(params[:project_ids])
    Project.update_cached_projects
  end
end

# View
<% benchmark("Showing projects partial") do %>
  <%= render :partial => @projects %>
<% end %>
```

Performance Test cases

a special type of integration tests

- script/generate performance_test welcome
- rake test:benchmark (will run 4 times)
- rake test:profile (will run 1 time)

Performance Test cases Example

```
require 'test_helper'
require 'performance_test_help'

class WelcomeTest < ActionController::PerformanceTest
  # Replace this with your real tests.
  def test_homepage
    get '/'
  end
end
```

rake test:benchmark

```
Started
BrowsingTest#test_homepage (7 ms warmup)
           wall_time: 2 ms
              memory: 414.53 KB
             objects: 2256
             gc_runs: 0
             gc_time: 0 ms
.WelcomeTest#test_homepage (4 ms warmup)
           wall_time: 2 ms
              memory: 414.53 KB
             objects: 2256
             gc_runs: 0
             gc_time: 0 ms
.
Finished in 0.908874 seconds.

10 tests, 0 assertions, 0 failures, 0 errors
```

rake test:profile

```
Started
BrowsingTest#test_homepage (6 ms warmup)
        process_time: 12 ms
              memory: 771.14 KB
             objects: 2653
.WelcomeTest#test_homepage (4 ms warmup)
        process_time: 12 ms
              memory: 771.14 KB
             objects: 2653
.
Finished in 3.49321 seconds.

6 tests, 0 assertions, 0 failures, 0 errors
```

Generic Tools

(black-box)

- httperf
- ab - Apache HTTP server benchmarking tool

How fast can this server serve requests?

- Use web server to serve static files as baseline measurement
- Do not run from the same server (I/O and CPU)
- Run from a machine as close as possible

You need know basic statistics

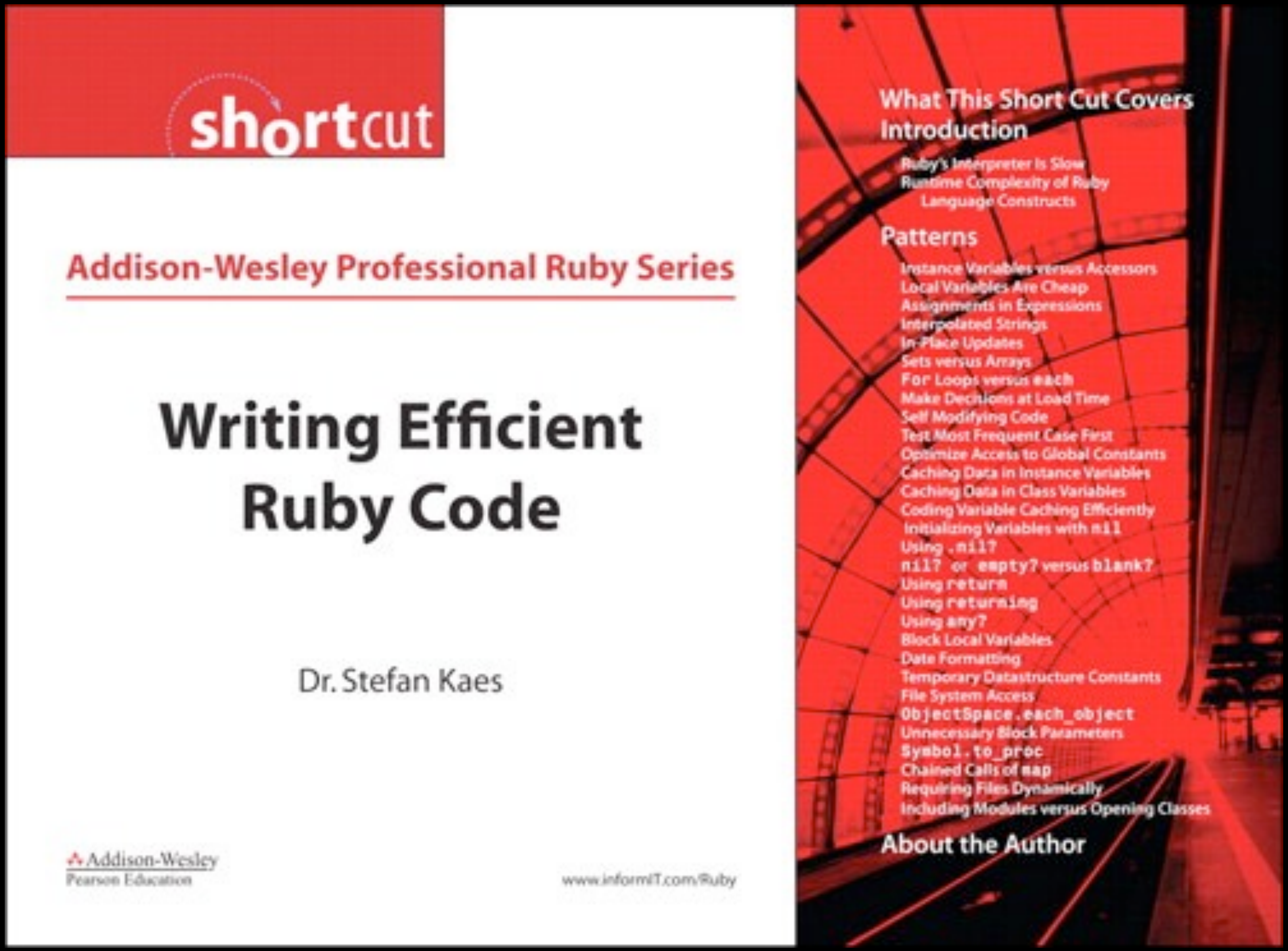
- compare not just their means but their standard deviations and confidence intervals as well.
  - Approximately 68% of the data points lie within one standard deviation of the mean
  - 95% of the data is within 2 standard deviation of the mean

httperf example

```
httperf --server localhost --port 3000 --uri / --num-conns 10000
httperf --client=0/1 --server=localhost --port=3000 --uri=/ --send-buffer=4096 --recv-buffer=16384 --num-conns=10000 --
num-calls=1

httperf: warning: open file limit > FD_SETSIZE; limiting max. # of open files to FD_SETSIZE
Maximum connect burst length: 1

Total: connections 10000 requests 10000 replies 10000 test-duration 18.373 s

Connection rate: 544.3 conn/s (1.8 ms/conn, <=1 concurrent connections)
Connection time [ms]: min 0.1 avg 1.8 max 4981.7 median 0.5 stddev 50.8
Connection time [ms]: connect 0.7
Connection length [replies/conn]: 1.000

Request rate: 544.3 req/s (1.8 ms/req)
Request size [B]: 87.0

Reply rate [replies/s]: min 55.0 avg 558.3 max 830.7 stddev 436.4 (3 samples)
Reply time [ms]: response 0.7 transfer 0.4
Reply size [B]: header 167.0 content 3114.0 footer 0.0 (total 3281.0)
Reply status: 1xx=0 2xx=10000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 3.24 system 14.09 (user 17.7% system 76.7% total 94.4%)
Net I/O: 1790.1 KB/s (14.7*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
```

1 sample need 5s
30 samples will be accurate

Writing Efficient Ruby Code



shortcut

Addison-Wesley Professional Ruby Series

Writing Efficient Ruby Code

Dr. Stefan Kaes

Addison-Wesley
Pearson Education

www.informIT.com/Ruby

What This Short Cut Covers

Introduction

- Ruby's Interpreter Is Slow
- Runtime Complexity of Ruby Language Constructs

Patterns

- Instance Variables versus Accessors
- Local Variables Are Cheap
- Assignments in Expressions
- Interpolated Strings
- In-Place Updates
- Sets versus Arrays
- `For` Loops versus `each`
- Make Decisions at Load Time
- Self Modifying Code
- Test Most Frequent Case First
- Optimize Access to Global Constants
- Caching Data in Instance Variables
- Caching Data in Class Variables
- Coding Variable Caching Efficiently
- Initializing Variables with `nil`
- Using `.nil?`
- `nil?` or `empty?` versus `blank?`
- Using `return`
- Using `returning`
- Using `any?`
- Block Local Variables
- Date Formatting
- Temporary Datastructure Constants
- File System Access
- `ObjectSpace.each_object`
- Unnecessary Block Parameters
- `Symbol.to_proc`
- Chained Calls of `map`
- Requiring Files Dynamically
- Including Modules versus Opening Classes

About the Author

Writing Efficient Ruby Code Tips

- Instance variable faster than accessor
- Interpolated string faster than + operator
- In-Place updates
- Module and class definition scope only execute once
- Caching Data in Instance or Class Variables
- Useless .nil?
- Unnecessary block parameter &block
- More...
  - http://www.igvita.com/2008/07/08/6-optimization-tips-for-ruby-mri/
  - http://ihower.tw/blog/archives/1691
  - http://en.oreilly.com/rails2009/public/schedule/detail/8680

Interpolated Strings

```
s = "#{a} #{b} #{c}"

# is faster than

s = a.to_s + b.to_s + c.to_s
```

In-Place Updates

```
Class   Copying   Destructive
------------------------------
String  #+        #<<
String  #sub      #sub!
String  #gsub     #gsub!
Hash    #merge    #merge!
Array   #+        #concat
Array   #map      #map!
Array   #compact  #compact!
Array   #uniq     #uniq!
Array   #flatten  #flatten!
```

Coding Variable Caching (1)

```ruby
def capital_letters
  @capital_lettters ||= ("A".."Z").to_a
end

# or

@@capital_lettters ||= ("A".."Z").to_a
def capital_letters
  @@capital_lettters
end
```

Coding Variable Caching (2)

```ruby
def actions
  unless @actions
    # do something expensive
      @actions = expr
  end

  @actions
end

# or

def actions
  @actions ||=
    begin
      # do something expensive
      expr
    end
end
```

ActiveSupport::Memoizable

```ruby
def total_amount
  # expensive calculate
end

def total_income
  # expensive calculate
end

def total_expense
  # expensive calculate
end

extend ActiveSupport::Memoizable
memoize :total_amount, :total_income, :total_expense
```

Method cache

- Ruby use cached method before method lookup
- Avoid these methods at runtime, it will clear cache.
  - def / undef
  - Module#define_method
  - Module#remove_method
  - alias / Module#alias_method
  - Object#extend
  - Module#include
  - public/private/protected/module_function

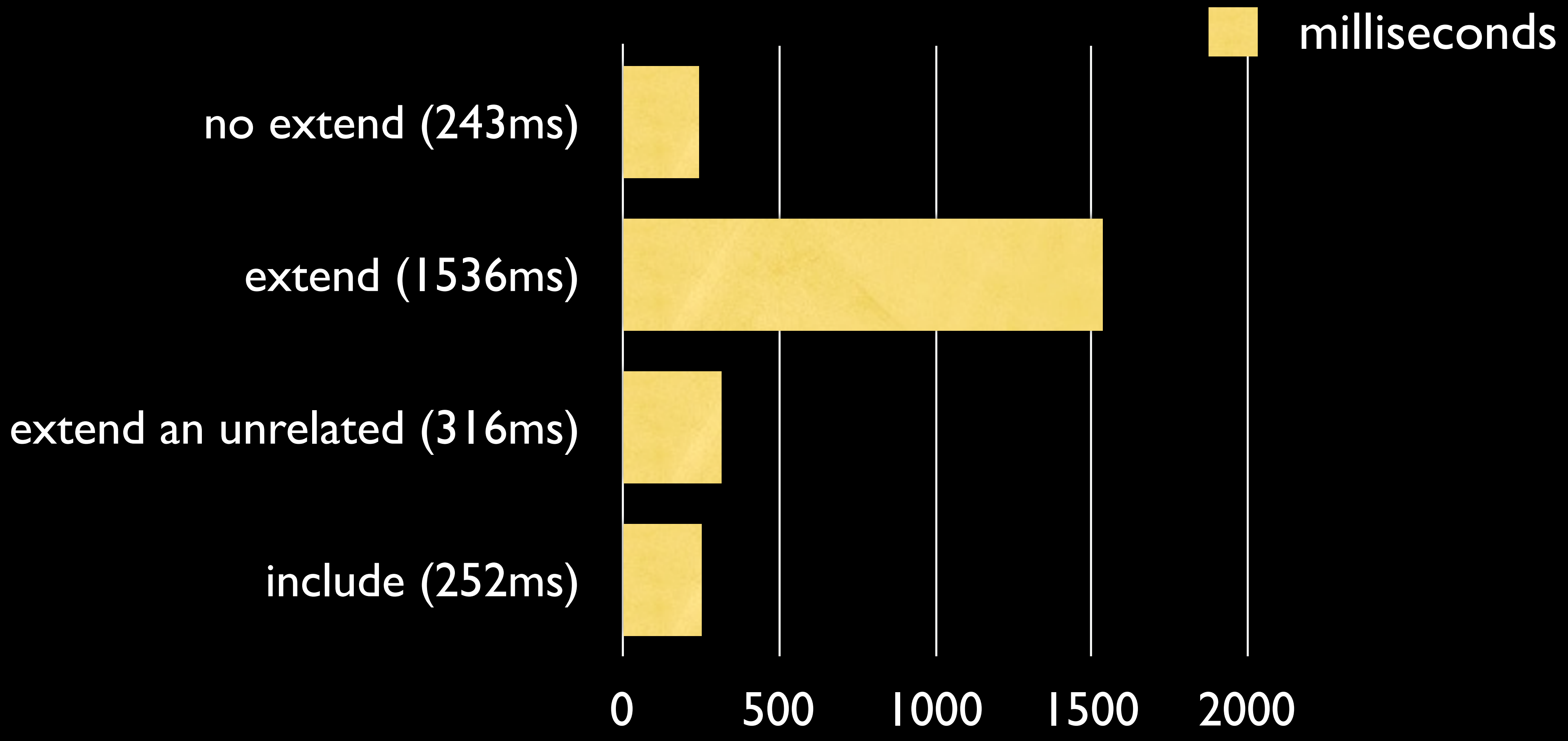
```ruby
require 'benchmark'

class C
  def m; end
end

module H
end

puts Benchmark.measure {
    i = 0
    while i < 100000
      i+=1
      l = C.new
      # l.extend H
      l.m; l.m; l.m; l.m;
      l.m; l.m; l.m; l.m;
    end
}
```

Example from “What Makes Ruby Go? An Implementation Primer (RailsConf 2009)”

```ruby
require 'benchmark'

class C
  def m; end
end

module H
end

puts Benchmark.measure {
    i = 0
    while i < 100000
      i+=1
      l = C.new
      l.extend H
      l.m; l.m; l.m; l.m;
      l.m; l.m; l.m; l.m;
    end
}
```

```ruby
require 'benchmark'

class C
  def m; end
end

module H
end

puts Benchmark.measure {
    i = 0
    x = C.new
    while i < 100000
      i+=1
      l = C.new
      x.extend H # Extend on an unrelated object!
      l.m; l.m; l.m; l.m;
      l.m; l.m; l.m; l.m;
    end
}
```

```ruby
require 'benchmark'

class C
  def m; end
end

module H
end

class MyC < C
  include H
end

puts Benchmark.measure {
    i = 0
    while i < 100000
      i+=1
      l = MyC.new
      l.m; l.m; l.m; l.m;
      l.m; l.m; l.m; l.m;
    end
}
```

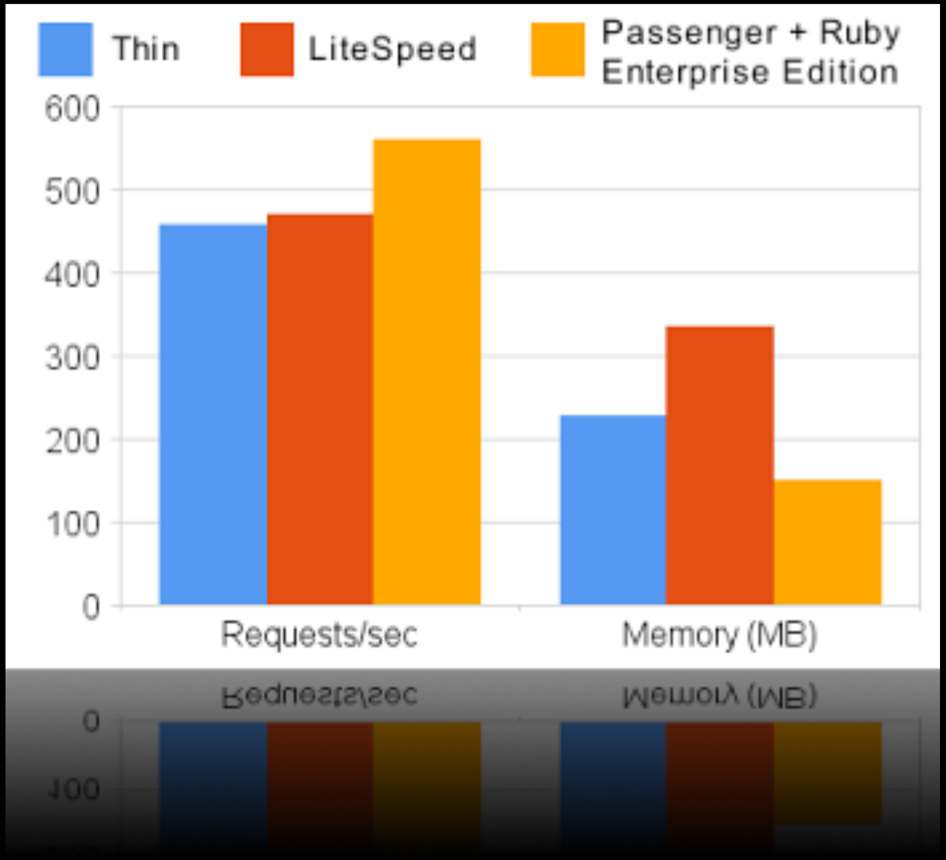


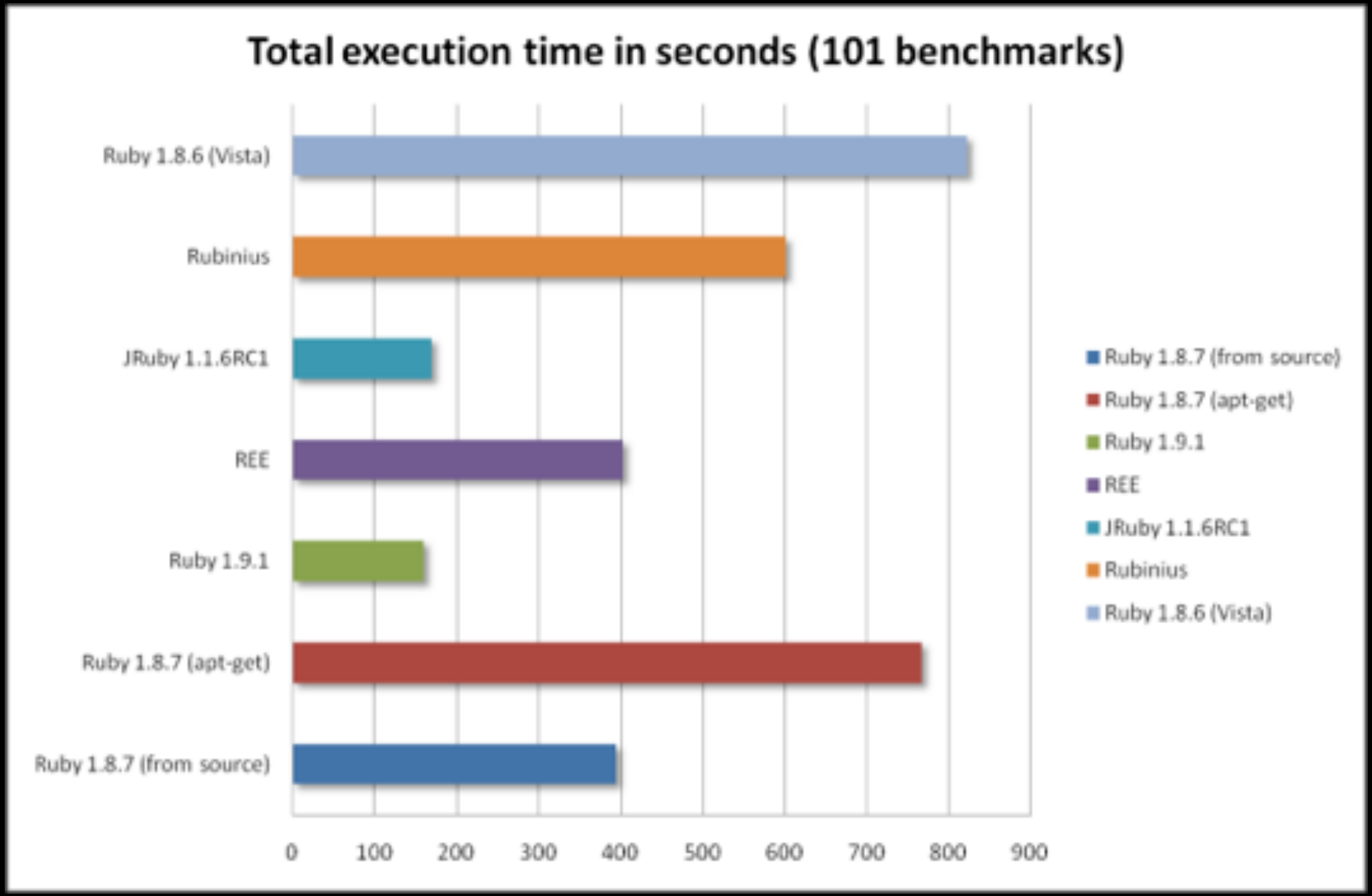
(MacBook Pro 2.2G)

Constant Caching

- Don’t redefined constants at runtime
- Don’t define new constants frequently

Use Ruby Enterprise Edition (REE) or Ruby 1.9





Ruby is slow?

- Language Micro-benchmarks != performance in complex systems
- Other factors:
  - application architecture
  - the ability to leverage higher-level
    (Simplify things which may be complex to implement in other languages.
    => No code is faster than no code.)
  - Rails is faster than many PHP frameworks
    http://avnetlabs.com/php/php-framework-comparison-benchmarks

Use faster Ruby Library

C Extension++

- XML parser
  http://nokogiri.org/
- JSON parser
  http://github.com/brianmario/yajl-ruby/
- CSV parser
  http://www.toastyapps.com/excelsior/
- HTTP client
  http://github.com/pauldix/typhoeus
- Date
  http://github.com/rtomayko/date-performance

Caching

About Caching

- Ugly if you cache everywhere
- More bugs and tough to debug
  includeing stale, data, inconsistent data, timing-based bugs.
- Complicated: expire, security
- Limit your user interface options

Cache Store

```
ActionController::Base.cache_store = :mem_cache_store, "localhost"
ActionController::Base.cache_store = :compressed_mem_cache_store, "localhost"

ActionController::Base.cache_store = :memory_store
ActionController::Base.cache_store = :synchronized_memory_store

ActionController::Base.cache_store = :file_store, "/path/to/cache/directory"
ActionController::Base.cache_store = :drb_store, "druby://localhost:9192"
```

View Caching

- Page Caching
- Action Caching
- Fragment Caching

Page Caching

```ruby
class ProductsController < ActionController

  caches_page :index

  def index; end

end
```

Action Caching

```
class ProductsController < ActionController

  before_filter :authenticate, :only => [ :edit, :create ]
  caches_action :edit

  def index; end

  def create
    expire_page :action => :index
    expire_action :action => :edit
  end

  def edit; end

end
```

Fragment Caching

```
<% cache(:key =>
  ['all_available_products', @latest_product.created_at].join(':')) do %>
  All available products:
<% end %>

expire_fragment(:key =>
  ['all_available_products', @latest_product.created_at].join(':'))
```

Sweepers

```
class StoreSweeper < ActionController::Caching::Sweeper
  # This sweeper is going to keep an eye on the Product model
  observe Product

  # If our sweeper detects that a Product was created call this
  def after_create(product)
          expire_cache_for(product)
  end

  # If our sweeper detects that a Product was updated call this
  def after_update(product)
          expire_cache_for(product)
  end

  # If our sweeper detects that a Product was deleted call this
  def after_destroy(product)
          expire_cache_for(product)
  end

  private
  def expire_cache_for(record)
    # Expire the list page now that we added a new product
    expire_page(:controller => '#{record}', :action => 'list')

    # Expire a fragment
    expire_fragment(:controller => '#{record}',
      :action => 'recent', :action_suffix => 'all_products')
  end
end
```

Caching yourself

```
Rails.cache.read("city")   # => nil
Rails.cache.write("city", "Duckburgh")
Rails.cache.read("city")   # => "Duckburgh"

Rails.cache.fetch("#{id}-data") do
    Book.sum(:amount, :conditions => { :category_id => self.category_ids } )
end
```

Use Memcached

- Free & open source, high-performance, distributed memory object caching system
- an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

Memcached

- Key: 256 characters
- Data: 1mb
- SET/ADD/REPLACE/GET operators
- NOT persistent data store
- caching “noreply” principle

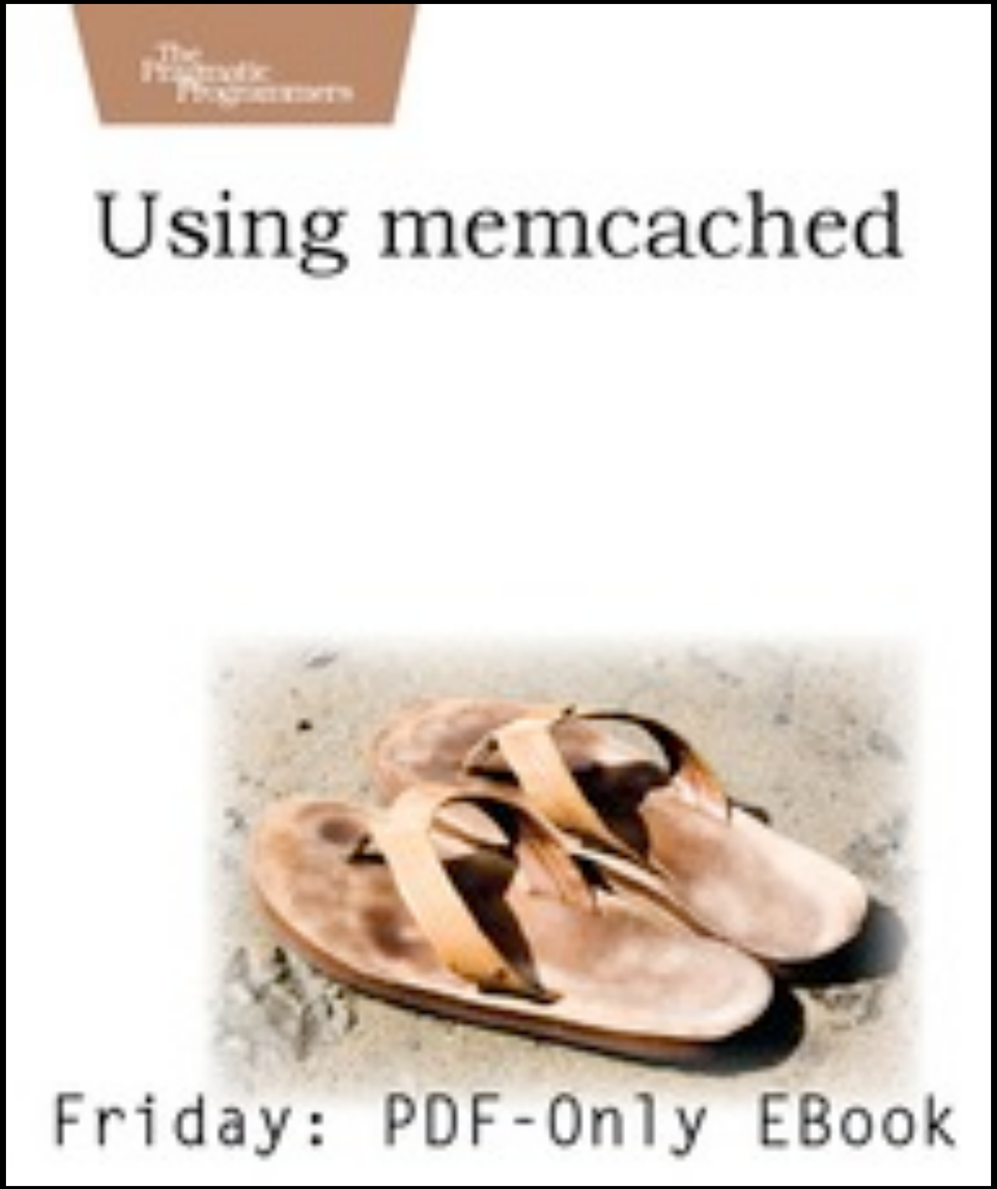
Caching secret

- Key naming
- Expiration

Caching Expire

- expire it after create/update/delete
  - race condition? lock it first.
- reset it after update
  - race condition first time? lock it first.
- set expire time
  - race condition? proactive cache refill

Using memcached


The Pragmatic Programmers

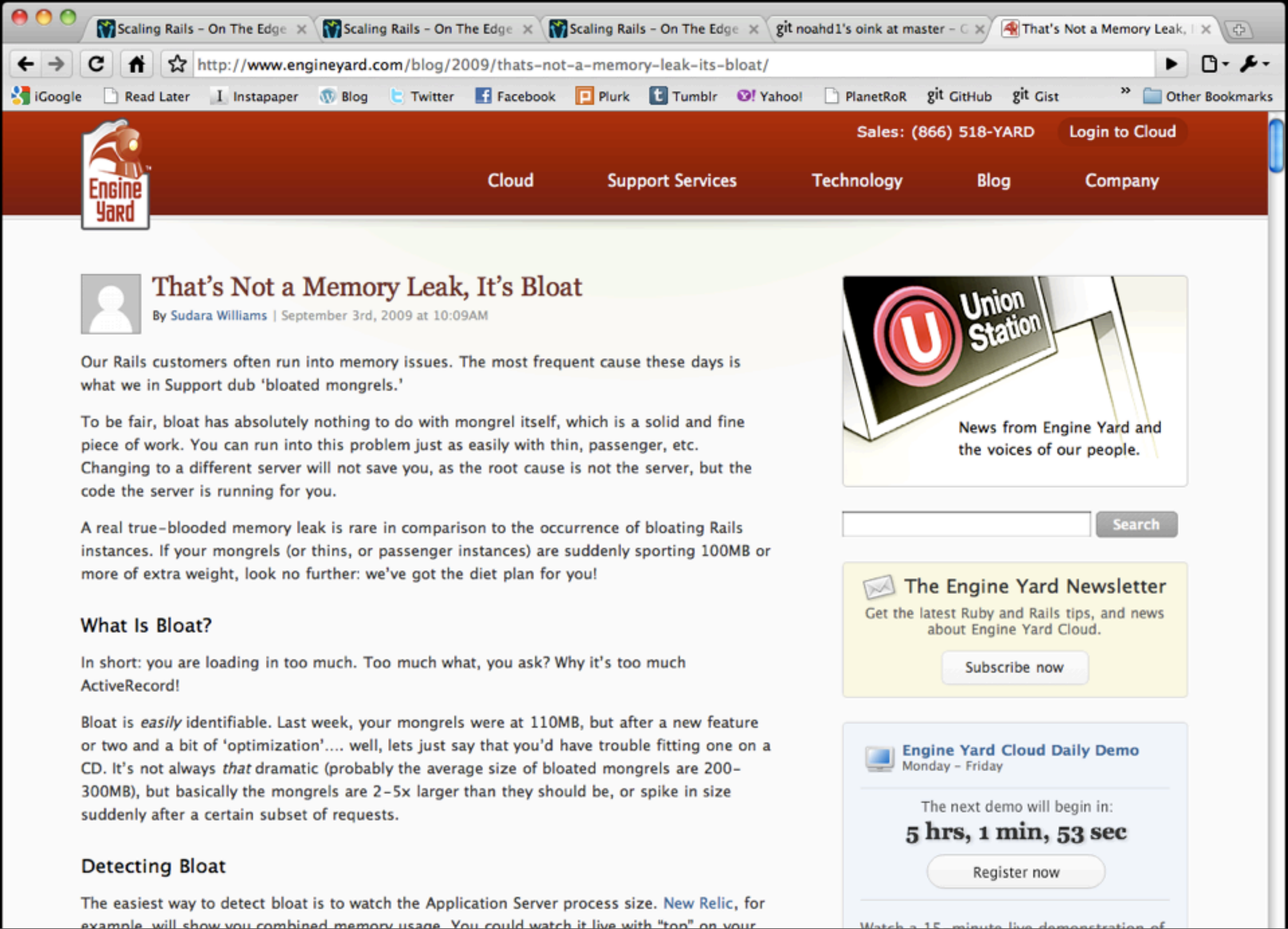
Using memcached

Friday: PDF-Only EBook

SQL and ActiveRecord

ORM is a high-level library that it's easy to forget about efficiency until it becomes a problem.

http://www.engineyard.com/blog/2009/thats-not-a-memory-leak-its-bloat/


Scaling Rails - On The Edge
Scaling Rails - On The Edge
Scaling Rails - On The Edge
git noahd1's oink at master
That's Not a Memory Leak,
http://www.engineyard.com/blog/2009/thats-not-a-memory-leak-its-bloat/
iGoogle
Read Later
Instapaper
Blog
Twitter
Facebook
Plurk
Tumblr
Yahoo!
PlanetRoR
GitHub
Gist
Other Bookmarks
Engine Yard
Sales: (866) 518-YARD
Login to Cloud
Cloud
Support Services
Technology
Blog
Company
Union Station
News from Engine Yard and the voices of our people.
Search
The Engine Yard Newsletter
Get the latest Ruby and Rails tips, and news about Engine Yard Cloud.
Subscribe now
Engine Yard Cloud Daily Demo
Monday - Friday
The next demo will begin in:
5 hrs, 1 min, 53 sec
Register now

That's Not a Memory Leak, It's Bloat

By Sudara Williams | September 3rd, 2009 at 10:09AM

Our Rails customers often run into memory issues. The most frequent cause these days is what we in Support dub 'bloated mongrels.'

To be fair, bloat has absolutely nothing to do with mongrel itself, which is a solid and fine piece of work. You can run into this problem just as easily with thin, passenger, etc. Changing to a different server will not save you, as the root cause is not the server, but the code the server is running for you.

A real true-blooded memory leak is rare in comparison to the occurrence of bloating Rails instances. If your mongrels (or thins, or passenger instances) are suddenly sporting 100MB or more of extra weight, look no further: we've got the diet plan for you!

What Is Bloat?

In short: you are loading in too much. Too much what, you ask? Why it's too much ActiveRecord!

Bloat is *easily* identifiable. Last week, your mongrels were at 110MB, but after a new feature or two and a bit of 'optimization'.... well, lets just say that you'd have trouble fitting one on a CD. It's not always *that* dramatic (probably the average size of bloated mongrels are 200-300MB), but basically the mongrels are 2-5x larger than they should be, or spike in size suddenly after a certain subset of requests.

Detecting Bloat

The easiest way to detect bloat is to watch the Application Server process size. New Relic, for

N+1 Queries

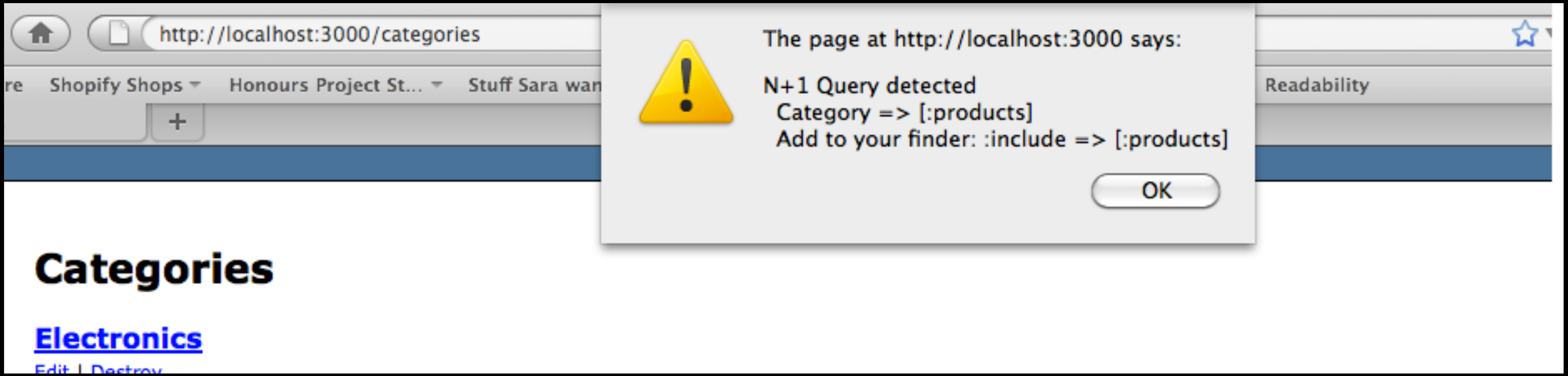
```
# model
class User < ActieRecord::Base
    has_one :car
end

class Car < ActiveRecord::Base
    belongs_to :user
end

# your controller
def index
  @users = User.paginate( :page => params[:page], :per_page => 20 )
end

# view
<% @users.each do |user| %>
   <%= user.car.name %>
<% end %>
```

Add :include

```
# your controller
def index
  @users = User.paginate( :include => :car, :page => params
[:page], :per_page => 20 )
end
```

Bullet plugin

- http://github.com/flyerhzm/bullet
  Help you reduce the number of queries with alerts (and growl).



The page at http://localhost:3000 says:

N+1 Query detected
  Category => [:products]
  Add to your finder: :include => [:products]

Categories

Electronics

Edit | Destroy

Missing indexing

- Foreign key indexs
- Columns that need to be sorted
- Lookup fields
- Columns that are used in a GROUP BY
- http://github.com/eladmeidar/rails_indexes
  Rake tasks to find missing indexes.

Only select you need

```
Event.find(:all, :select => "id, title, description")

class User < ActiveRecord::Base
    named_scope :short, :select => "id, name, email"
end

User.short.find(:all)
```

- http://github.com/methodmissing/scrooge SQL query optimizer, so you query for only what your page needs.

Replace :include to :join for some case

```
Group.find(:all, :include => [ :group_memberships ], :conditions =>
[ "group_memberships.created_at > ?", Time.now - 30.days ] )

# you can replace :include to :join

Group.find(:all, :joins => [ :group_memberships ], :conditions =>
[ "group_memberships.created_at > ?", Time.now - 30.days ] )
```

Batch finding

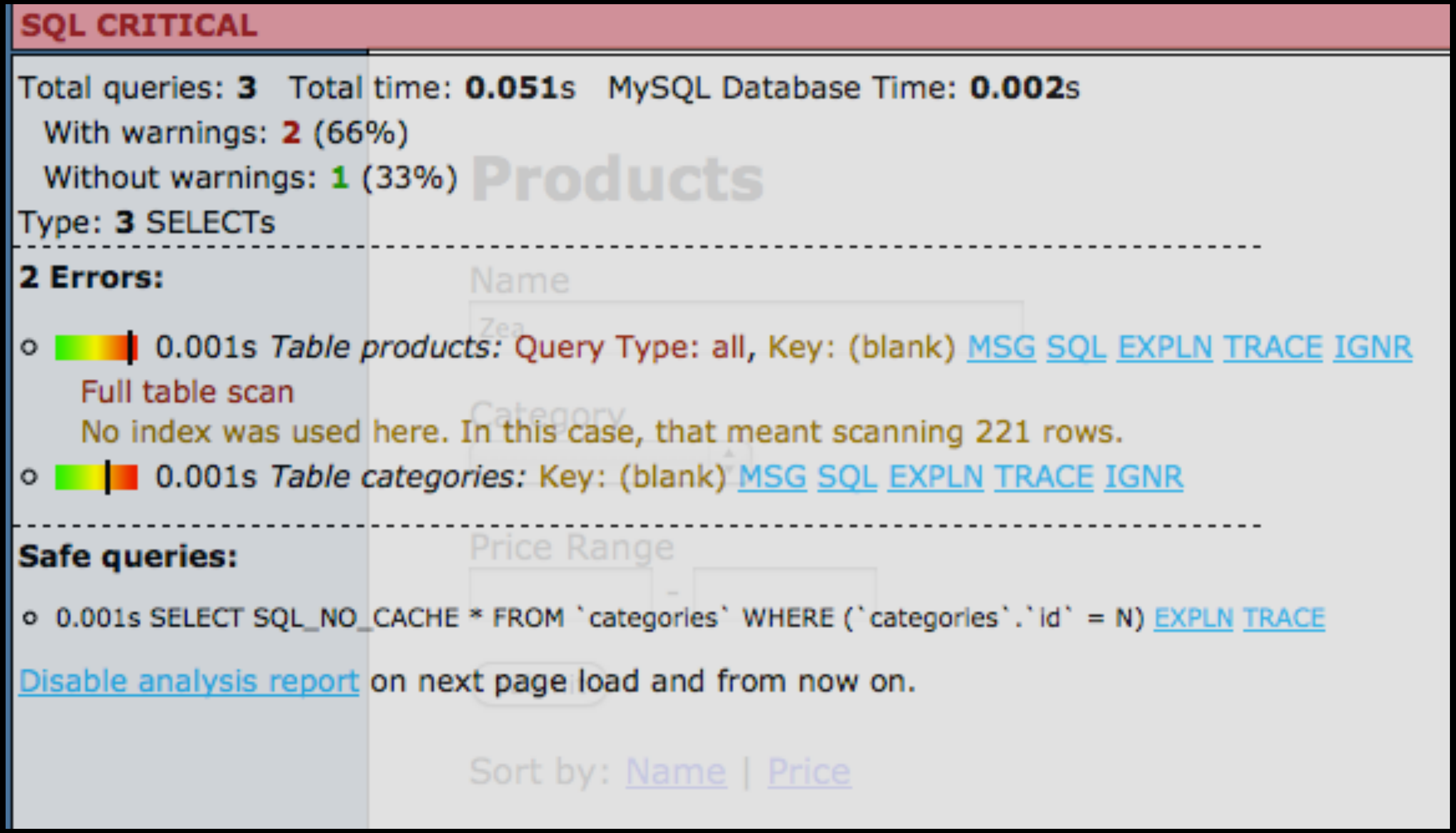
```
Article.find_each do |a|
    # iterate over all articles, in chunks of 1000 (the default)
end

Article.find_each(:conditions => { :published => true }, :batch_size => 100 ) do |a|
  # iterate over published articles in chunks of 100
end

Article.find_in_batches do |articles|
    articles.each do |a|
      # articles is array of size 1000
    end
end

Article.find_in_batches(batch_size => 100 ) do |articles|
    articles.each do |a|
      # iterate over all articles in chunks of 100
    end
end
```

Transaction for group operations

```
my_collection.each do |q|
   Quote.create({:phrase => q})
end

# Add transaction
Quote.transaction do
 my_collection.each do |q|
   Quote.create({:phrase => q})
 end
end
```

SQL query planner

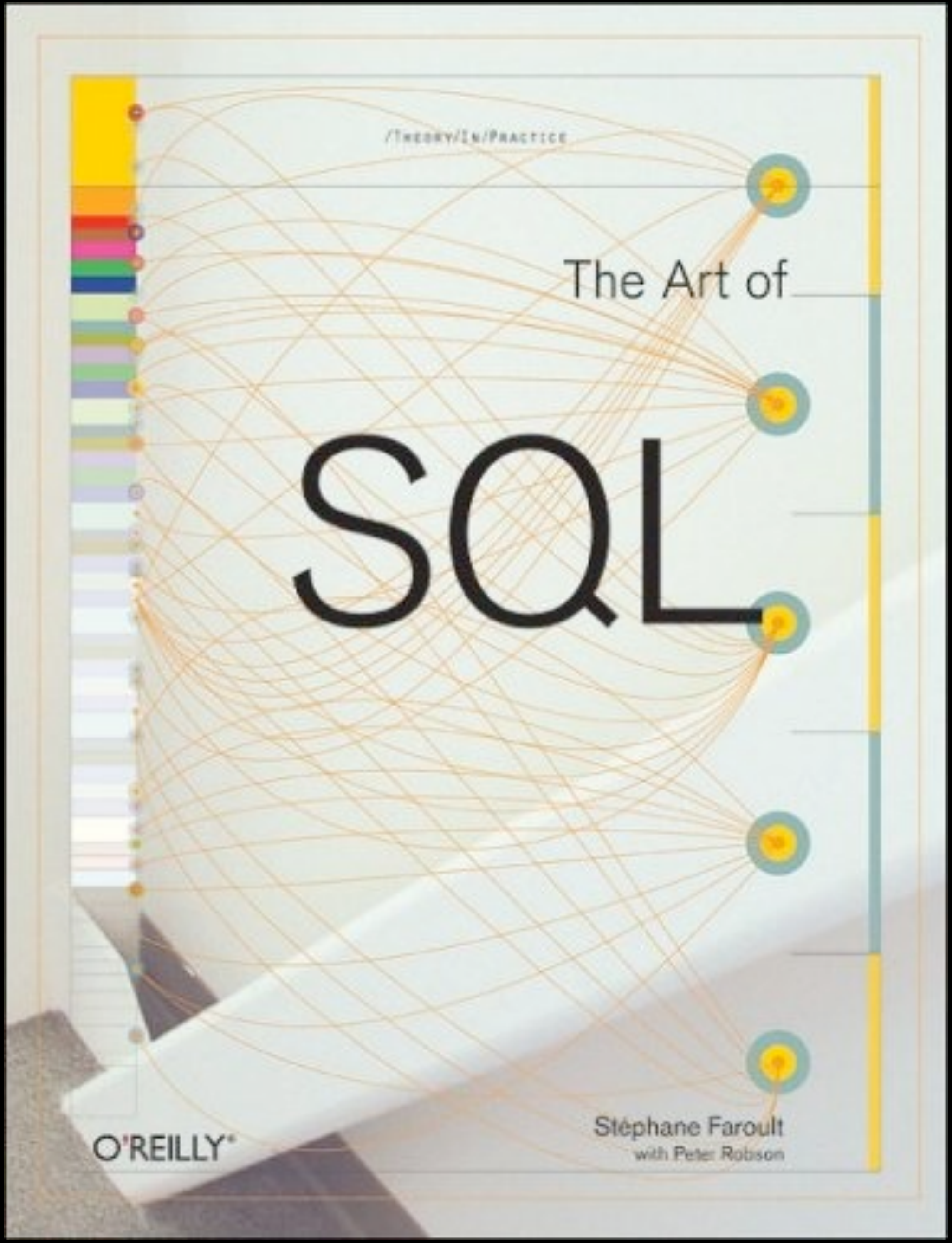
- EXPLAIN keyword
- http://github.com/dsboulder/query_reviewer



SQL CRITICAL

```
Total queries: 3   Total time: 0.051s   MySQL Database Time: 0.002s
  With warnings: 2 (66%)
  Without warnings: 1 (33%)
Type: 3 SELECTs
-------------------------------------------------------------------------
2 Errors:

o 0.001s Table products: Query Type: all, Key: (blank) MSG SQL EXPLN TRACE IGNR
    Full table scan
    No index was used here. In this case, that meant scanning 221 rows.
o 0.001s Table categories: Key: (blank) MSG SQL EXPLN TRACE IGNR
-------------------------------------------------------------------------
Safe queries:

o 0.001s SELECT SQL_NO_CACHE * FROM `categories` WHERE (`categories`.`id` = N) EXPLN TRACE

Disable analysis report on next page load and from now on.
```

SQL best practices



Use Full-text search engine

- Sphinx and thinking_sphinx plugin
- Ferret gem and acts_as_ferret

Use Constant for domain data

```
class Rating < ActiveRecord::Base

    G  = Rating.find_by_name('G')
    PG = Rating.find_by_name('PG')
    R  = Rating.find_by_name('R')
    #....

end

Rating::G
Rating::PG
Rating::R
```

Counter cache

```
class Topic < ActiveRecord::Base
    has_many :posts
end

class Posts < ActiveRecord::Base
    belongs_to :topic, :counter_cache => true
end

@topic.posts.size
```

Store Your Reports

- Aggregate reports via cron and rake

AR Caching Plugins

- http://github.com/nkallen/cache-money
- http://github.com/fauna/interlock
- http://github.com/defunkt/cache_fu
- Need careful to go to these solution, because it’s very intrusive.

Consider NoSQL data store

- Key-value stores for high performance
  Redis, Tokyo Cabinet, Flare
- Document stores for huge storage
  MongoDB, CouchDB
- Record store for high scalability and availability
  Cassandra, HBase, Voldemort

Use key-value store from now

- Redis, Tokyo Cabinet are very very fast
- Avoid touching the RDBMS when storing non-critical data, hit count, download count, online users count...etc

Moneta

- a unified interface to key/value stores
- http://github.com/wycats/moneta/tree

moneta supports:

- File store for xattr
- Basic File Store
- Memcache store
- In-memory store
- The xattrs in a file system
- DataMapper
- S3
- Berkeley DB
- Redis
- SDBM
- Tokyo
- CouchDB

moneta API:

- #[](key)
- #[]=(key, value)
- #delete(key)
- #key?(key)
- #store(key, value, options)
- #update_key(key, options):
- #clear

example

```
begin
  # for developer has tokyo cabinet
  STORE = Moneta::Tyrant.new( :host => 'localhost', :port => 1978 )
rescue
  # for developer has not tokyo cabinet
  STORE = Moneta::BasicFile.new( :path => "tmp" )
end

STORE["mykey"] = { :foo => 111 , :bar => 222 }
```

Rails Metal

a subset of Rack middleware

- 2~3x faster than a controller, because it bypasses routes and controller.
- APIs and anything which need not ActionView

Use web server or CDN for static file

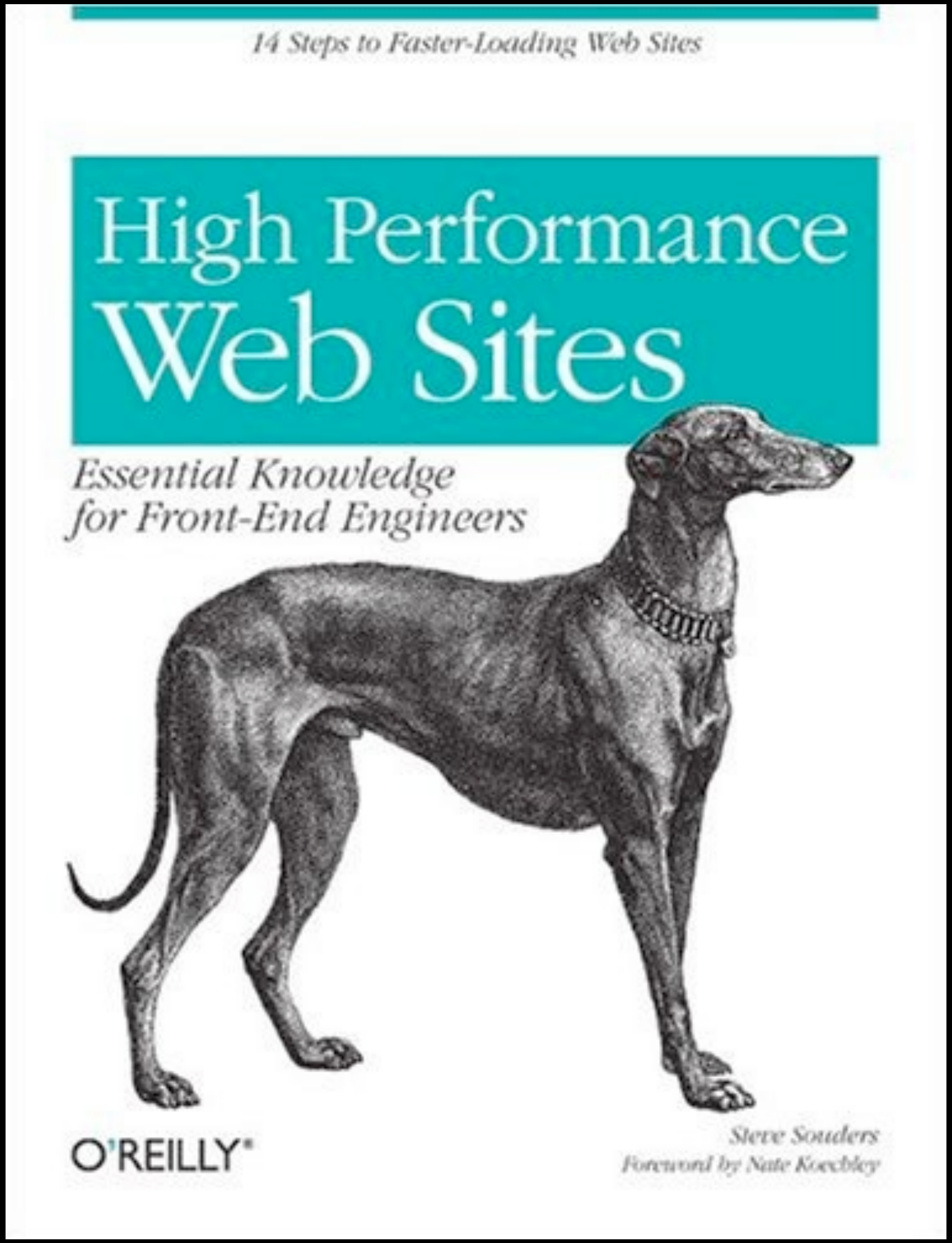
- Web server is 10x faster than your application server
- Set :x_sendfile to true if you use Apache mod_xsendfile or Lighttpd

Web performance client-side analysis

- http://developer.yahoo.com/yslow/
- http://code.google.com/speed/page-speed/

Web performance Rules 14

- Make Fewer HTTP Requests
- Use a Content Delivery Network
- Add an Expires Header
- Gzip Components
- Put Stylesheets at the Top
- Put Scripts at the Bottom
- Avoid CSS Expressions
- Make JavaScript and CSS External
- Reduce DNS Lookups
- Minify JavaScript
- Avoid Redirects
- Remove Duplicates Scripts
- Configure ETags
- Make Ajax Cacheable



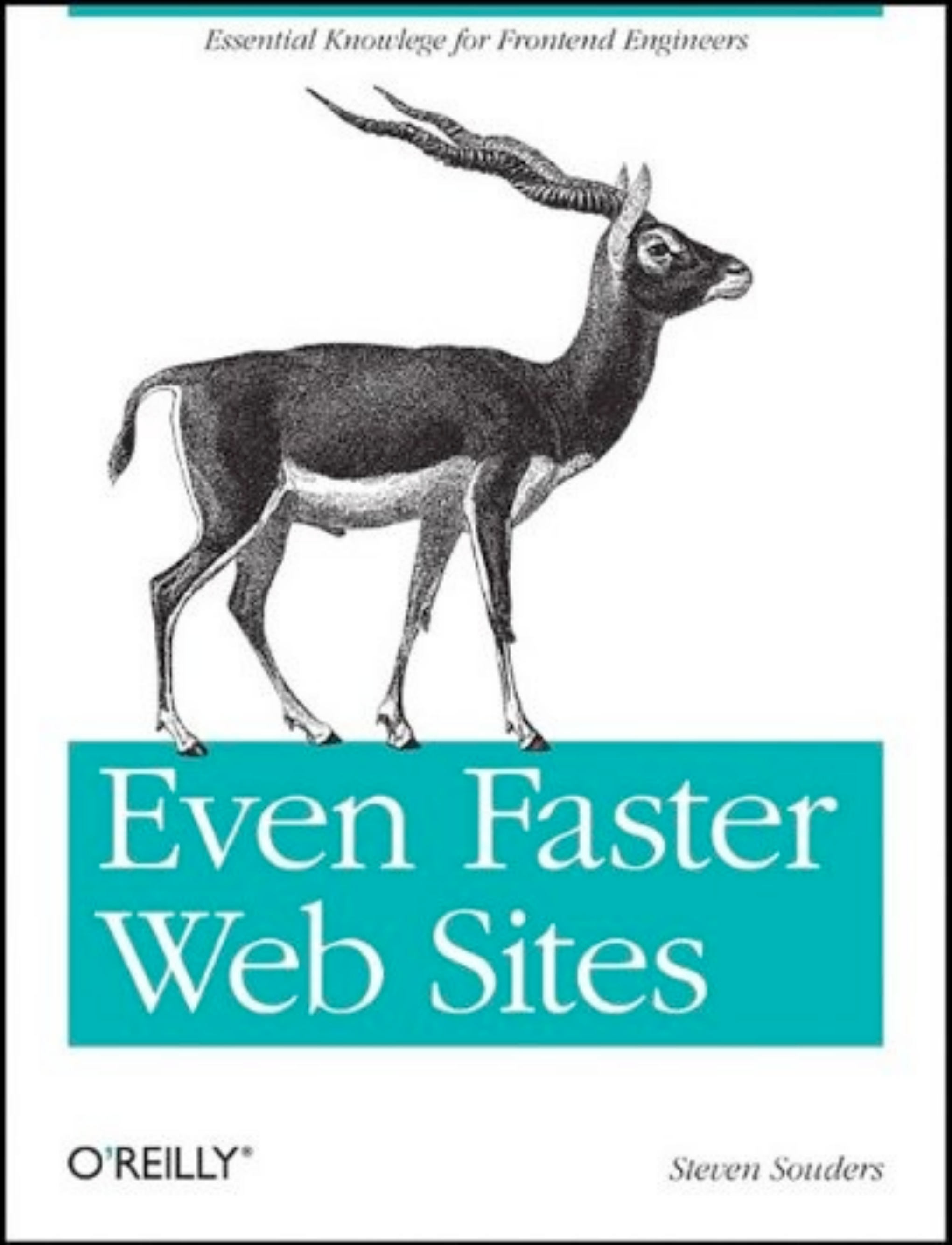
14 Steps to Faster-Loading Web Sites

High Performance Web Sites

Essential Knowledge for Front-End Engineers

O'REILLY®

Steve Souders
Foreword by Nate Koechley



Essential Knowlege for Frontend Engineers

Even Faster Web Sites

O'REILLY®

Steven Souders

Use external programs

```ruby
def thumbnail(temp, target)
    system("/usr/local/bin/convert #{escape(temp)} -resize
48x48! #{escape(target}")
end
```

Write inline C/C++ code

- RubyInline: Write foreign code within ruby code
  http://rubyinline.rubyforge.org/RubyInline/
- Rice: Ruby Interface for C++ Extensions
  http://rice.rubyforge.org/
- Ruby-FFI: a ruby extension for programmatically loading dynamic libraries
  http://github.com/ffi/ffi

Reference

- Advanced Rails Chap.6 Performance (O'Reilly)
- Rails Rescue Handbook
- Writing Efficient Ruby Code (Addison-Wesley)
- Ruby on Rails Code Review (Peepcode)
- Rails 2 Chap. 13 Security and Performance Enhancements (friendsof)
- Deploying Rails Application Chap.9 Performance (Pragmatic)
- http://guides.rubyonrails.org/caching_with_rails.html
- http://guides.rubyonrails.org/performance_testing.html
- http://railslab.newrelic.com/scaling-rails
- http://antoniocangiano.com/2007/02/10/top-10-ruby-on-rails-performance-tips/
- http://www.engineyard.com/blog/2009/thats-not-a-memory-leak-its-bloat/
- http://jstorimer.com/ruby/2009/12/13/essential-rails-plugins-for-your-inner-dba.html
- http://asciicasts.com/episodes/161-three-profiling-tools
- http://robots.thoughtbot.com/post/163627511/a-grand-piano-for-your-violin

So, how to scale?

- Rails performance (AR...etc how to ruby code)
- Web performance (yslow related)
- Asynchrony Processing (Message queue)
- HTTP Reverse Proxy Caching
- Partition Component using SOA
- Distributed Filesystem/Database

TODO (maybe next time)

- NoSQL: Key-value data store
- Front-end web performance
- HTTP reverse proxy caching

The End
感謝聆聽