

Philadelphia Emerging Technology

First to welcome you.
Claudia gave amazing talk.
Online advertising, fib
numbers



Aaron Patterson

@tenderlove

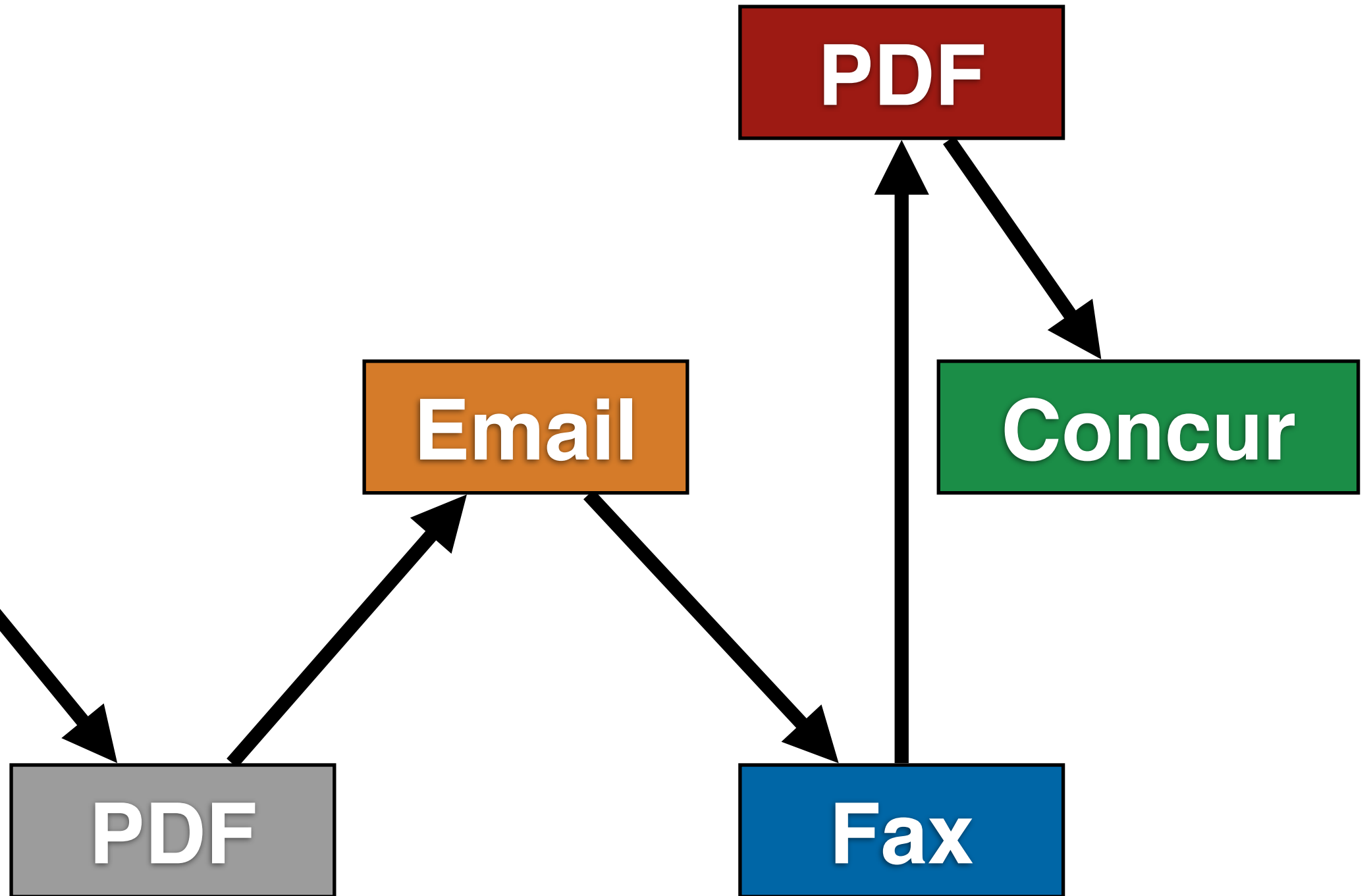
Rails Core Team

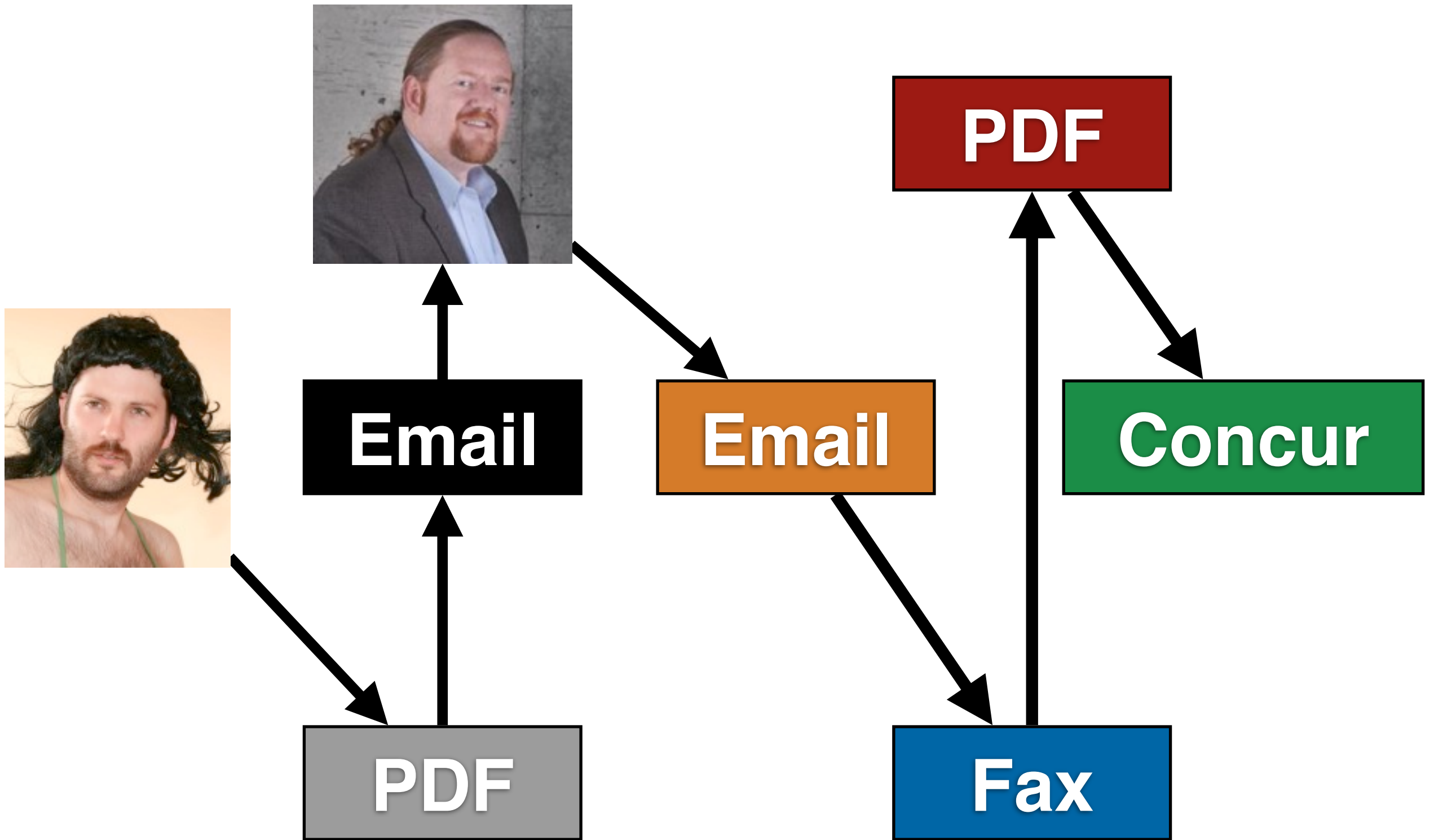
Ruby Core Team

enterprise gem

<https://github.com/tenderlove/enterprise>

Expense Reports





When I previewed the fax in the expense reporting system, there were tons of these at the bottom.



Please con

nting this email



Please con

nting this email



Please con

nting this email



Please con

nting this email



Please con

nting this email



Please con

nting this email



Please con

nting this email



Please con

nting this email

We actually hired
Rube Goldberg to
design the system
back in the day.

Goldberg

Expense

Reporting System

It's kind of genius because there was a legacy system, the fax machine, they needed to support.

People want Email, so we give them the email interface, but it actually just adapts to the existing code

Interfaces & Adapters

Imagine being in the meeting where they decided to do this.

Title:

Exploring the
internals of
Active Record

Title:

Also new things
in **Rails 4**

Title:

And some pictures of
my cat, but I'll front
load those.

Order of Operations

- ♥ Cats
- ♥ My side gig
- ♥ Rails Four Stuff
- ♥ Active Record Internals



こんにちは、
わたしが社長です。

I think he only
speaks Japanese.











I love my cat!

Ryan hates hearing
about my cat all the
time.

“If your wondering why I contacted
you instead of Aaron it's because
I don't want to hear about his cat.
I'm interested in mastering

Since last
year.

I got married!



Adequate

Everything. Adequately.

Talk about some
of our services.

“You get what we
think you paid for”

Disrupt Space

Disrupt the Space of Space Disruption



To understand SEO, we must understand the search engine.

SEO Optimization

It says Search Engine Optimization Optimization, but that is what we do.



First search engines
were invented in
1547 and were
steam powered.

First Search Engine

Invented in holland and came to the united states along with Lief Erikson (who later went on to invent cell phones).

In the US they were improved to be gasoline powered with pistons. You can tell this bit of history from the names.



Leif Erikson

(Later invented Cell Phones)



Google

The number of pistons
their search engine
has.

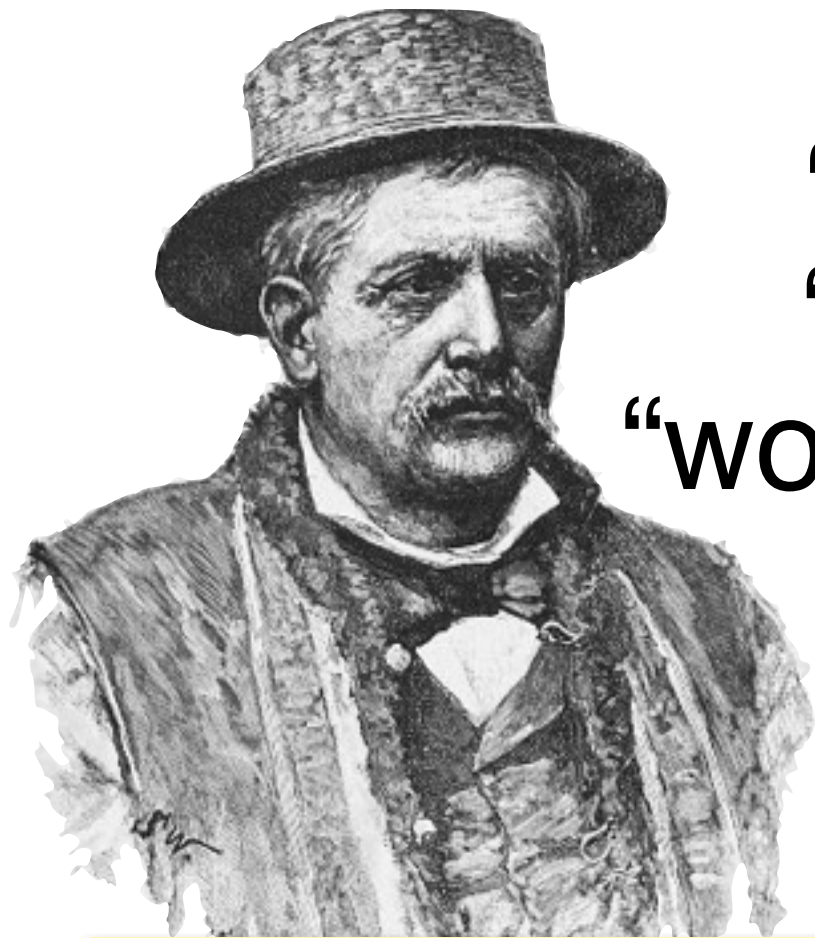
The sound the exhaust
system for the engine
makes.

Yahoo!

Bing



The sound of a
broken engine.



“faster!”
“better!”
“work damnit!”

By shouting, we can actually improve the performance of the search engine. Back when shouting at machines actually did something.





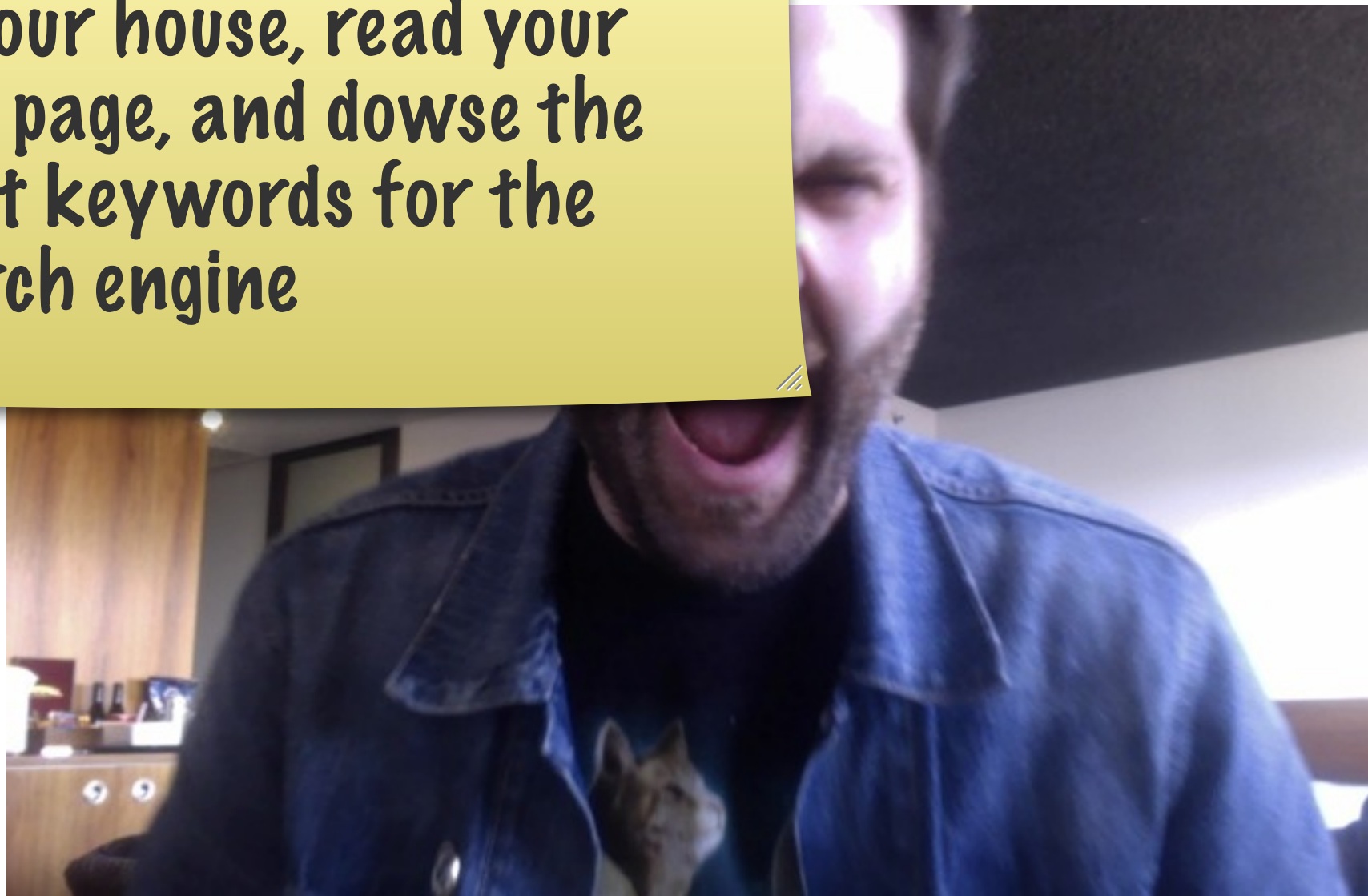
“buy!”
“cheap!”
“pills!”

**Alexander
Searchkeyword**

This lead to search engine dowsers. They would use dowsing rods to find the right keywords from dictionaries and shout them at the search engine.



For a small fee, I will come to your house, read your web page, and dowse the right keywords for the search engine



Doing Client Work.

recruiterspam.com

Rails **FOUROROR**

Release Date:
September 2012

Rails.queue

Using the Queue

```
job = MyJob.new  
Rails.queue << job
```

Consumer

```
Thread.new {  
  while job = Rails.queue.pop  
    job.run  
  end  
}
```

Problems.

“What about exceptions?”

“It’s queue specific.”

“What about serialization?”

“It’s queue specific.”

“What about job construction?”

“It must be marshallable.”

Job Construction

```
user = User.find(1)
job = Job.new(user_id = user.id)
Rails.logger.debug {<div data-bbox="225 400 775 714" data-label="Text">

NOPE


```

Job Construction

```
user = User.find 1  
job   = Job.new user.id  
Rails.queue << job
```

Job Definition

```
class Job
  def initialize(user_id)
    @user_id = user_id
  end

  def run
    user = User.find @user_id
    # ....
  end
end
```

Too Many
Open Questions.

Rails.queue

Job Definition

```
class RailsJob
  def initialize(*args)
    @args = args.map { |arg|
      ActiveRecord::Base
    }
  end
end
```

```
class Job < RailsJob
  def run
    do_stuff user
  end
end
```

This got pulled out of master months ago when we thought there was going to be a release. It seems like we could solve most of the problems by writing a Rails specific superclass.

minitest/spec

```
describe "whatever" do
  setup do
    # ...
  end
end
```

```
it "does some stuff" do
  1.must_equal 1
end
```

```
describe "some other stuff" do
  it "does some other stuff" do
    'foo'.must_match /foo/
  end
end
end
```

This is what a minitest/spec looks like. It looks very similar to RSpec, but it isn't.

Rails Test

Here is a Rails test with some of the DSL features that Rails adds on.

```
class SomeTest < ActiveSupport::TestCase
  setup { # ... }

  test "some thing" do
    # ...
  end
end
```

MiniTest

If we compare this to a minitest/spec test, it looks very similar.

```
class SomeTest < MiniTest::Spec
  setup { # ... }

  it "some thing" do
    # ...
  end
end
```

Refac

We can make the appropriate change to minitest/spec and now it looks exactly the same.

```
class SomeTest
  class << spec
    alias :test :spec
  end
end
```

```
  setup { # ... }
```

```
  test "some thing" do
    # ...
  end
```

```
end
```

AS::TestCase

```
class ActiveSupport::TestCase
  class << self
    alias :test :it
  end
end
```

```
class SomeTest < ActiveSupport::TestCase
  setup { # ... }

  test "some thing" do
    # ...
  end
end
```

The cool thing is that it's
100% backwards
compatible. Works
exactly the same as
minitest/unit.

mt/spec superclass

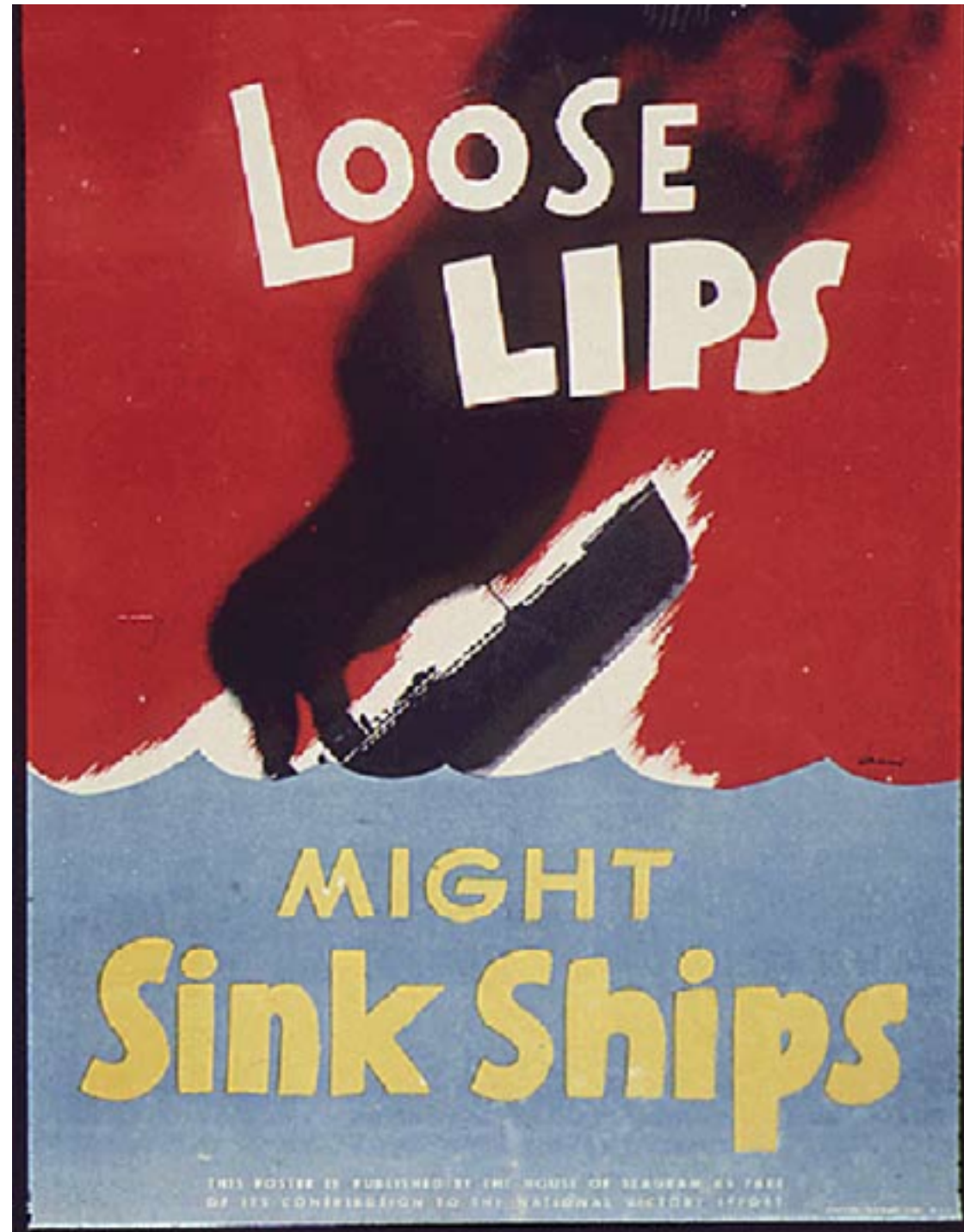
```
> MiniTest::Spec.ancestors  
=> [MiniTest::Spec,  
MiniTest::Unit::TestCase, ... ]
```


Free goodies!

```
describe "whatever" do
  it "does some stuff" do
    1.must_equal 1
  end
end
```

```
  describe "some other stuff" do
    it "does some other stuff" do
      'foo'.must_match /foo/
    end
  end
end
end
```

minitest/spec



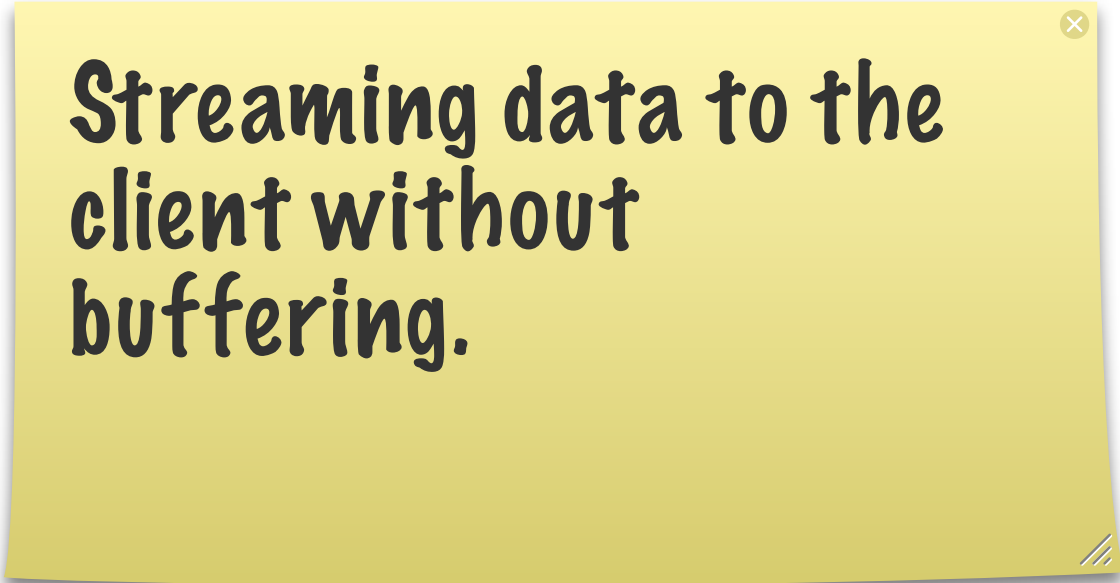
Someone Talked!



“I like the aesthetic”

TurboLinks

I kid, I kid!



**Streaming data to the
client without
buffering.**

Streaming

ActionController::Live

Example

```
class BrowserController < ApplicationController
  include ActionController::Live

  def index
    100.times do
      response.stream.write "hello!\n"
    end
    response.stream.close
  end
end
```



Mix in



Stream



[aaron@higgins lolwut (master)]\$

I

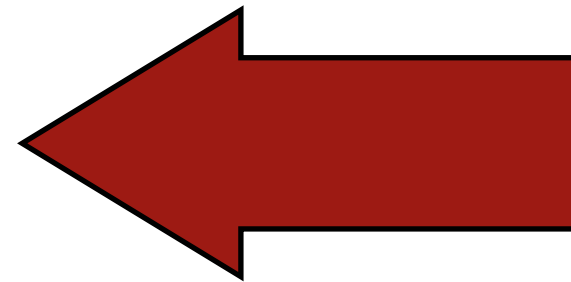
Applications

- ♥ Streaming ERB
- ♥ Infinite Stream APIs
- ♥ Server Sent Events

SSE R

SSE responses are infinite streams, but browsers will fire a javascript function every time an event is received.

```
HTTP/1.1 200 OK
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Type: text/event-stream
Transfer-Encoding: chunked
```



```
event: ping
data: {"ping":"2012-10-06T21:44:41-07:00"}
```

```
event: reload
data: {"changed":["/Users/aaron/git/lolwut/app/views/users/"]}
```

Listing users

Name

[New User](#)

```
<h1>Listing users</h1>
```

```
<table>  
<thead>
```

lolwut — bash — 94x24

bash

bash

```
[aaron@higgins lolwut (master)]$
```

[confirm: 'Are yo

FS Events

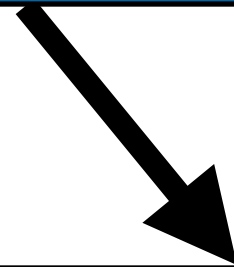
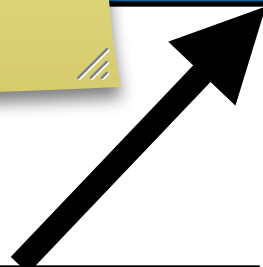
When the FS changes,
it tells the webserver,
webserver tells the
browser

Puma

All in the same
process.

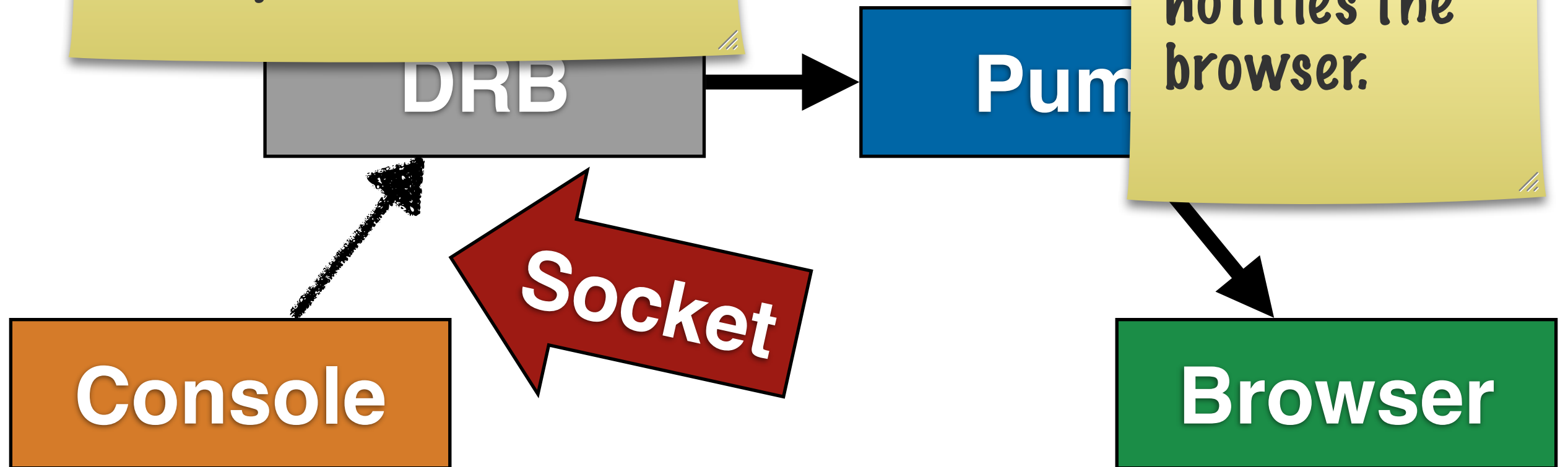
FS-Events

Browser



DRB Events

When a model changes, a message is sent via DRB running in the Puma process.



The Arrows

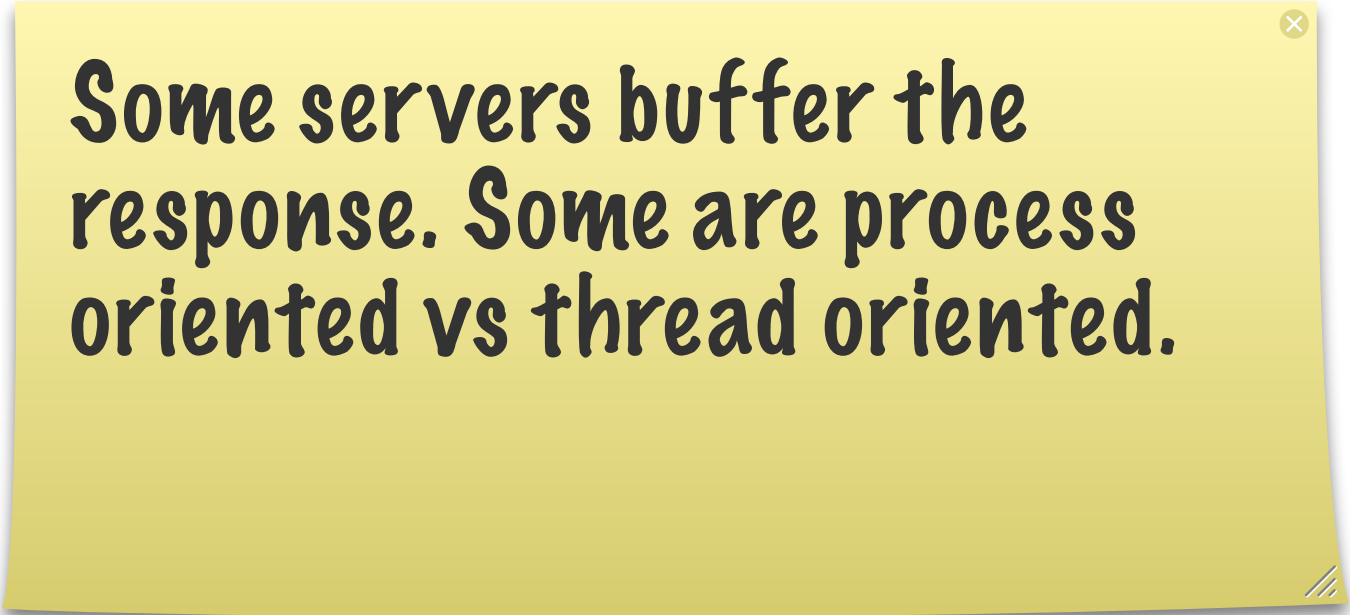
- ♥ DRB server
- ♥ SSE implementation
- ♥ JS Components

We can use SSEs for streaming or polling because you can provide a connection timeout.

Streaming / Polling

(with timeout)

Pitfalls



Some servers buffer the response. Some are process oriented vs thread oriented.

Webserver

If your stream is infinite, it's going to keep that socket open for an infinite amount of time. You need to design for this. But this is why we're putting effort in to a thread safe Rails.

Long Responses

There is only one reliable way to tell if the client disconnected, and that is to send data and have it fail.

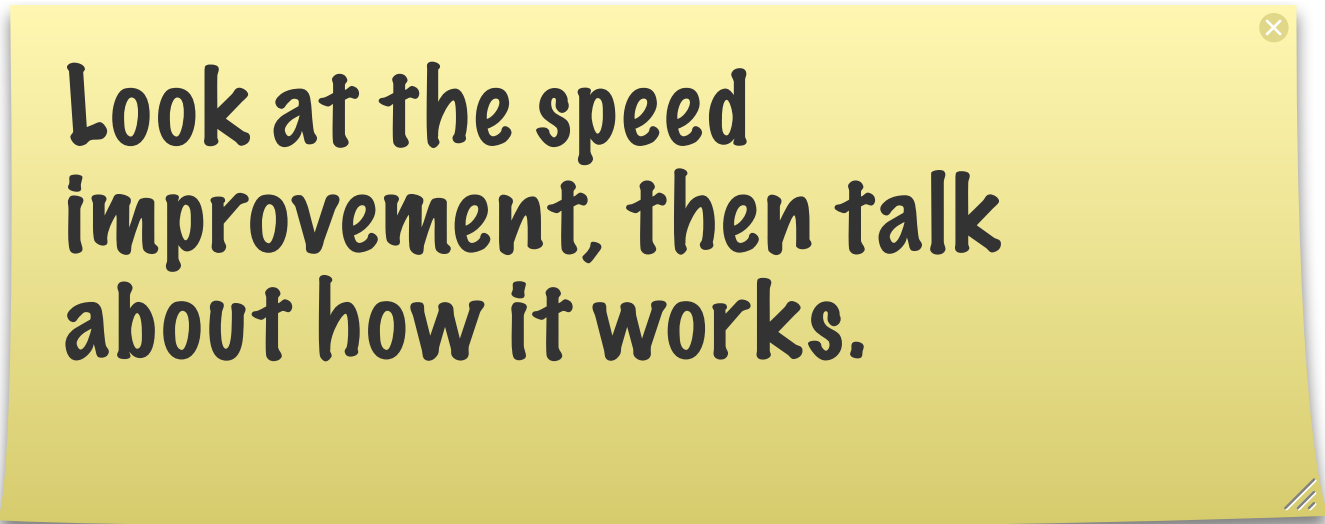
Client Disconnect

Send a ping

```
def index
  Thread.new(response.stream) do |stream|
    loop do
      begin
        stream.write ping_packet
      rescue IOError
      end
    end
  end
end

begin
  response.stream.write generate_sse
rescue IOError
end
end
```

Use Polling



Look at the speed
improvement, then talk
about how it works.

Faster tests*

*they're not actually faster (well, they might be)

3.2.x

```
$ time rake test
```

```
[snip]
```

```
real 0m4.756s
```

```
user 0m4.147s
```

```
sys 0m0.582s
```

4.0.0.beta

```
$ time rake test
```

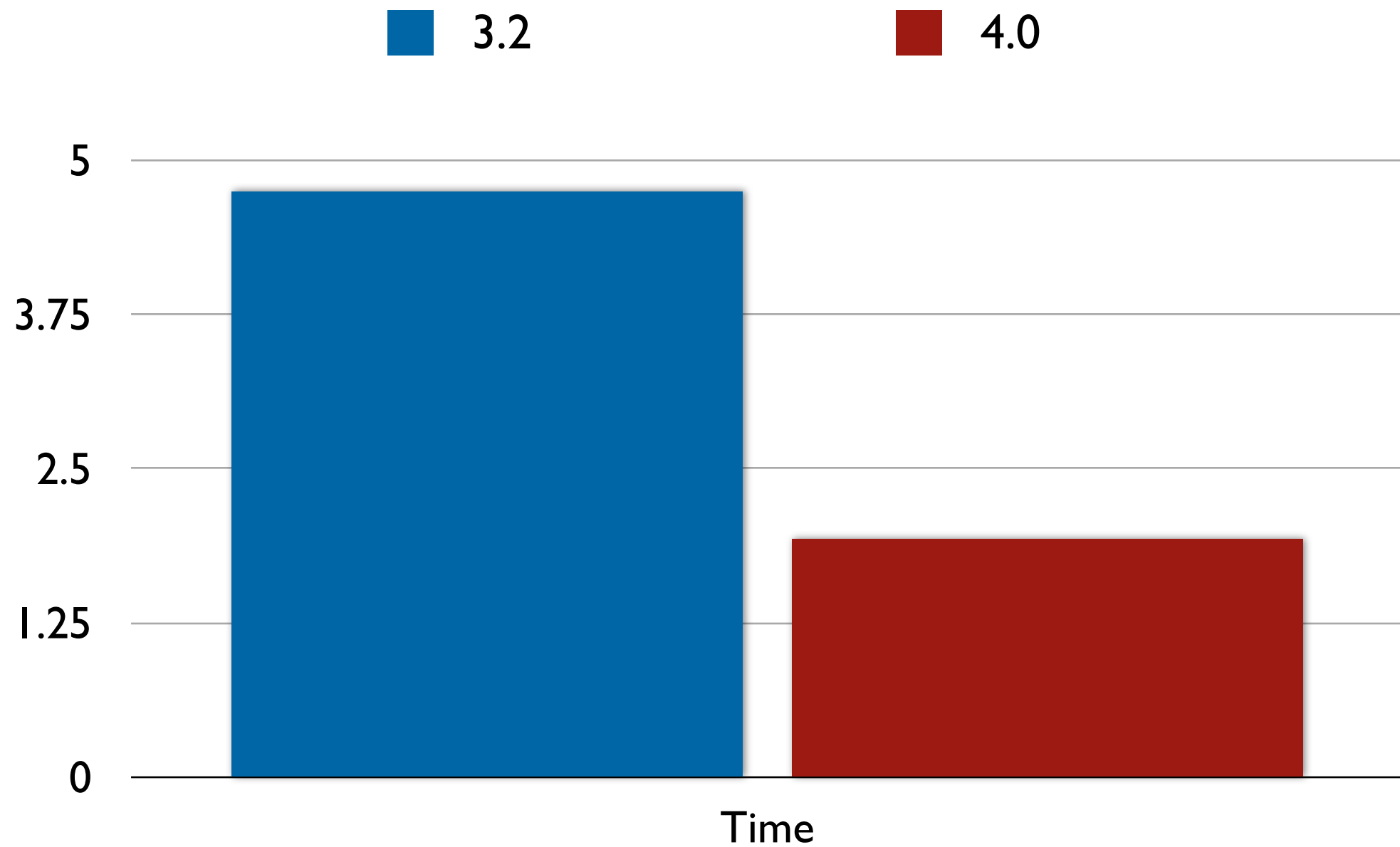
```
[snip]
```

```
real 0m1.934s
```

```
user 0m1.701s
```

```
sys 0m0.224s
```


Speed



Where is the
bottleneck?

Time Breakdown

Rails 3.2 (1 test)

```
$ time ruby -I lib:test test/functional/  
line_items_controller_test.rb
```

```
real 0m1.733s  
user 0m1.518s  
sys  0m0.203s
```

Rails 4.0 (1 test)

```
$ time ruby -I lib:test test/controllers/  
line_items_controller_test.rb
```

```
real 0m1.753s  
user 0m1.535s  
sys  0m0.208s
```

Environment

Require the environment, and do nothing. Loads the application.

```
$ time ruby -Ilib:test:. -rconfig/  
environment -e ' '
```

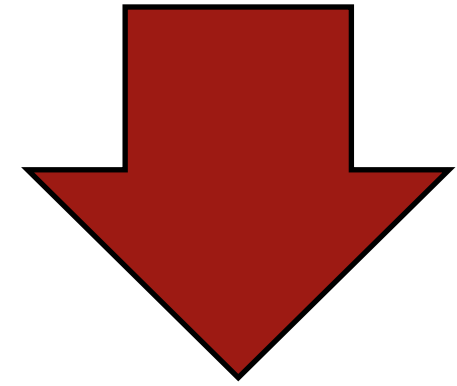
```
real 0m1.442s  
user 0m1.255s  
sys 0m0.179s
```

How do the test
tasks work?

Sample Test Task

```
# Rakefile
Rake::TestTask.new do |t|
  t.libs << "test"
  t.verbose = true
  t.warning = true
end
```

Test Run



```
/Users/aaron/.rbenv/versions/2.1.0-dev/bin/ruby -w  
-I"lib:test"  
-I"/Users/aaron/.rbenv/versions/2.1.0-dev/lib/ruby/  
gems/2.1.0/gems/rake-10.0.4/lib" \  
"/Users/aaron/.rbenv/versions/2.1.0-dev/lib/ruby/  
gems/2.1.0/gems/rake-10.0.4/lib/rake/  
rake_test_loader.rb" \  
"test/test*.rb"
```

How does this relate
to Rails?

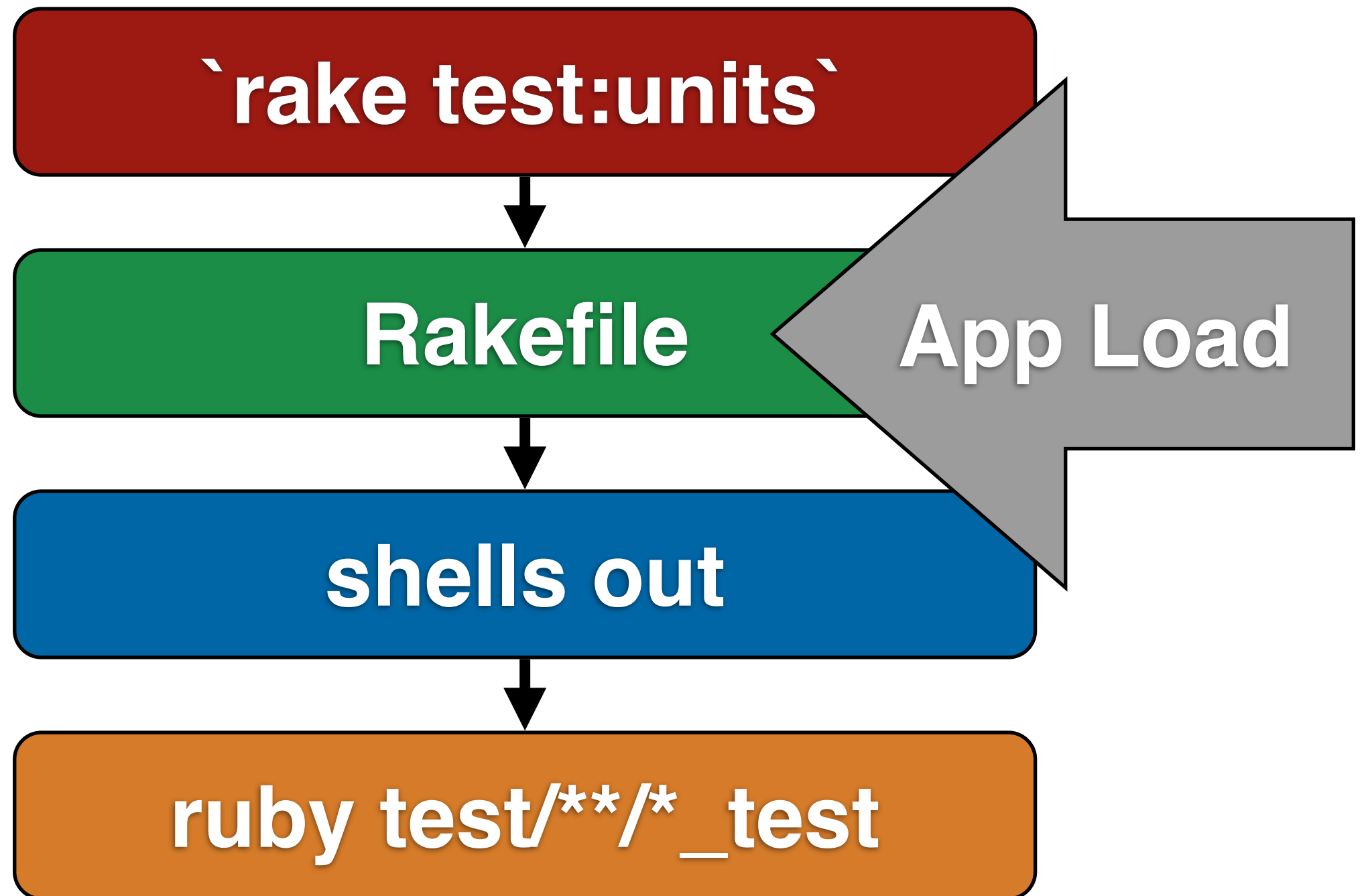
`rake test`
runs Ruby twice.

Rake + Rails

You app can custom Rake tasks

lib/tasks/**

In order to find
those, we need to
load your
application.



`rake test:units` =
2 * app load time

$$\text{'rake test'} = 4 * \text{app load time}$$

Multiple Loads

Just Require

```
task :test do
  Dir['test/**/*_test'].each do |file|
    require file
  end
end
```

Challenges

Switching Environments

Your app is a
singleton

What env is `rake`?

If it looks like you're running a test task, then change the env to the test env.

Changing
before app load

Migrations

```
$ rake test
```

```
You have 1 pending migrations:
```

```
  20130401175825 CreateUsers
```

```
Run `rake db:migrate` to update your  
database then try again.
```


loading schema.rb

Real Solution: Remove Singleton

Active Record

Internals

Connection Pooling

Thread Safety

SQL Construction

Statement Caching

Connection Pooling

Configuration

development:

adapter: sqlite

database: db.sqlite3

pool: 5

timeout: 500



Pool Size Limit

Pool limit \approx

Server threads

AR::Base.connection

Do Stuff

Send Response

Check in connection

Check-in

Middleware class,
operates on the
connection associated
with the current thread.

`ActiveRecord::ConnectionManagement`

Threads > Pool Size
is Just Fine™

Manual Checkout

```
# Checkout
```

```
conn = ActiveRecord::Base.  
  connection_handler.  
  checkout
```

```
# Checkin
```

```
ActiveRecord::Base.  
  connection_handler.checkin(conn)
```

Reaping

How long has it been checked out? How often should we check?

production:

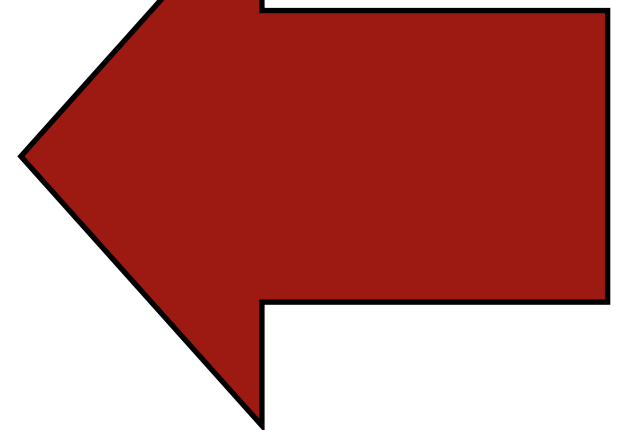
adapter: sqlite3

database: db/development.sqlite3

pool: 5

If a thread dies, what happens to the connection?

on_timeout: 5
frequency: 10



Thread Safety



Short rant about
thread safety.

“Is XXX Library
Thread Safe?”

Everything is Thread Safe,
if you know the rules.

DB Connections
ARE NOT
Thread Safe

No locks around
socket operations

```
conn = ActiveRecord::Base.connection
```

```
t1 = Thread.new do  
  loop do  
    conn.execute "SOME SQL"  
  end  
end  
}
```

```
t2 = Thread.new do  
  loop do  
    conn.execute "OTHER SQL"  
  end  
end
```

AR Objects
ARE NOT
Thread Safe

No locks around
read / write ops

RACE CONDITION

```
person = Person.new
```

```
t1 = Thread.new {
```

```
  100.times {
```

```
    person.friends
```

```
  }
```

```
end
```

```
}
```

```
t2 = Thread.new {
```

```
  100.times {
```

```
    person.friends = Person.new
```

```
  }
```

JRuby, probably
get an exception,
MRI, it may do the
right thing.

Underlying DS is a
hash with no locks

How do we
parallelize?

para-ella-ella-ella-eh-eh-eh-ize

Split Work by Type


```
work_queue = Queue.new  
write_queue = Queue.new
```

```
Thread.new {  
  loop do; work_queue << find_work; end  
}
```

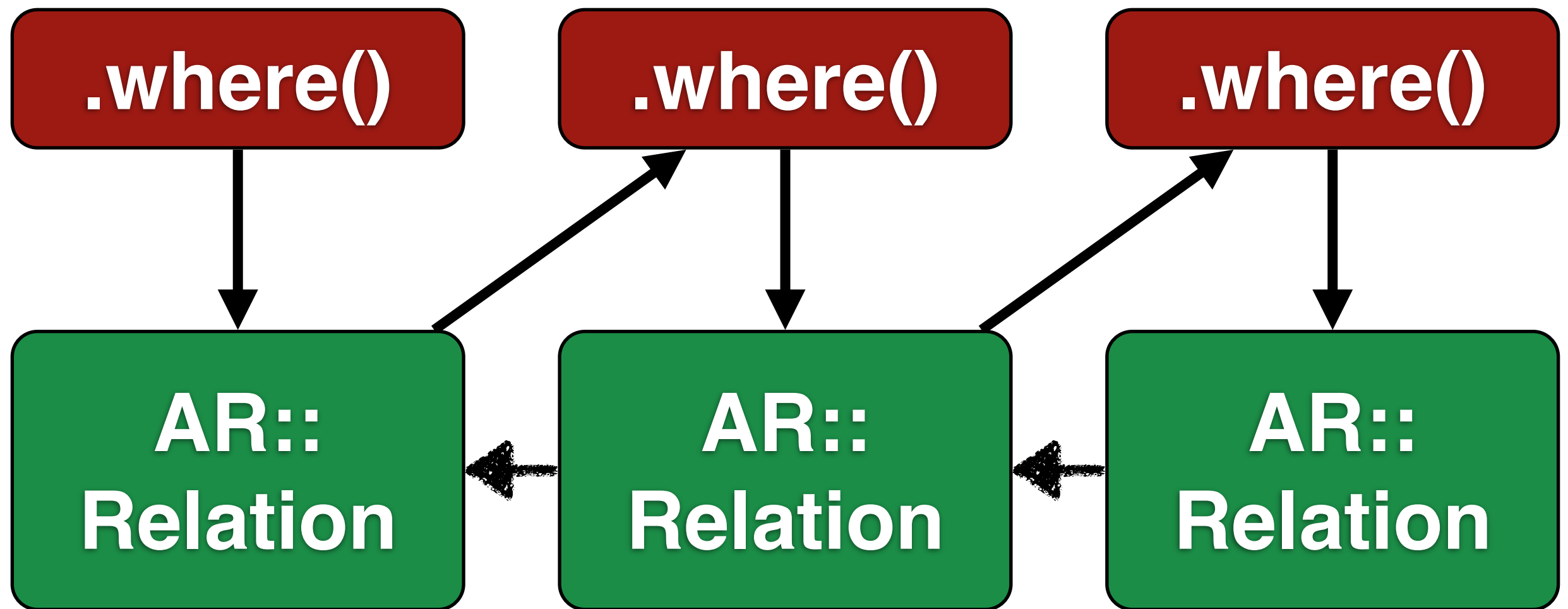
```
Thread.new {  
  while job = work_queue.pop  
    # process job  
    write_queue << result  
  end  
}
```

```
Thread.new {  
  while record = write_queue.pop; record.save; end  
}
```

SQL Construction

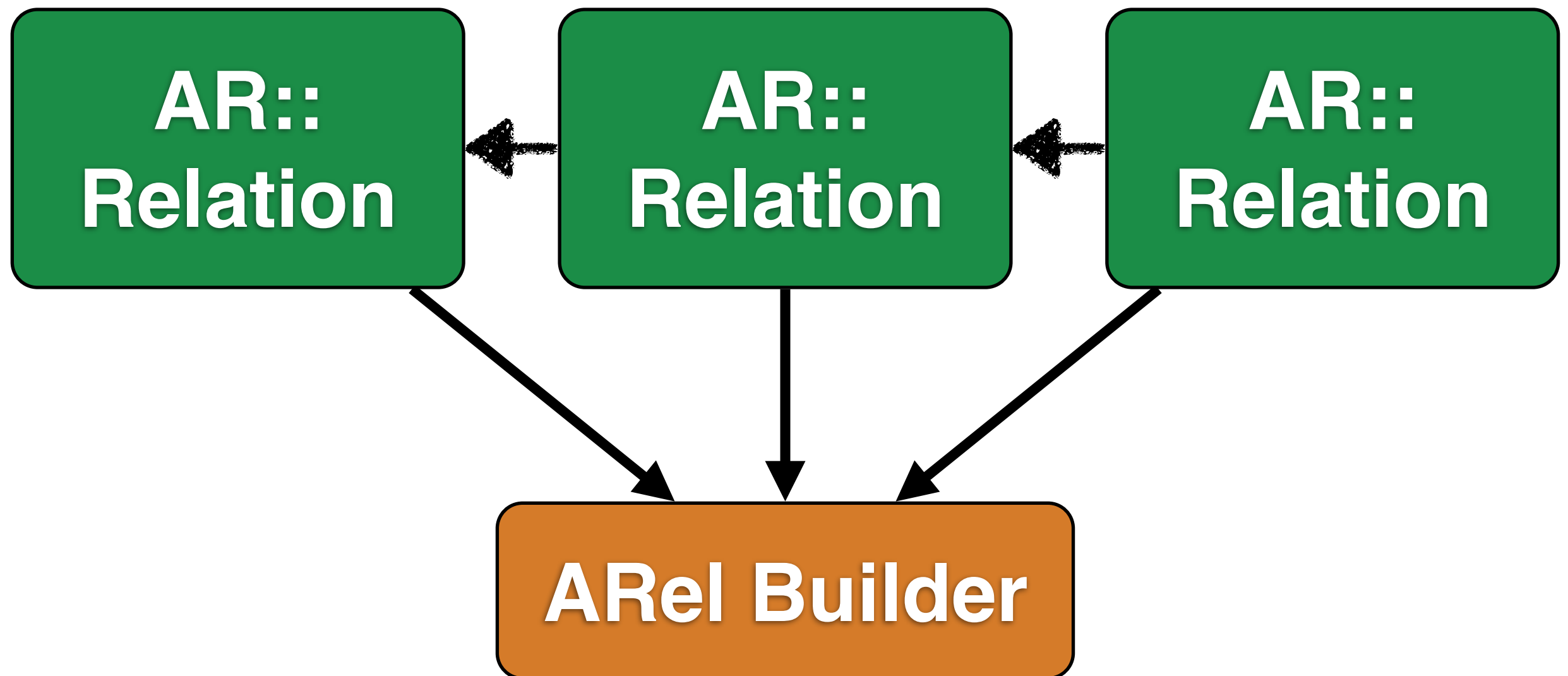
```
Post.where(...).where(...).where(...)
```

Post.where.where.where



AR::Relation holds
Query Information

`.first, .each, .to_a`



`AR::PredicateBuilder`

Conversion

Seems we're doing lots of work, but we have reduced it some.

ARel Builder

SQL

Database

to_a => execute

Statement Cache

Two Caches

♥ Query Cache

♥ Statement Cache

Query Cache:
Same Statement
Same Results

Only 1 Query

`Post.find(1)`

`Post.find(1)`

`Post.find(1)`

`Post.find(1)`

`Post.find(1)`

`Post.find(1)`

Statement Cache

Post.

where(..).

where(..).

find(x)

```
SELECT * FROM  
"posts" WHERE ...  
AND id = $1
```

Statement Cache

- ♥ Generate SQL
- ♥ Send SQL
- ♥ Get a token
- ♥ Always send the token

Saves Parse Time

Saves Planning Time

Saves Bandwidth

*Except on MySQL

But we're at an Enterprise Conference, right?

Memory Increase

Cache Size Limit

LRU

Post.

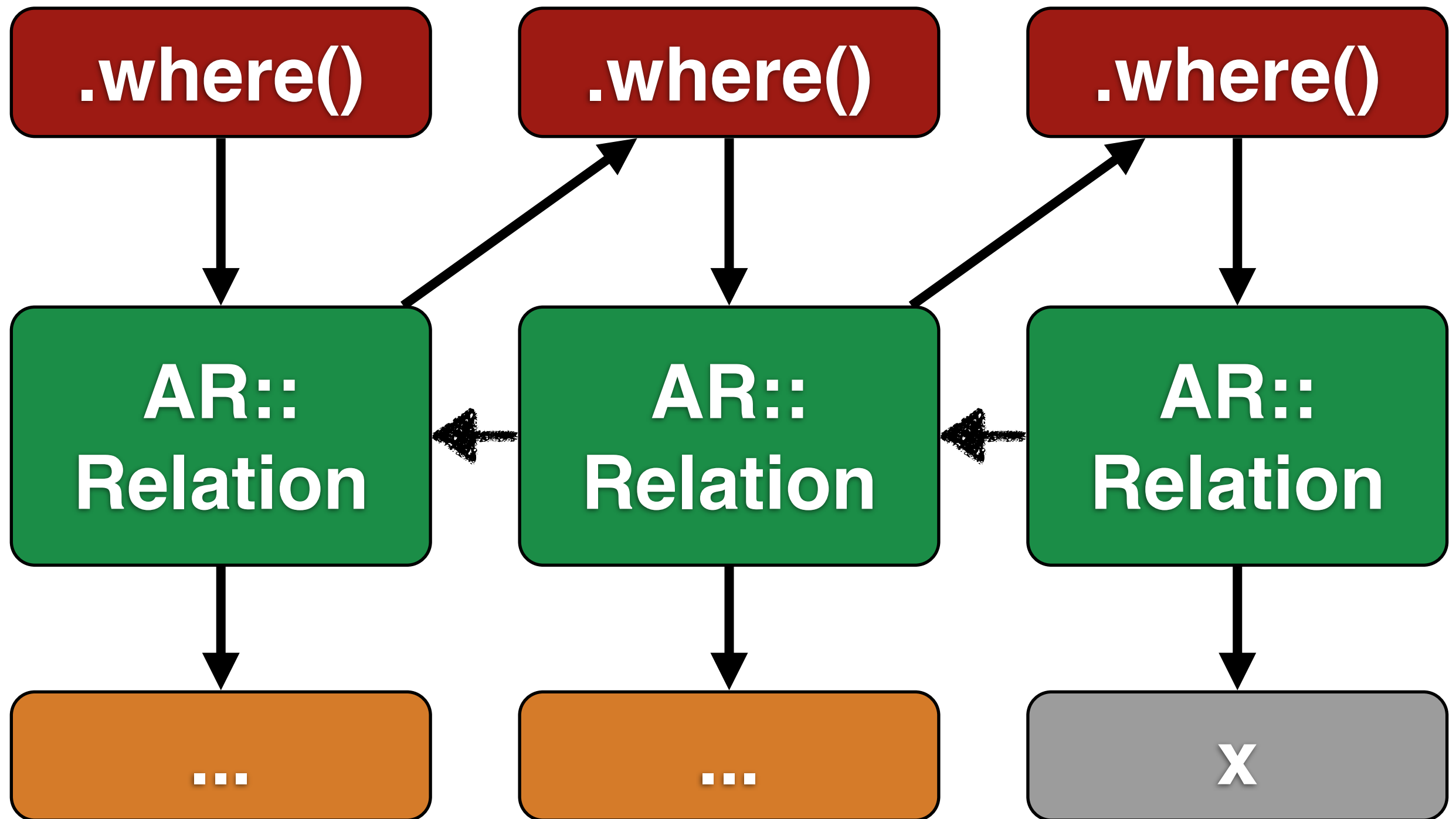
where(..).

where(..).

find(x)

Problem is that when we do this execution, we get all these objects created.

Post.where.where.where.find(x)

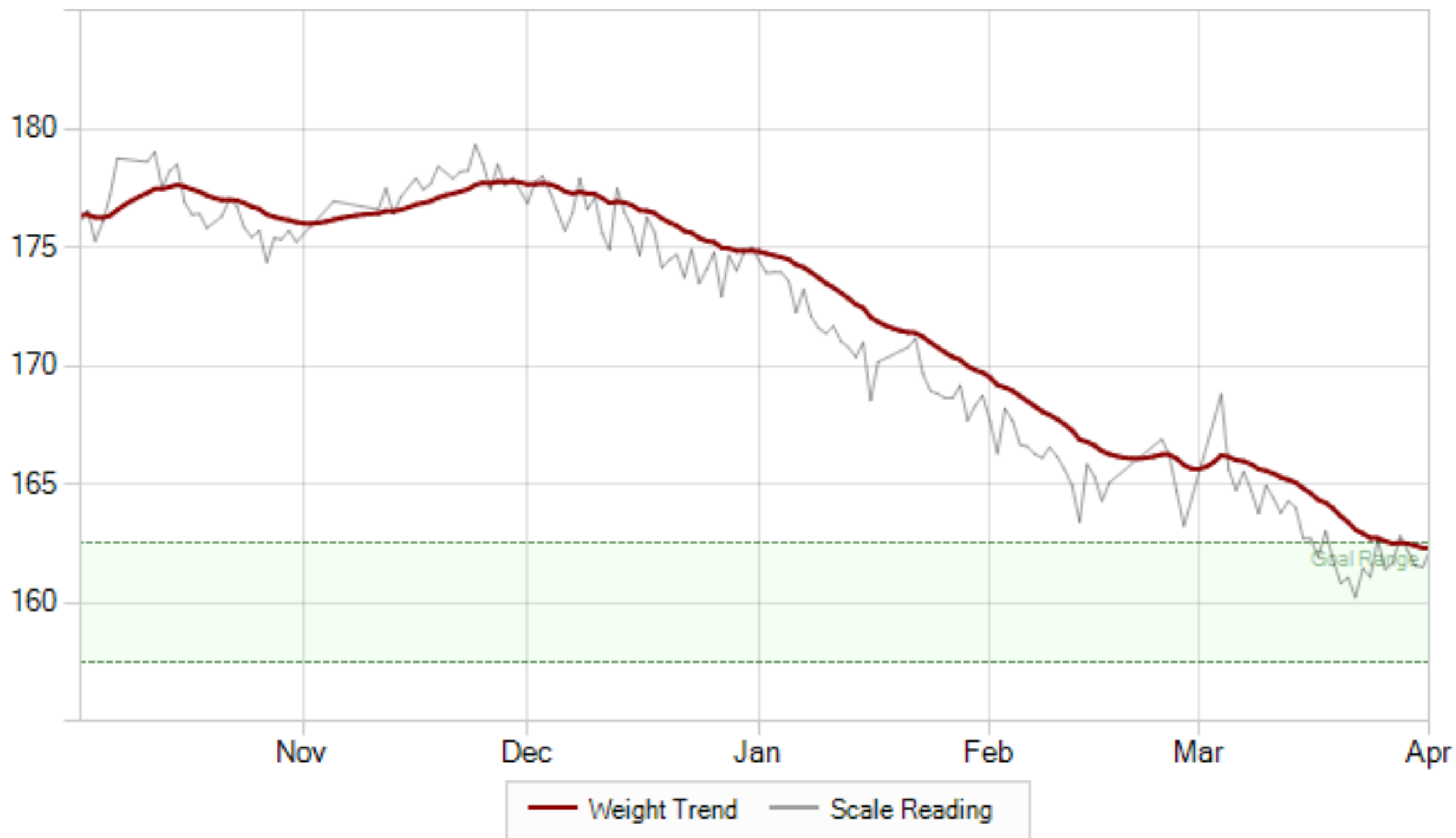


Cache all invariants

```
QUERY = CacheQuery do |variant|  
  Post.where.where.where.find(variant)  
end
```

```
QUERY.execute(params[:id])
```

Mamba Time



My Weight

Go to 7-11, find gummies, start buying them every week. One day the guy says "hey! Mamba Time!"

Health Walk

I am the guy that buys gummy bears. I will be the best damn gummy candy buyer ever.

Find **your**
Mamba Time

Be the Best
Mamba Timer

Thank You!

Questions?