An ok compromise:
Faster development by designing for the Rails Autoloader

Find a copy of these slides at `https://speakerdeck.com/bensheldon`

My goal

Give some meaning and actions to this extra-dense documentation to help you make and keep your Rails development environment wicked fast.

- Rails Guides: The Rails Initialization Process
- Rails Guides: Autoloading and Reloading Constants
- Rails Guides: Configuring Rails Applications: Initialization Events
- Rails API: `ActiveSupport::LazyLoadHooks`

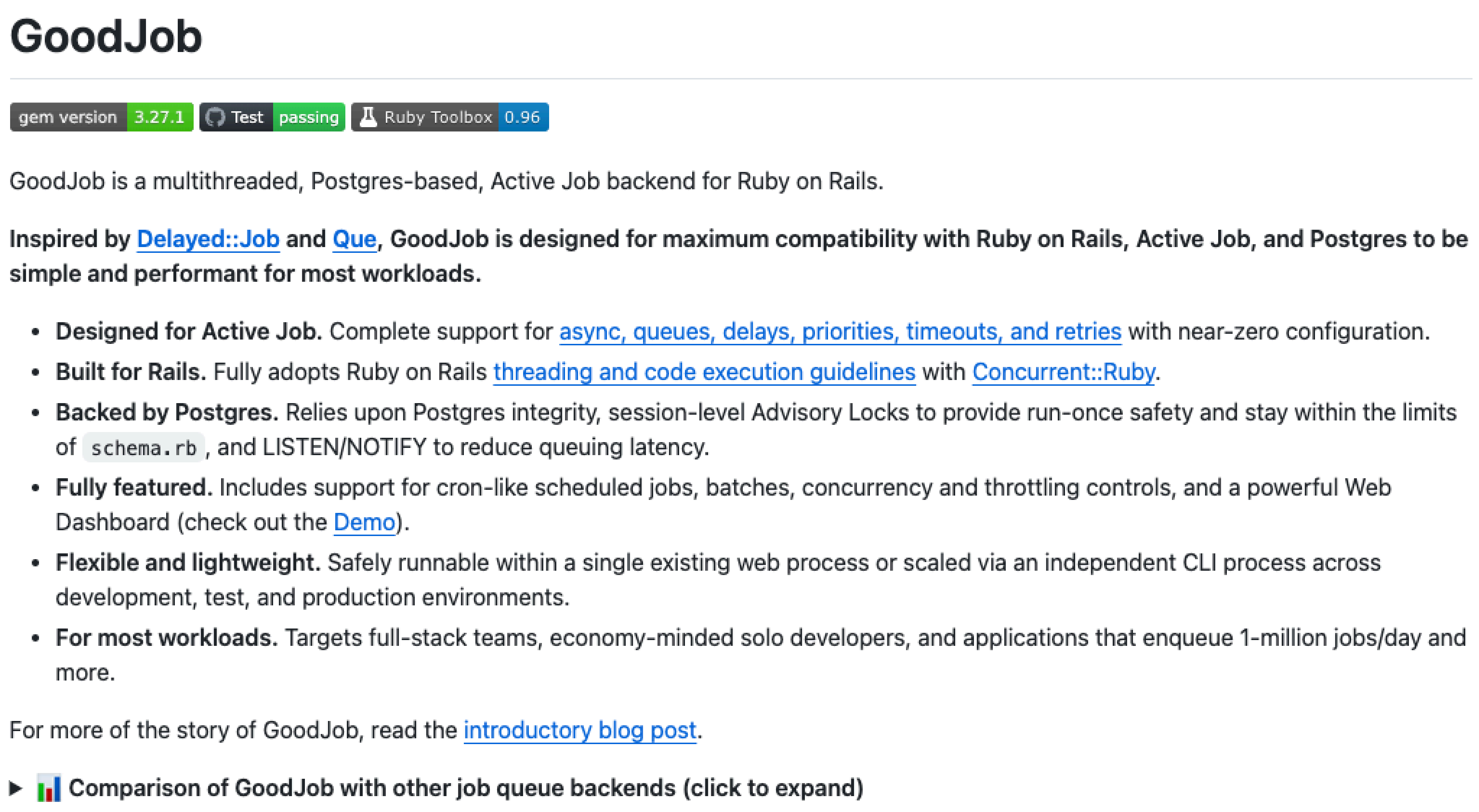
About me, Ben Sheldon

💻 Work at GitHub. Engineering Manager of the Ruby Architecture Team

👍 Author of GoodJob, multithreaded Postgres-based Active Job backend

🌆 Live in San Francisco

🐈‍⬛ 🐈‍⬛ Fostering 2 very good rescue cats that need a nice home



GoodJob

gem version 3.27.1 | Test passing | Ruby Toolbox 0.96

GoodJob is a multithreaded, Postgres-based, Active Job backend for Ruby on Rails.

Inspired by Delayed::Job and Que, GoodJob is designed for maximum compatibility with Ruby on Rails, Active Job, and Postgres to be simple and performant for most workloads.

- **Designed for Active Job.** Complete support for async, queues, delays, priorities, timeouts, and retries with near-zero configuration.
- **Built for Rails.** Fully adopts Ruby on Rails threading and code execution guidelines with Concurrent::Ruby.
- **Backed by Postgres.** Relies upon Postgres integrity, session-level Advisory Locks to provide run-once safety and stay within the limits of `schema.rb`, and LISTEN/NOTIFY to reduce queuing latency.
- **Fully featured.** Includes support for cron-like scheduled jobs, batches, concurrency and throttling controls, and a powerful Web Dashboard (check out the Demo).
- **Flexible and lightweight.** Safely runnable within a single existing web process or scaled via an independent CLI process across development, test, and production environments.
- **For most workloads.** Targets full-stack teams, economy-minded solo developers, and applications that enqueue 1-million jobs/day and more.

For more of the story of GoodJob, read the introductory blog post.

▶ 📊 **Comparison of GoodJob with other job queue backends (click to expand)**

A brief history of background jobs in Rails

- 2009 - **Delayed::Job**
- 2012 - **Sidekiq**
- 2013 - **Que**
- 2014 - **ActiveJob** in Rails v4.2
- 2020 - **GoodJob** ⬅️ That's my gem 🙌
- 2023 - **Solid Queue**

Feature Request: Cron-style repeating/recurring jobs

> As a developer, I want GoodJob to enqueue Active Job jobs on a recurring basis so it can be used as a replacement for cron. Example interface:

```
class MyRecurringJob < ApplicationJob
  repeat_every 1.hour
  # or
  with_cron "0 * * * *"

  def perform
  # ... ...
end
```

That sure would be a nice way to do it 💅

💔 But I built it in GoodJob this way, instead

```ruby
# config/application.rb or config/initializers/good_job.rb
config.good_job.cron = {
  recurring_job: {
    cron: "0 * * * *",
    class: "MyRecurringJob", # a String, not the constant 😰
  }
}
```

The configuration doesn't live with the job 😕

```ruby
# in the GoodJob gem...
ActiveSupport.after_initialize do
  config.good_job.cron.each do |_, config|
    when_scheduled(config[:cron]) do
      # only do this at the scheduled time
      config[:class].constantize.perform_later
    end
  end
end
```

😭 But why?

- We want our Rails application to boot really fast in Development.
- Loading files and constants, just to read configuration inside of them, is slow.
- Rails goes to a lot of trouble to defer loading files and constants, because that's fast.
- The mechanism is called **Autoloading**, which is built into Ruby (`autoload`) and via the Zeitwerk library (Rails uses both).
- This all largely happens behind the scenes, and we largely don't think about it.
- Let's not mess it up.

Development != Production

- I'm specifically talking about **Development** workflows, where code is **Autoloaded**
- In Production, Rails will (mostly) **Eagerly Autoload** these files/constants instead of *lazily* autoloading them.
  - `config.cache_classes = true`
  - `config.eager_load = true`
- Remember, we're talking here about **Development**, where we want fast feedback from code changes and not ⏳

Application Structure

	Configuration	Behavior

Application	`config/environment.rb`
	`config/application.rb`
	`config/initializers/*.rb`
	`config/routes.rb`
	
	any `require`’s in those files
		`app/controllers/*.rb`
		`app/helpers/*.rb`
		`app/jobs/*.rb`
		`app/models/*.rb`
		`app/services/*.rb`
		
		`app/lib or`
		`lib/` if `config.autoload_lib`

Framework	`Railties`
	`Engines`
	`ActiveRecord`
	`ActionController`
	`..etc`
		`ActiveRecord::Base`
		`ActionController::Base`
		`…etc`

Break it down

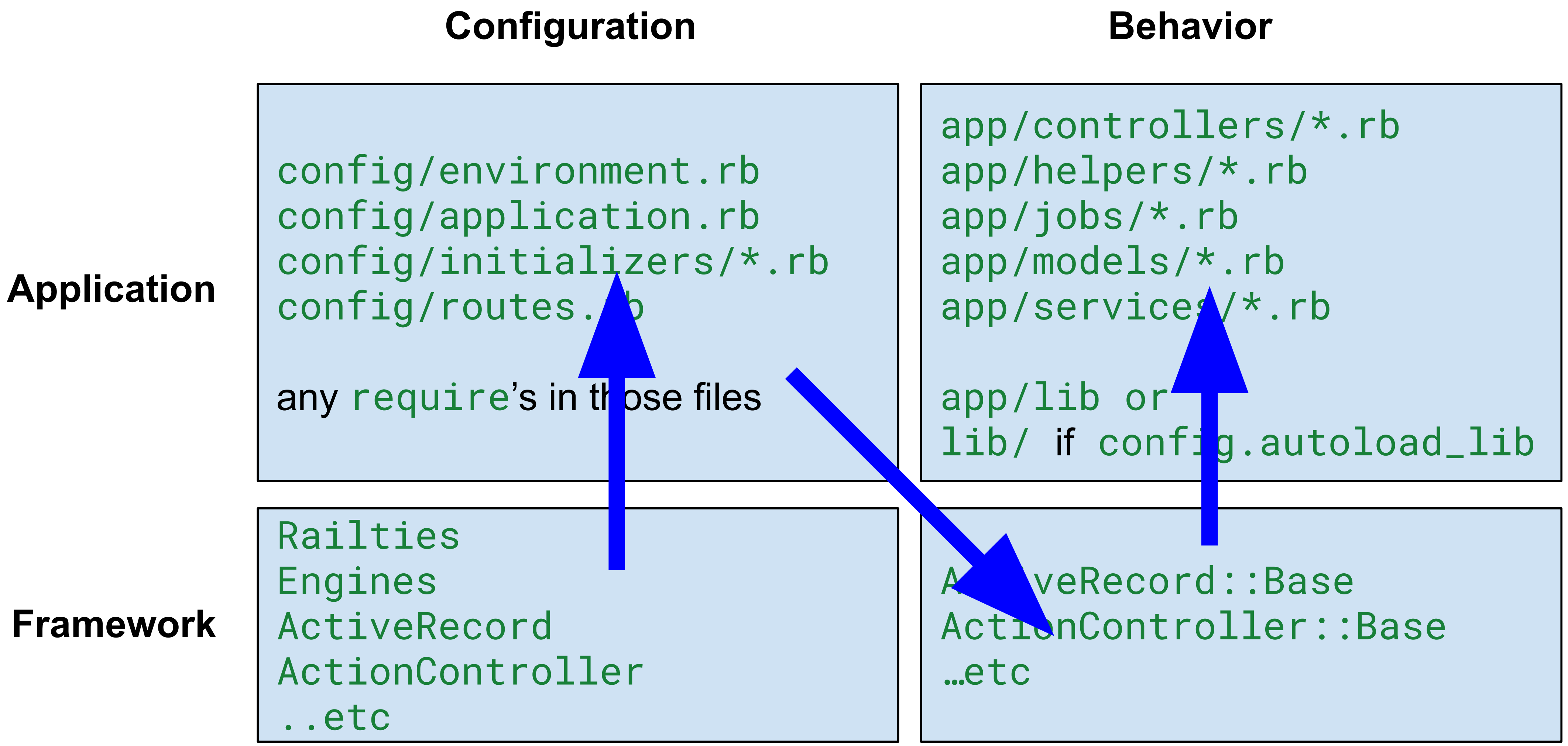
- **Configuration** is `require`'d during boot.
- **Behavior** is autoloaded
  - Only load what is needed when it's needed, *ideally* never.
  - To serve a single web request, to run that one test, to open the console, and code reload.

	Configuration	**Behavior**

Application	`config/environment.rb`
	`config/application.rb`
	`config/initializers/*.rb`
	`config/routes.rb`
	
	any `require`'s in those files
		`app/controllers/*.rb`
		`app/helpers/*.rb`
		`app/jobs/*.rb`
		`app/models/*.rb`
		`app/services/*.rb`
		
		`app/lib or`
		`lib/` if `config.autoload_lib`

Framework	`Railties`
	`Engines`
	`ActiveRecord`
	`ActionController`
	`..etc`
		`ActiveRecord::Base`
		`ActionController::Base`
		`…etc`

Application boot: ordered

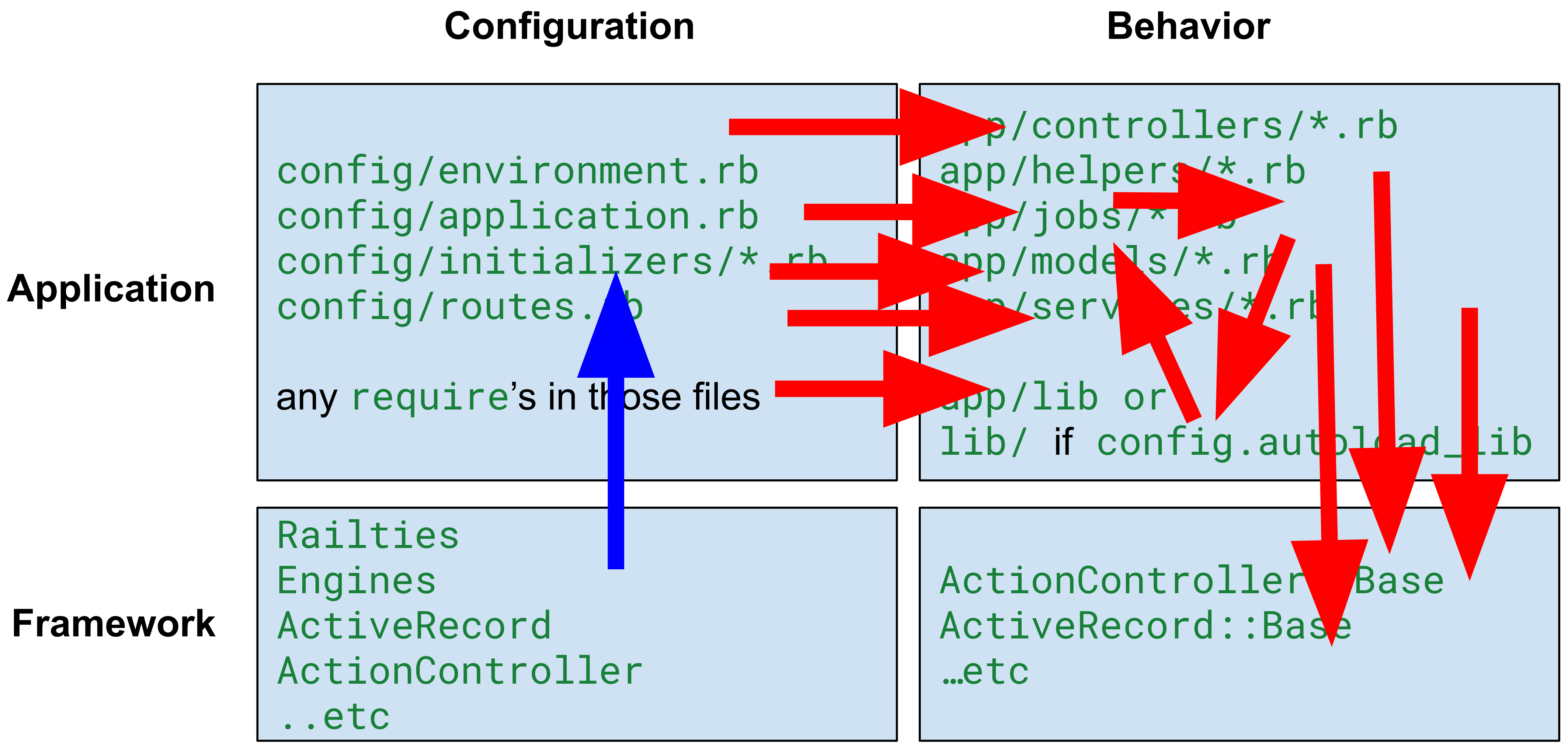


	Configuration	Behavior

Application	`config/environment.rb`
	`config/application.rb`
	`config/initializers/*.rb`
	`config/routes.rb`
	
	any `require`'s in those files
		`app/controllers/*.rb`
		`app/helpers/*.rb`
		`app/jobs/*.rb`
		`app/models/*.rb`
		`app/services/*.rb`
		
		`app/lib or`
		`lib/` if `config.autoload_lib`

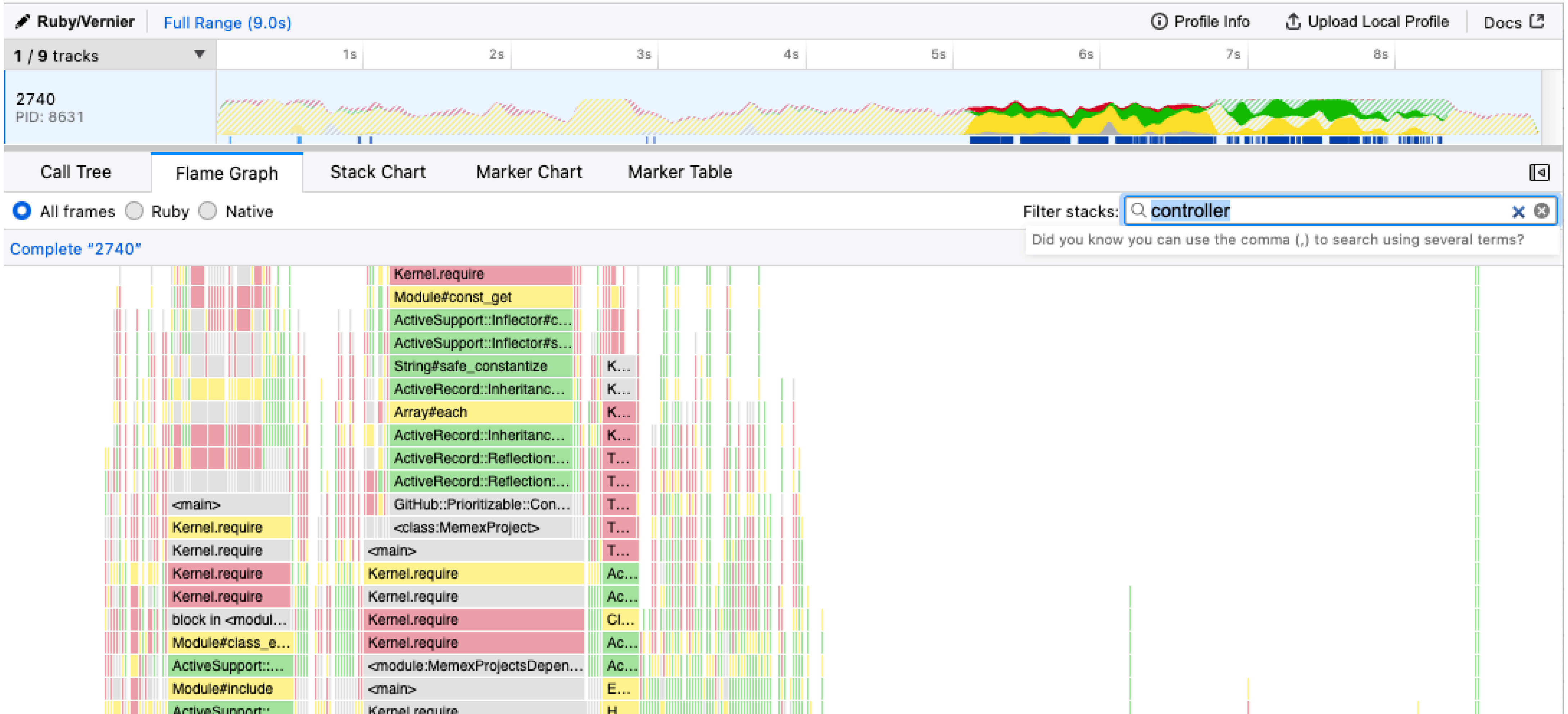
Framework	`Railties`
	`Engines`
	`ActiveRecord`
	`ActionController`
	`..etc`
		`ActiveRecord::Base`
		`ActionController::Base`
		`…etc`

Application boot: disordered



An example, from GitHub.com

`$ vernier run rails runner "puts true"` : *4 seconds spent loading controllers* 🙀



Filter stacks: controller

Did you know you can use the comma (,) to search using several terms?

Repair work

🔦 Finding the culprit (temporary debugging)

```
# config/application.rb
require "rails"
+ ActiveSupport.on_load(:action_controller_base) do
+   puts "action_controller_base loaded"
+   puts caller
+ end
```

🚧 Fixing it (and like 7 more like this)

```
# config/initializers/asset_path.rb
+ ActiveSupport.on_load(:action_controller_base) do
  ActionController::Base.asset_host = ...
+ end
```

`ActiveSupport::LazyLoadHooks`

```
# rails/actionpack/lib/actioncontroller/base.rb
# ...

  ActiveSupport.run_load_hooks(:action_controller_base, self)
end
```

Most Rails framework autoloaded behavioral files will offer LazyLoadHooks (`ActiveRecord::Base`, `ActiveJob::Base`, etc.)

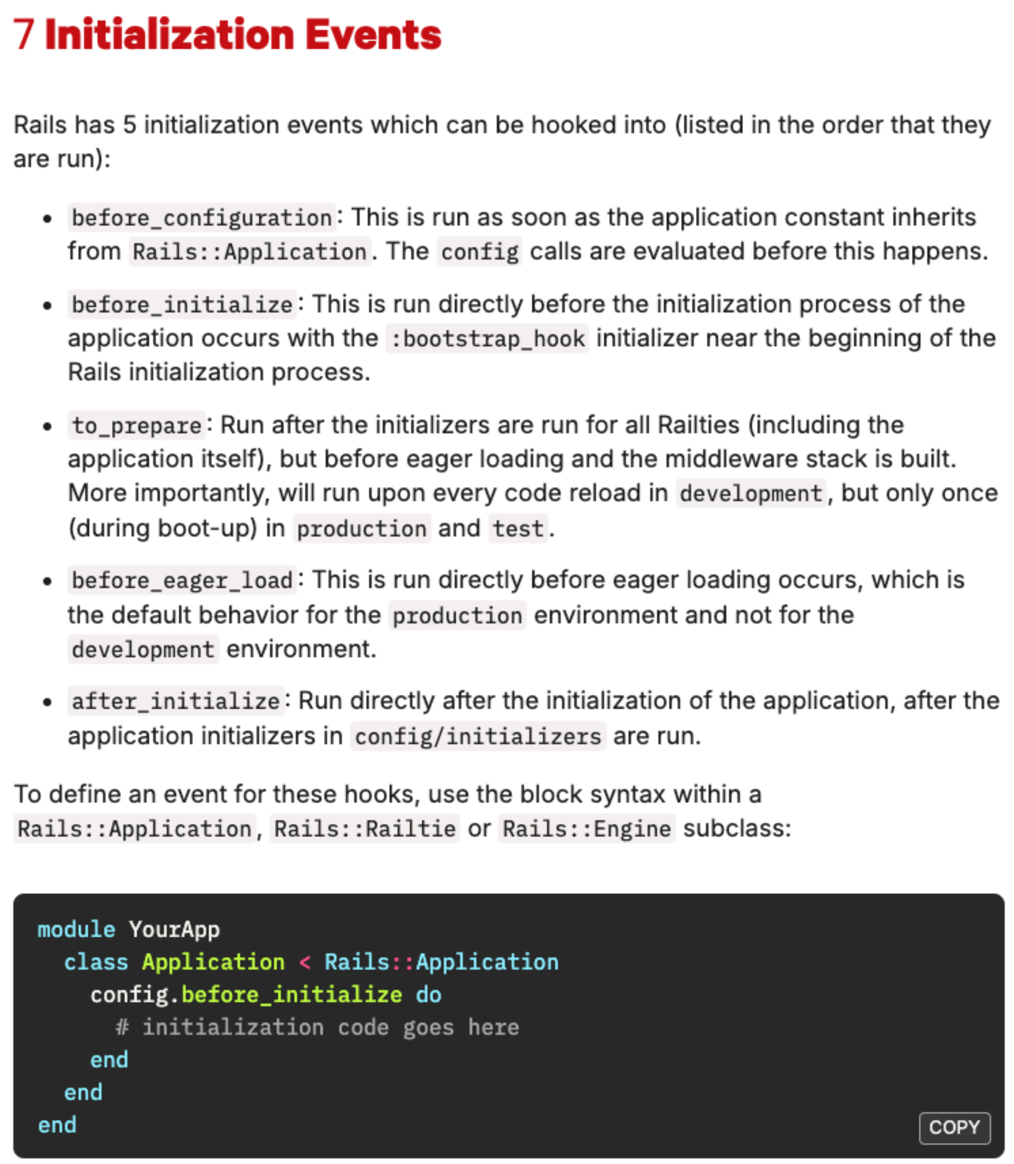
Gems do too, like Devise:

```
  ActiveSupport.run_load_hooks(:devise_controller, self)
```

Use these to add configuration only *when the constant is first accessed*, not before.

Also,
Rails Initialization Events

These won't completely defer constant loading, but still valuable tools to manage load order during application boot.



7 Initialization Events

Rails has 5 initialization events which can be hooked into (listed in the order that they are run):

- `before_configuration`: This is run as soon as the application constant inherits from `Rails::Application`. The `config` calls are evaluated before this happens.
- `before_initialize`: This is run directly before the initialization process of the application occurs with the `:bootstrap_hook` initializer near the beginning of the Rails initialization process.
- `to_prepare`: Run after the initializers are run for all Railties (including the application itself), but before eager loading and the middleware stack is built. More importantly, will run upon every code reload in `development`, but only once (during boot-up) in `production` and `test`.
- `before_eager_load`: This is run directly before eager loading occurs, which is the default behavior for the `production` environment and not for the `development` environment.
- `after_initialize`: Run directly after the initialization of the application, after the application initializers in `config/initializers` are run.

To define an event for these hooks, use the block syntax within a `Rails::Application`, `Rails::Railtie` or `Rails::Engine` subclass:

```ruby
module YourApp
  class Application < Rails::Application
    config.before_initialize do
      # initialization code goes here
    end
  end
end
```

COPY

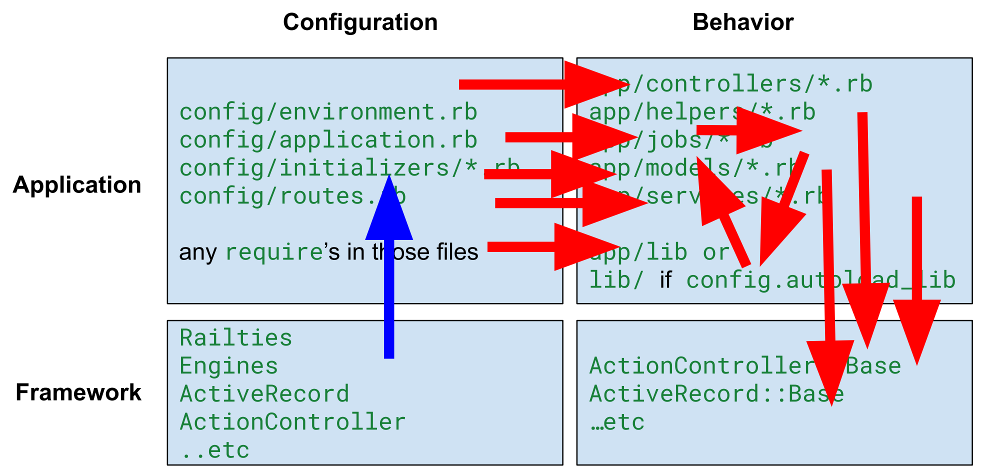
A very common problem 🙀

Using Devise:

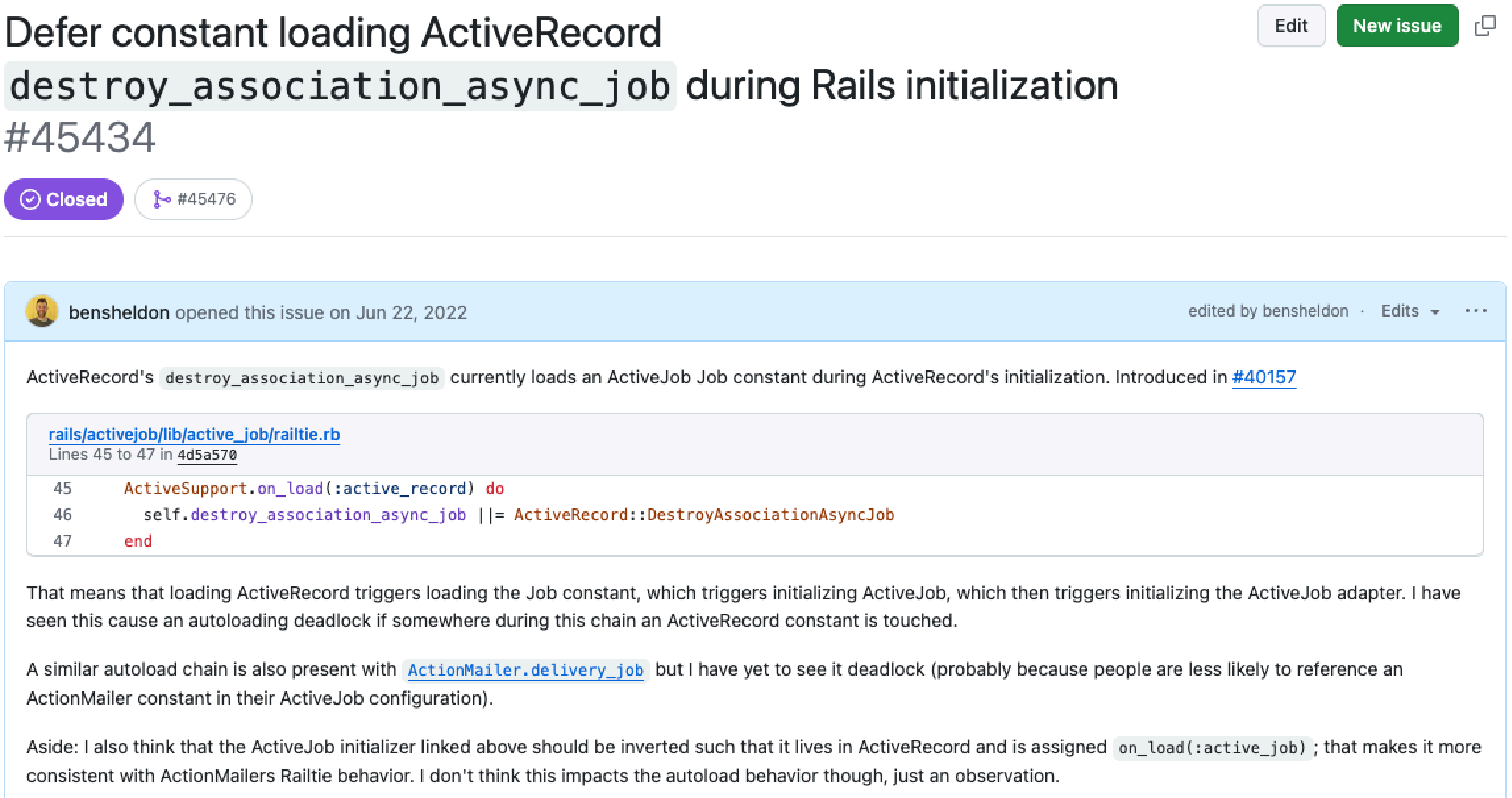
```ruby
# config/routes
devise_for :users
```

That twists around and then... 💥

```ruby
# gems/devise/lib/devise.rb
# ...
class Devise::Getter
  # ...
  def get
    @name.constantize # 💥
  end
end
```



In Rails itself, too 😞



Defer constant loading ActiveRecord `destroy_association_async_job` during Rails initialization #45434

Edit New issue

Closed #45476

bensheldon opened this issue on Jun 22, 2022 · edited by bensheldon · Edits

ActiveRecord's `destroy_association_async_job` currently loads an ActiveJob Job constant during ActiveRecord's initialization. Introduced in #40157

rails/activejob/lib/active_job/railtie.rb
Lines 45 to 47 in 4d5a570

```
ActiveSupport.on_load(:active_record) do
  self.destroy_association_async_job ||= ActiveRecord::DestroyAssociationAsyncJob
end
```

That means that loading ActiveRecord triggers loading the Job constant, which triggers initializing ActiveJob, which then triggers initializing the ActiveJob adapter. I have seen this cause an autoloading deadlock if somewhere during this chain an ActiveRecord constant is touched.

A similar autoload chain is also present with `ActionMailer.delivery_job` but I have yet to see it deadlock (probably because people are less likely to reference an ActionMailer constant in their ActiveJob configuration).

Aside: I also think that the ActiveJob initializer linked above should be inverted such that it lives in ActiveRecord and is assigned `on_load(:active_job)`; that makes it more consistent with ActionMailers Railtie behavior. I don't think this impacts the autoload behavior though, just an observation.

Some code for diagnosing

If your application is already fast *enough* (run it twice for Bootsnap), take the win 🫡:

```
$ time bin/rails runner "puts true"
true
bin/rails runner "puts true"  0.97s user 0.90s system 35% cpu 5.204 total
$ time bin/rails runner "puts true"
true
bin/rails runner "puts true"  0.90s user 0.67s system 62% cpu 2.503 total
```

If not, John Hawthorn's Vernier is an amazing Ruby profiler:

```
$ vernier run bin/rails runner "puts true"
starting profiler with interval 500
true
#<Vernier::Result 2.768903 seconds, 18 threads, 10928 samples, 3468 unique>
written to /var/folders/vm/p1vcrf3114s10pll1rm5pxbh0000gn/T/profile20240328-45567-7pu64h.vernier.json
```

... and then drop that `profile*.json` onto https://vernier.prof

More debugging ideas

```ruby
# config/initializers/explore_autoloading.rb
Rails.application.config.to_prepare do
  puts "Rails.config.to_prepare do ... end"
end

Rails.application.config.after_initialize do
  puts "Rails.config.after_initialize do ... end"
end

ActiveSupport.on_load(:active_record) do
  puts "ActiveSupport.on_load(:active_record) do ... end"
  puts caller # ⬅️ to see what calls it
end

ActiveSupport.on_load(:action_controller) do
  puts "ActiveSupport.on_load(:action_controller) do ... end"
end

# ... etc for LazyLoadHooks
```

And more...

```ruby
# ruby find_autoloaded.rb
require "./config/application"

autoloaded_constants = []

Rails.autoloaders.each do |loader|
  loader.on_load do |cpath, value, abspath|
    autoloaded_constants << [cpath, caller]
  end
end

Rails.application.initialize!

autoloaded_constants.each do |x|
  x[1] = Rails.backtrace_cleaner.clean(x[1]).first
end

if autoloaded_constants.any?
  puts "ERROR: Autoloaded constants were referenced during during boot."
  puts "These files/constants were autoloaded during the boot process, which will result in"\
    "inconsistent behavior and will slow down and may break development mode."\
    "Remove references to these constants from code loaded at boot."
  w = autoloaded_constants.map(&:first).map(&:length).max
  autoloaded_constants.each do |name, location|
    puts "#{name.ljust(w)} referenced by #{location}"
  end
  fail
end
```

(TIL `ActionText::ContentHelper` is prematurely loaded in ActionText 🤦)

Finishing up

- Keep Rails booting fast in development 🏎️
- Don't design your interfaces in ways that necessitate accessing autoloaded constants during boot.
- Defer accessing autoloaded constants until they're loaded, using Rails hooks if those constants live in Rails or Gems/Engines.

Lastly 🐈‍⬛ 🐈‍⬛

- Find a copy of these slides at `speakerdeck.com/bensheldon`
- Help me find a forever home for these two fine lads: Cameron and Monty.
- They have an Instagram 📸: `@redcarpetcats`