ADVANCING OUR RUBY SERIES
PART 1: LIBRARIES, GEMS & RAILS

by Konstantin Gredeskoul

WHAT'S THE ONE THING I WANT YOU TO LEARN AND APPLY AFTER THIS TALK?

I WANT EVERYONE TO MAKE A RUBY GEM TODAY. YOU CAN EVEN DO IT WHILE YOU WATCH THIS TALK.

(You do not have to publish it publicly)

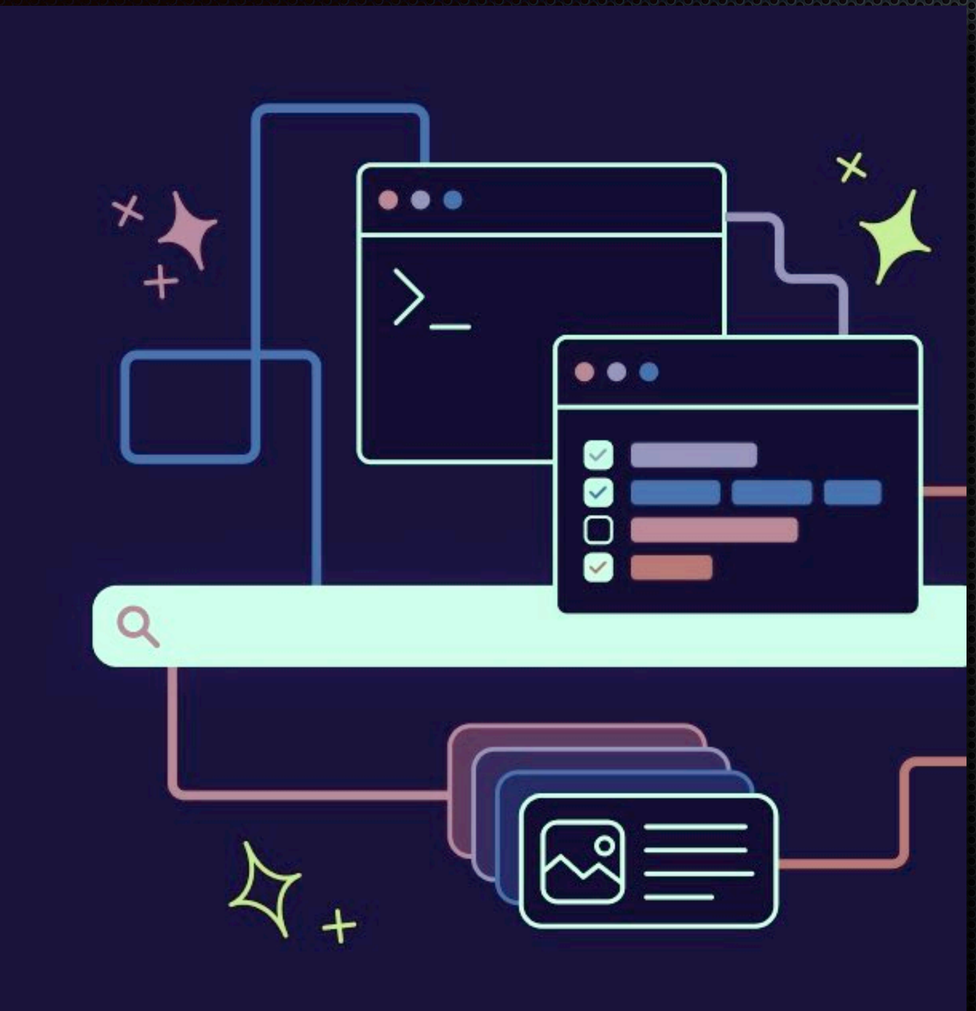
BUT FIRST, LET'S TALK ABOUT SOFTWARE DESIGN

Here is a million dollar question: what makes software great?

"Great software is the software that's easy to change."

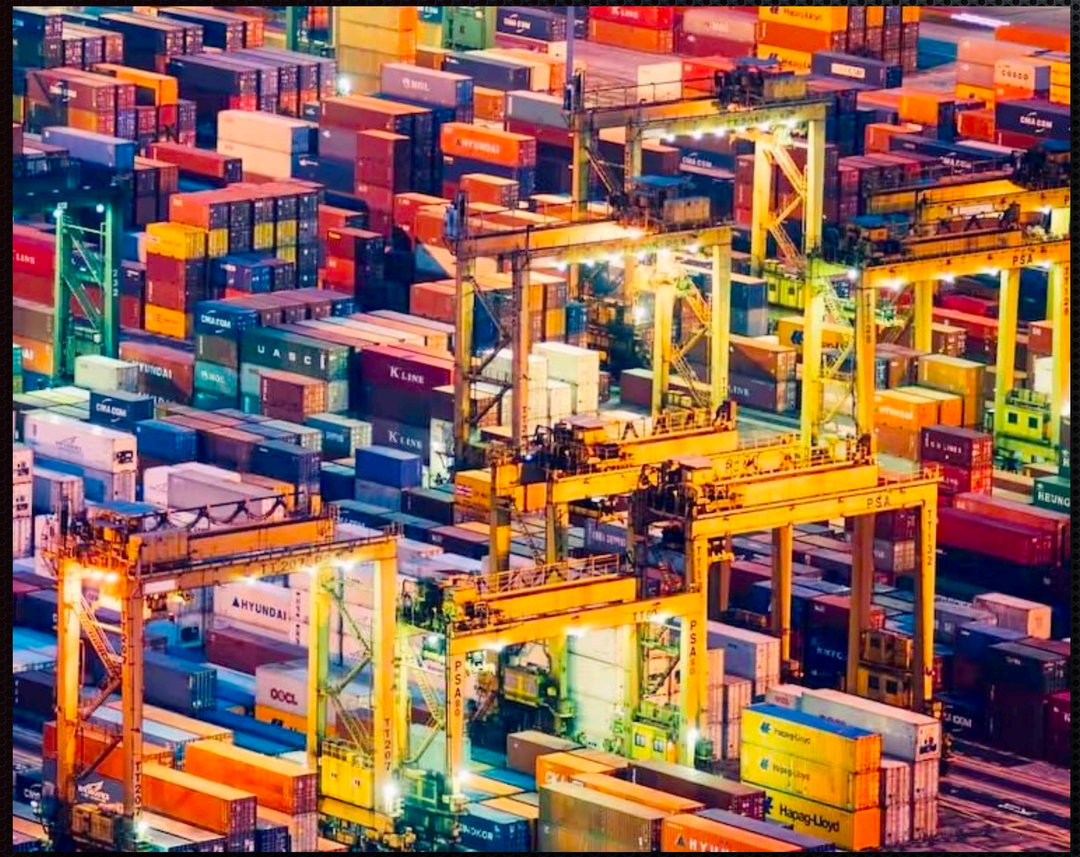
— DAVE THOMAS, AUTHOR OF "PROGRAMMING RUBY"

WHAT DOES IT MEAN, "EASY TO CHANGE"?



- It means that the software is easy to understand (it's either self-explanatory or it has great comments)
- It means that there are comprehensive tests covering every area, that allow for a change to be validated.
- It means that the classes are designed with single responsibility in mind, and that design patterns (like dependency injection) are used to decouple components.
- It also means that the source files are small and that the public APIs is clearly defined and ideally documented.

THIS BRINGS US TO A CONVERSATION ABOUT DEPENDENCIES



- Dependencies in software are inevitable
- Good dependencies are uni-directional: A → B (but B doesn't know about A)
- "Spaghetti Codebase" refers to a proliferation of bi-directional or even circular dependencies:

  A → B → C → A

HOW RAILS COOKED SPAGHETTI

- Rails introduced the concept of auto-loading everything.
- Every Rails file typically knows about every other Rails file.
- If you are writing a method on User model, you don't need to worry about resolving a dependency on a Permission model — it's there "for free".
- This innovation is the reason behind most Rails codebases becoming a spaghetti codebase over time.

HOW DOES RAILS DO THIS?

TODAY THE ANSWER IS ZEITWERK

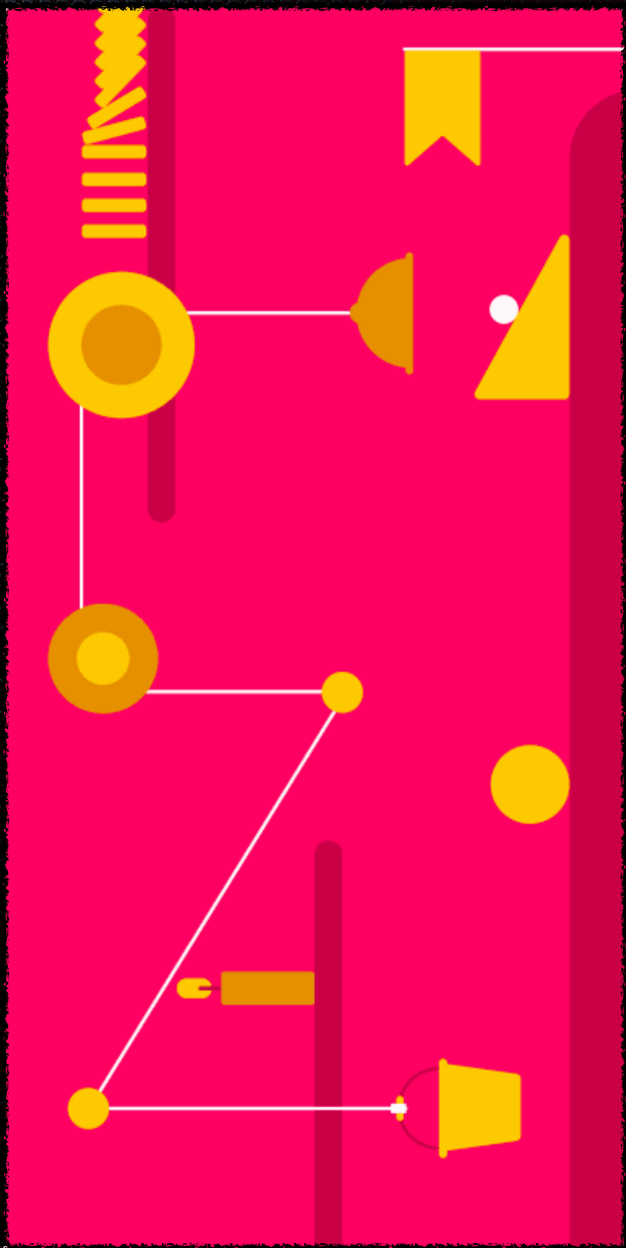
- Zeitwerk is an efficient and thread-safe code loader for Ruby.
- Given a conventional file structure, Zeitwerk is able to load your project's classes and modules on demand (autoloading), or upfront (eager loading).
- You don't need to write require calls for your own files, rather, you can streamline your programming knowing that your classes and modules are available everywhere. This feature is efficient, thread-safe, and matches Ruby's semantics for constants.
- Zeitwerk is also able to reload code, which may be handy while developing web applications. Coordination is needed to reload in a thread-safe manner.

https://github.com/fxn/zeitwerk

IS A SPAGHETTI CODEBASE EVEN AN ISSUE?

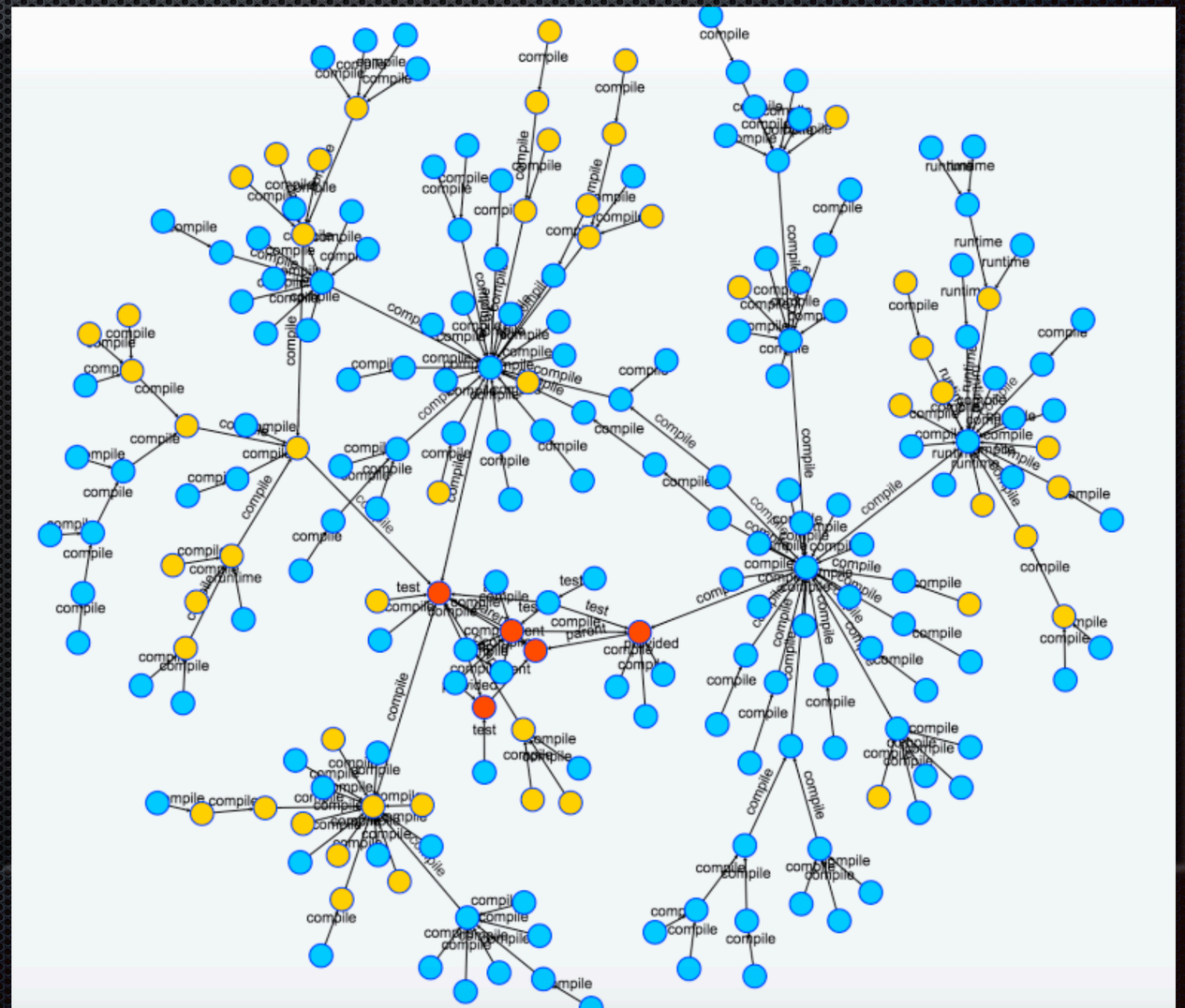
Well, yes it bloody is, pardon my Australian.

- It's hard to find things.
- It's easy to inadvertently duplicate existing functionality, since it's hard to find things.
- The larger the app, the slower are the tests and the boot time
- Finally, as companies grow it becomes problematic when 50+ engineers are working on the same codebase.
- The solution has been to split the codebase up into "pieces" (micro-services, extract gems, rails engines, libraries, etc) and assign a team to own each component.



DEPENDENCY HELL IS REAL

- This is a real diagram from Mozilla Firefox (see this tweet)
- What about this one?
- This diagram still has bidirectional deps, but overall it's infinitely better.



THIS BRINGS US TO
A VERY DANGEROUS BUT WIDE-SPREAD IDEA

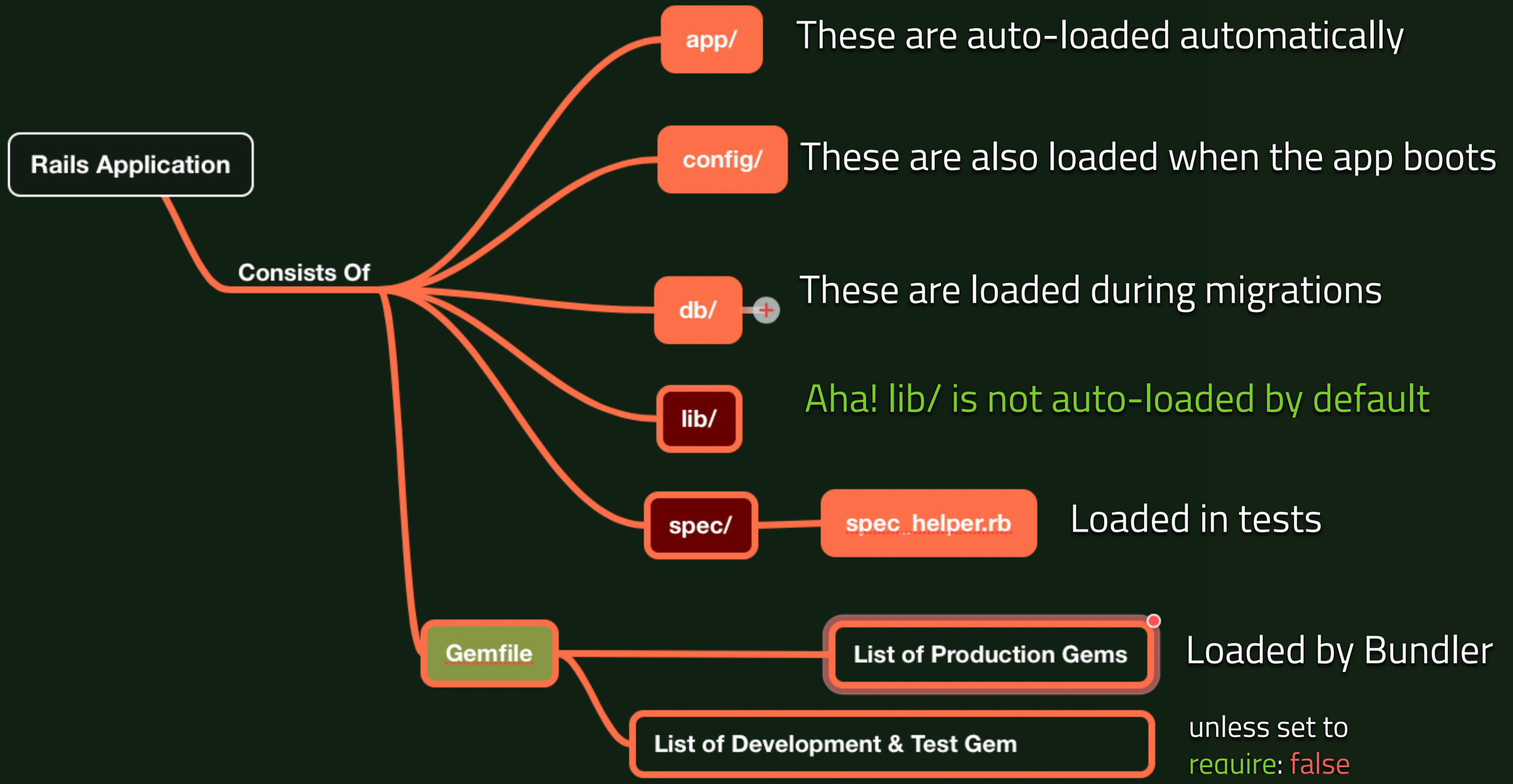
When I learnt Rails myself, I wrote Ruby like a procedural language for four years.

THE IDEA THAT ONE CAN BECOME A GREAT RAILS ENGINEER WITHOUT PROPERLY LEARNING RUBY... IS DEAD WRONG.

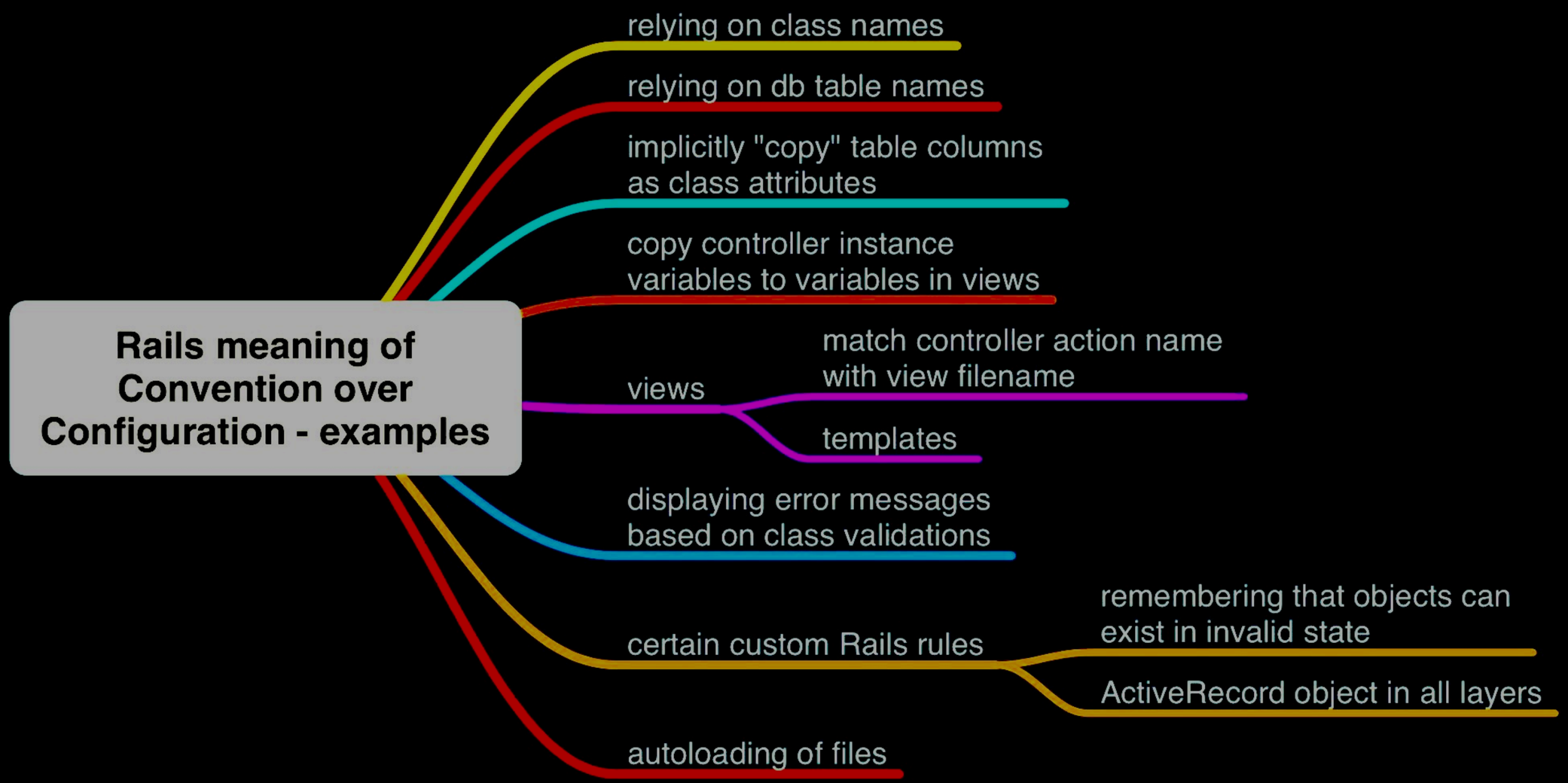
TO BE A GREAT RAILS ENGINEER, YOU MUST FIRST BECOME A GREAT RUBY ENGINEER.

OTHERWISE, IT'S SORT OF LIKE LEARNING HOW TO DRIVE A CAR WITHOUT LEARNING HOW TO APPLY BREAKS.

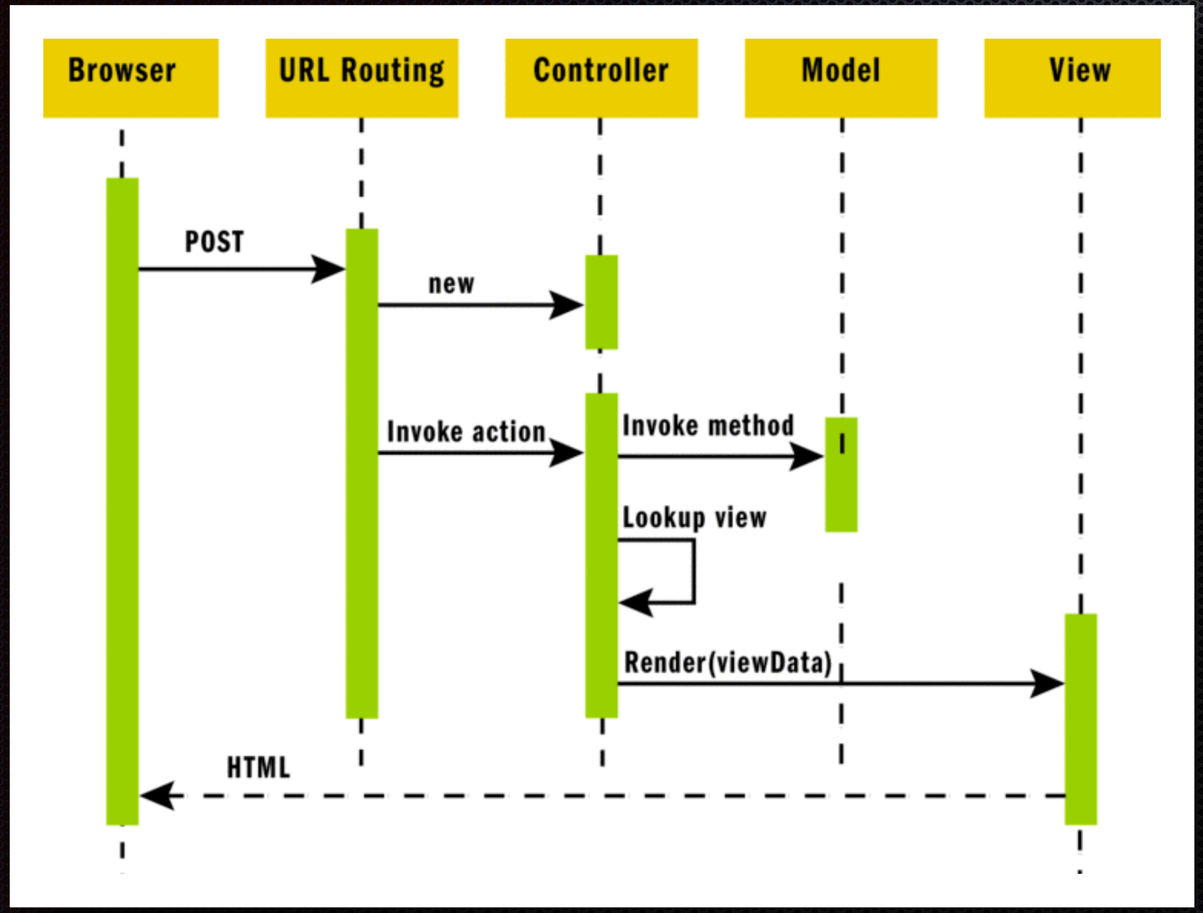
HOW DOES RAILS MAGIC WORK?



CONVENTION OVER CONFIGURATION



TYPICAL RAILS SEQUENCE DIAGRAM



- It helps us with the MVC pattern...
  - BUT....
- Where does the business logic reside?
- in Controllers? What if we want to offer both API and web controllers?
- Perhaps some shared Service objects?
- Maybe in the Models themselves? Who doesn't love "God Models" with their 300+ methods....
- Maybe we can extract some of the code elsewhere?
- Is this UML diagram a good representation of our application?

WHAT ABOUT EXCHANGE COMPARE?

WHAT FILES ARE TOO BIG?

```
❯ cd app
❯ find . -name '*.rb' -exec wc -l {} \; | sort -n | tail -10

    1481 ./models/insurance_full/plan.rb
    1527 ./controllers/statics_controller.rb
    1556 ./classes/qhp_importers/serff_importer.rb
    1712 ./models/enrollment_lead.rb
    2515 ./controllers/enroll/households_controller.rb
    2531 ./controllers/application_controller.rb
    2658 ./models/agent.rb
    3863 ./models/managed_person.rb
    4175 ./models/managed_application.rb
    5833 ./models/ffm_application.rb
```

WHAT ABOUT EXCHANGE COMPARE?

WHAT FILES HAVE TOO MANY METHODS?

```
❯ cd app
❯ find . -name '*.rb' -exec bash -c \
    "grep 'def ' {} | wc -l | tr -d '\n' ; \
     printf '\t%-90s\n' {}" \; | \
  sort -n | \
  tail -10
```

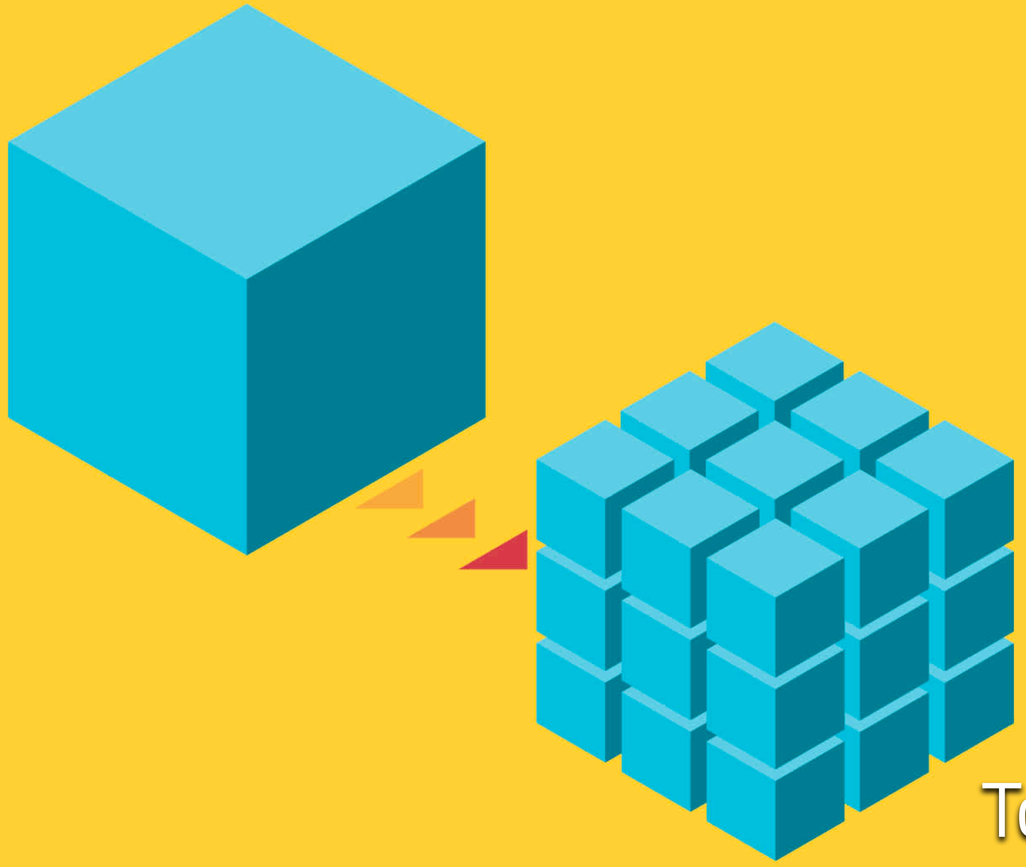
```
      75  ./classes/off_exchange/carrier_configurations/default_carrier_configuration.rb
      77  ./classes/qhp_importers/serff_importer.rb
      78  ./controllers/find_plans_controller.rb
      97  ./models/off_ex/application.rb
      98  ./classes/federal_exchange/data_hub_service.rb
     156  ./controllers/application_controller.rb
     192  ./models/managed_person.rb
     210  ./models/managed_application.rb
     222  ./models/agent.rb
     354  ./models/ffm_application.rb
```

SO THIS IS WHERE OUR BUSINESS LOGIC, HELPERS, API LOGIC IS LOCATED

- Its in files that are up to 5K lines long
- And have up to 354 methods
- I hope you see that our Rails app is in trouble.
- It needs our love. Love to refactor. Love of extracting the code that does not depend on anything, but other code depends on it.
- In order for this app to be maintainable by future engineers, we simply can not afford models that have 5K lines and hundreds of methods.

SO HOW DO WE GO

From here....

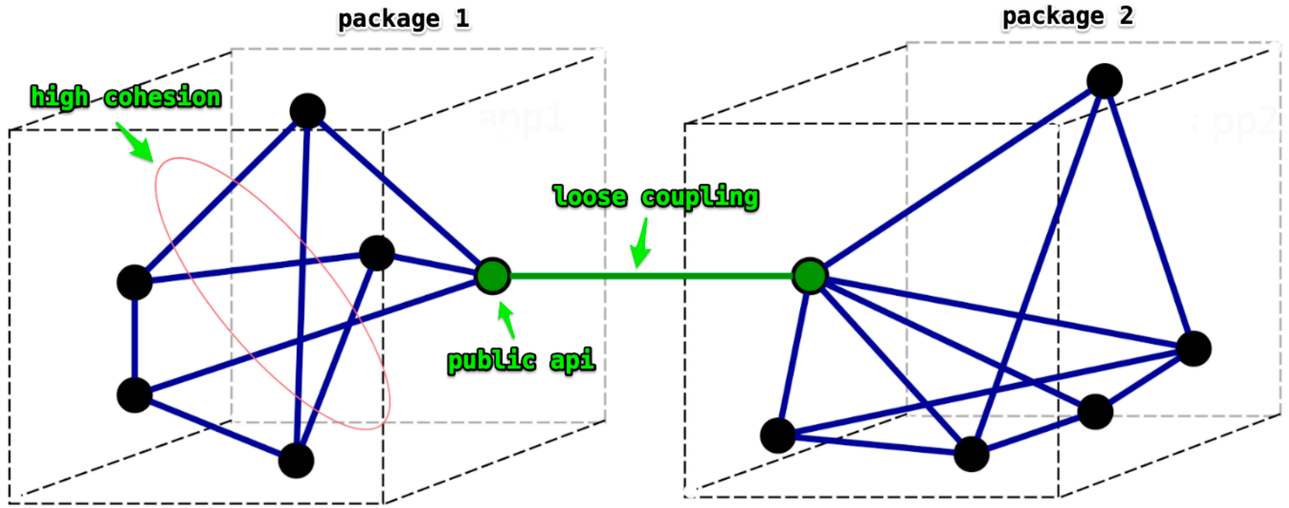


To here?

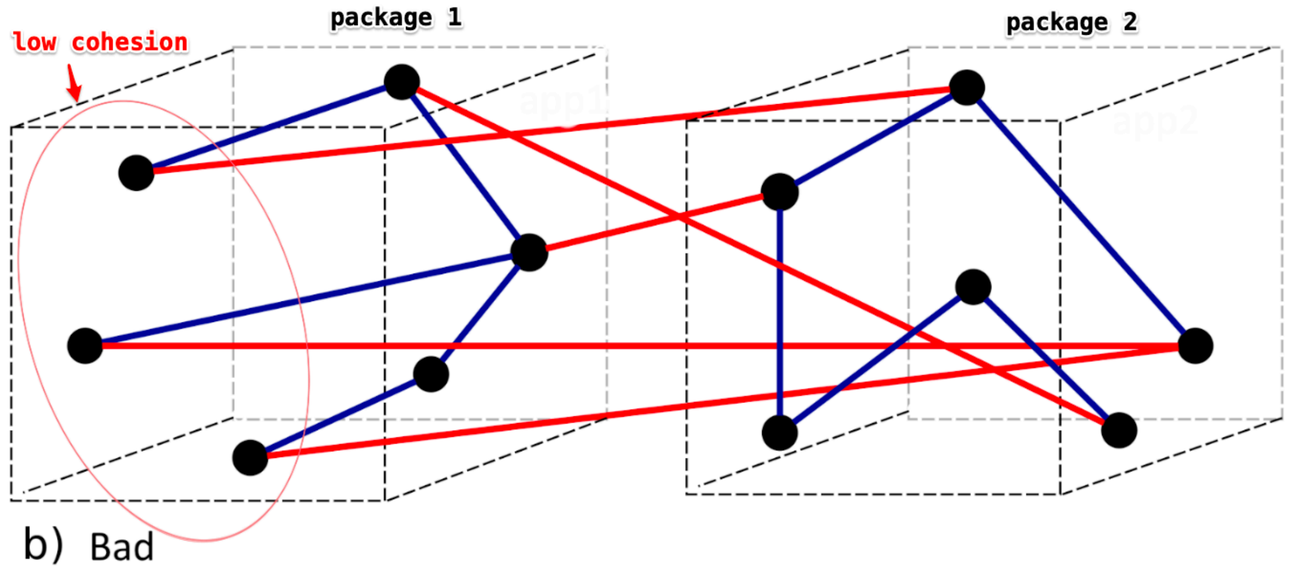
THERE ARE SEVERAL ESTABLISHED WAYS OF SPLITTING A MONOLITH

- What are our options:
  - Identify code blocks that **do not depend on anything else**. These often live in the /lib folder.
    - This code can be extracted into the ruby gems.
  - With business logic it's trickier.
    - But one of the most useful design patterns is called "The Facade": where one module or a class acts as a public API gateway to an entire sub-system.
    - This allows grouping related models together, and "packaging" them as a single unit.
    - Shopify built a tool that can verify that module boundaries have not been broken (it's called packwerk — https://github.com/Shopify/packwerk

PACKWERK VERIFIES THAT THE DECLARED PACKAGE CONTRACTS ARE VALID.



a) Good



b) Bad

COHESION & COUPLING

Structured design (SD) is concerned with the development of modules and the synthesis of these modules in a so-called "module hierarchy".

In order to design optimal module structure and interfaces two principles are crucial:

- *Cohesion* which is "concerned with the grouping of functionally related processes into a particular module", and,
- *Coupling* relates to "the flow of information or parameters passed between modules. Optimal coupling reduces the interfaces of modules and the resulting complexity of the software".

Structured design was developed by Larry Constantine in the late 1960s, then refined and published with collaborators in the 1970s

GEMS ARE RUBY LIBRARIES THAT CAN BE EXTRACTED FROM THE CODEBASE.

So, let's make one and see what they can do!

RUBY LIBRARIES

- Ruby libraries are typically packaged up using the gem format, and distributed using rubygems.org.
- A gem contains a number of useful pieces of information (this list is not exhaustive):
  - A bunch of metadata, like name, version, description, summary, email address, etc.
  - A list of all the files contained in the package.
  - A list of executables that come with the gem and their location in the gem (usually bin)
  - A list of directories that should be put on the Ruby load path (usually lib)
  - A list of other Ruby libraries that this library needs in order to function (dependencies)

SOME IMPORTANT REFERENCES

- Clarifying the Roles of the .gemspec and Gemfile
  - https://yehudakatz.com/2010/12/16/clarifying-the-roles-of-the-gemspec-and-gemfile/
- How to create a Ruby gem with Bundler
  - https://bundler.io/guides/creating_gem.html
- Gemsmith by Brooke Kuhlmann can create opinionated gems with many options
  - https://alchemists.io/projects/gemsmith
- HOE software by Ryan Davis, can produce a template-based gems
  - https://www.zenspider.com/projects/hoe.html

gem creation demo

bundle gem sqrt --exe --mit --ci=github --linter=rubocop

```
❯ bundle gem sqrt --exe --mit --ci=github  --linter=rubocop
Running `bundle gem sqrt --ci "github" --exe --git --linter "rubocop" --mit` with bundler 2.4.8
Creating gem 'sqrt'...
MIT License enabled in config
Code of conduct enabled in config
Changelog enabled in config
rubocop is already configured, ignoring --linter flag.
RuboCop enabled in config
Initializing git repo in /Users/kig/gems/sqrt
      create  sqrt/Gemfile
      create  sqrt/lib/sqrt.rb
      create  sqrt/lib/sqrt/version.rb
      create  sqrt/sig/sqrt.rbs
      create  sqrt/sqrt.gemspec
      create  sqrt/Rakefile
      create  sqrt/README.md
      create  sqrt/bin/console
      create  sqrt/bin/setup
      create  sqrt/.gitignore
      create  sqrt/.rspec
      create  sqrt/spec/spec_helper.rb
      create  sqrt/spec/sqrt_spec.rb
      create  sqrt/.github/workflows/main.yml
      create  sqrt/LICENSE.txt
      create  sqrt/CODE_OF_CONDUCT.md
      create  sqrt/CHANGELOG.md
      create  sqrt/.rubocop.yml
      create  sqrt/exe/sqrt
Gem 'sqrt' was successfully created. For more information on making a RubyGem visit https://bundler.io/guides/creating_gem.html
(r) 3.1.1  (n) v18.14.2  ~/gems                                  13:56:34  kig
```

Fix the sqrt.gemspec by removing TODOs

Then run "bundle install", and...

```
 127 ❯ bundle exec rspec

Sqrt
  has a version number
  does something useful (FAILED - 1)

Failures:

  1) Sqrt does something useful
     Failure/Error: expect(false).to eq(true)

       expected: true
            got: false

       (compared using ==)

       Diff:
       @@ -1 +1 @@
       -true
       +false

     # ./spec/sqrt_spec.rb:9:in `block (2 levels) in <top (required)>'

Finished in 0.01599 seconds (files took 0.05101 seconds to load)
2 examples, 1 failure

Failed examples:

rspec ./spec/sqrt_spec.rb:8 # Sqrt does something useful
```

THANK YOU!

- Some of my gems that are published:
  - https://rubygems.org/profiles/kigster
- My blog:
  - https://kig.re/