Safe, flexible tests

Fritz Meissner

iftheshoefritz.com

~~onesizefritzall.com~~

Prodigy
Finance

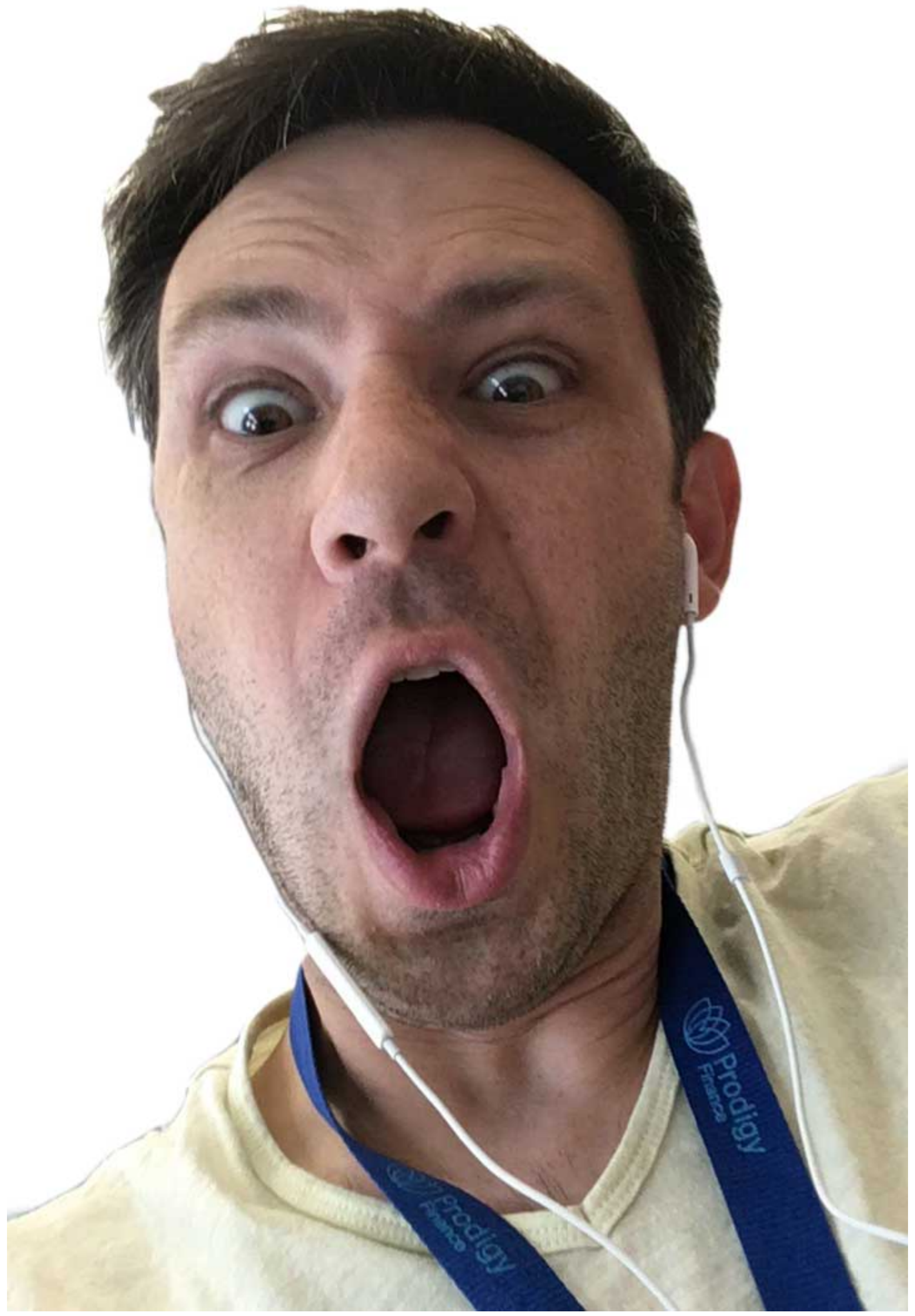
Team Unicode

team emojis

Fritzmojis

:fritzcited

:fritzplosion



Take a breath

Scope

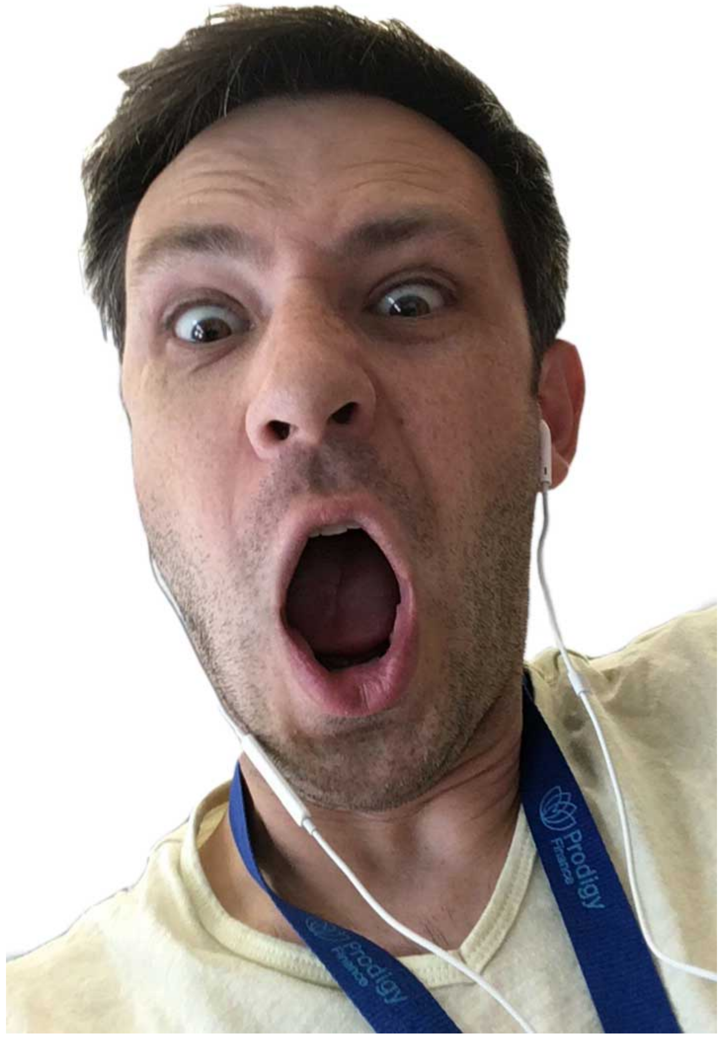
RAILS

Avoiding

- TDD is dead?
- Testing vs monitoring in production
- Flaky tests
- Manual tests

Safe

Tell me when it's broken



Before you write

Not just the how

I don't wanna!

- Context-switching
- Time to write
- Too complicated
- Speed

Flexible

Changing the code (later)

Fixing unrelated tests

Testing tools and techniques

Dependencies

What are dependencies?

Code uses other code

- call method of object X
- query attribute Y
- update column Z
- etc

Simple dependencies

dependency

```
def my_method(x)
  x.to_str
end
```

More complex dependencies

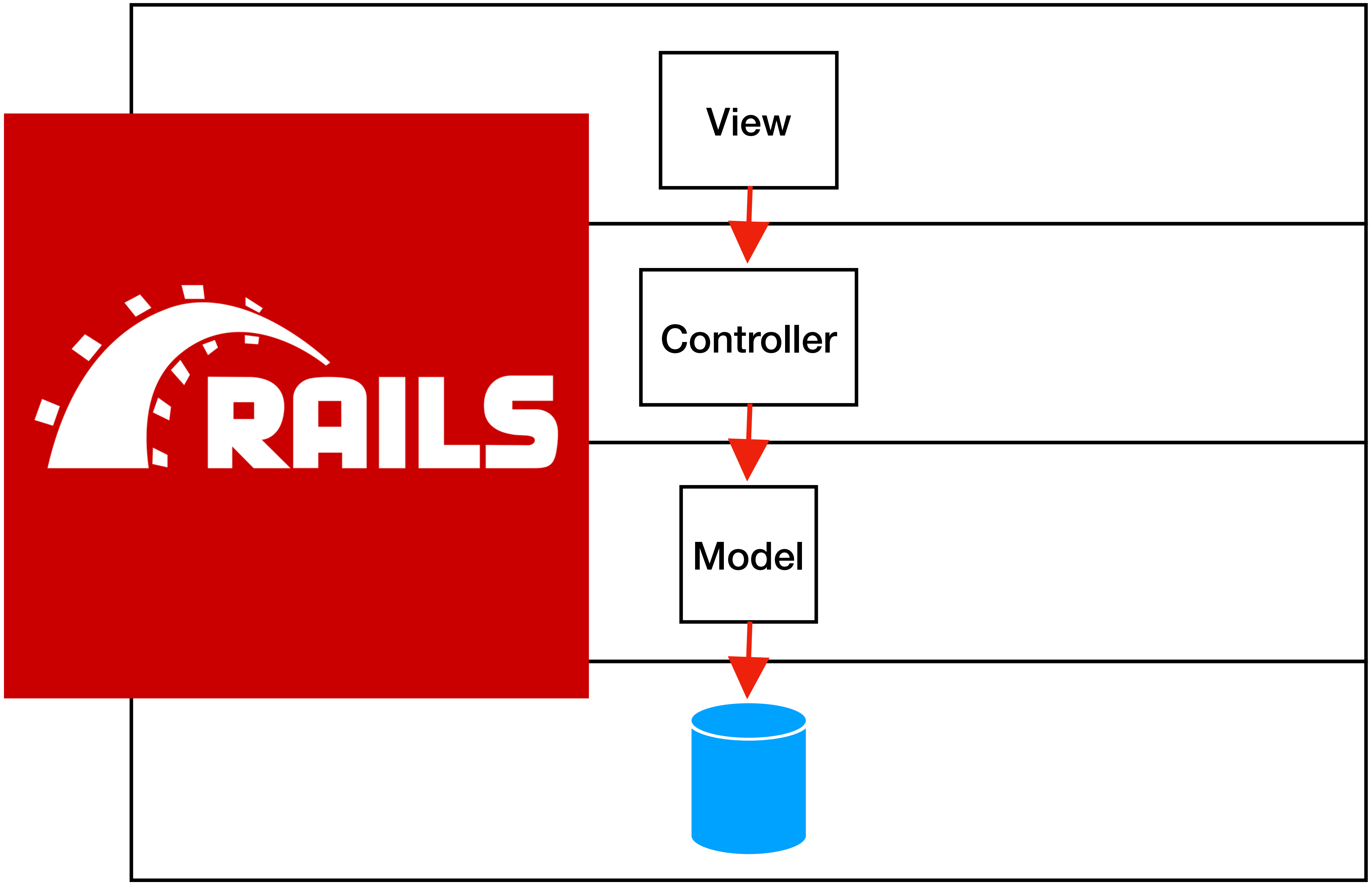
dependency

```
def my_method(user)
  user.street_address
end
```

dependency's dependency

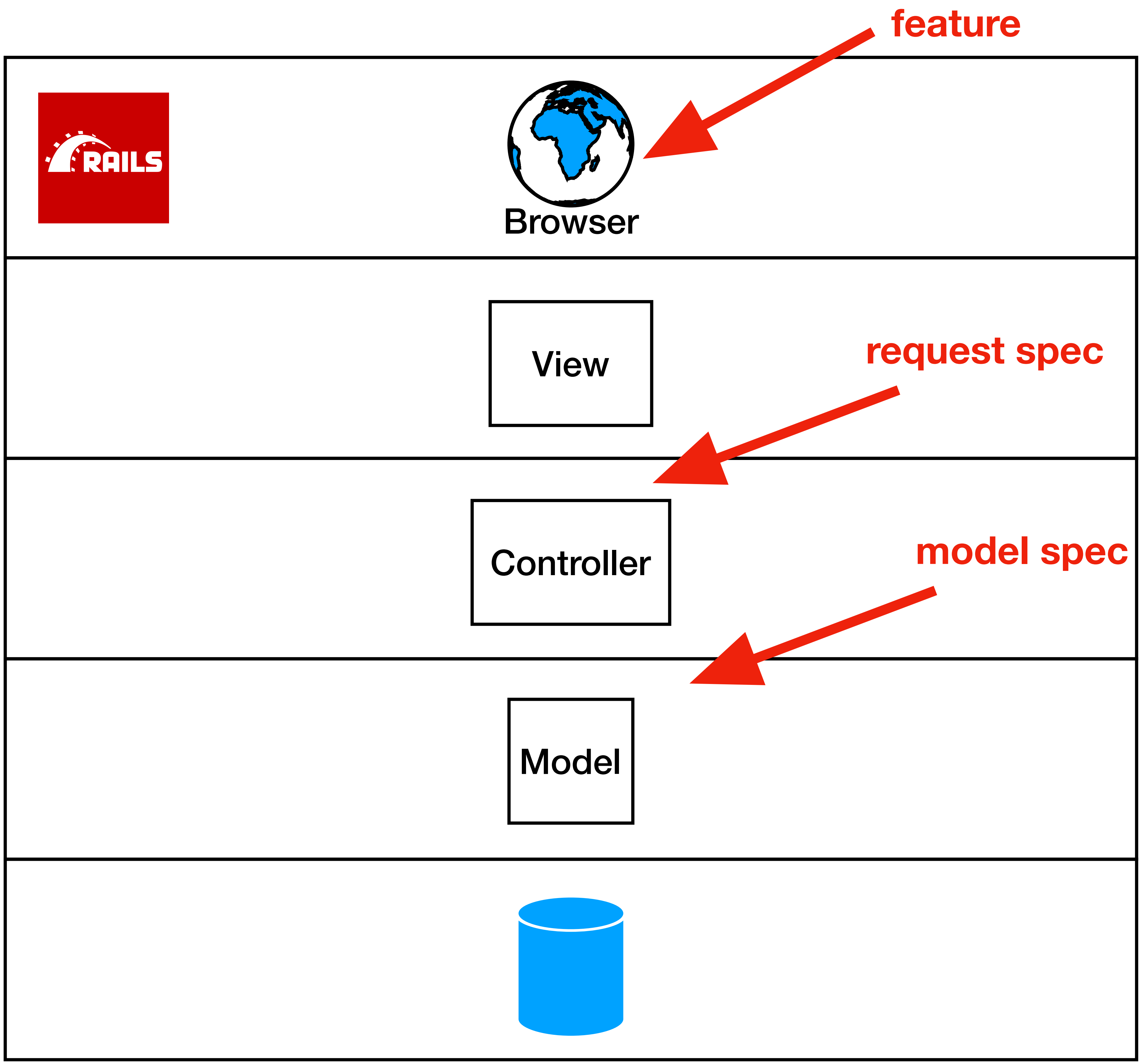
```
class User
  def street_address
    account.address.street
  end
end
```

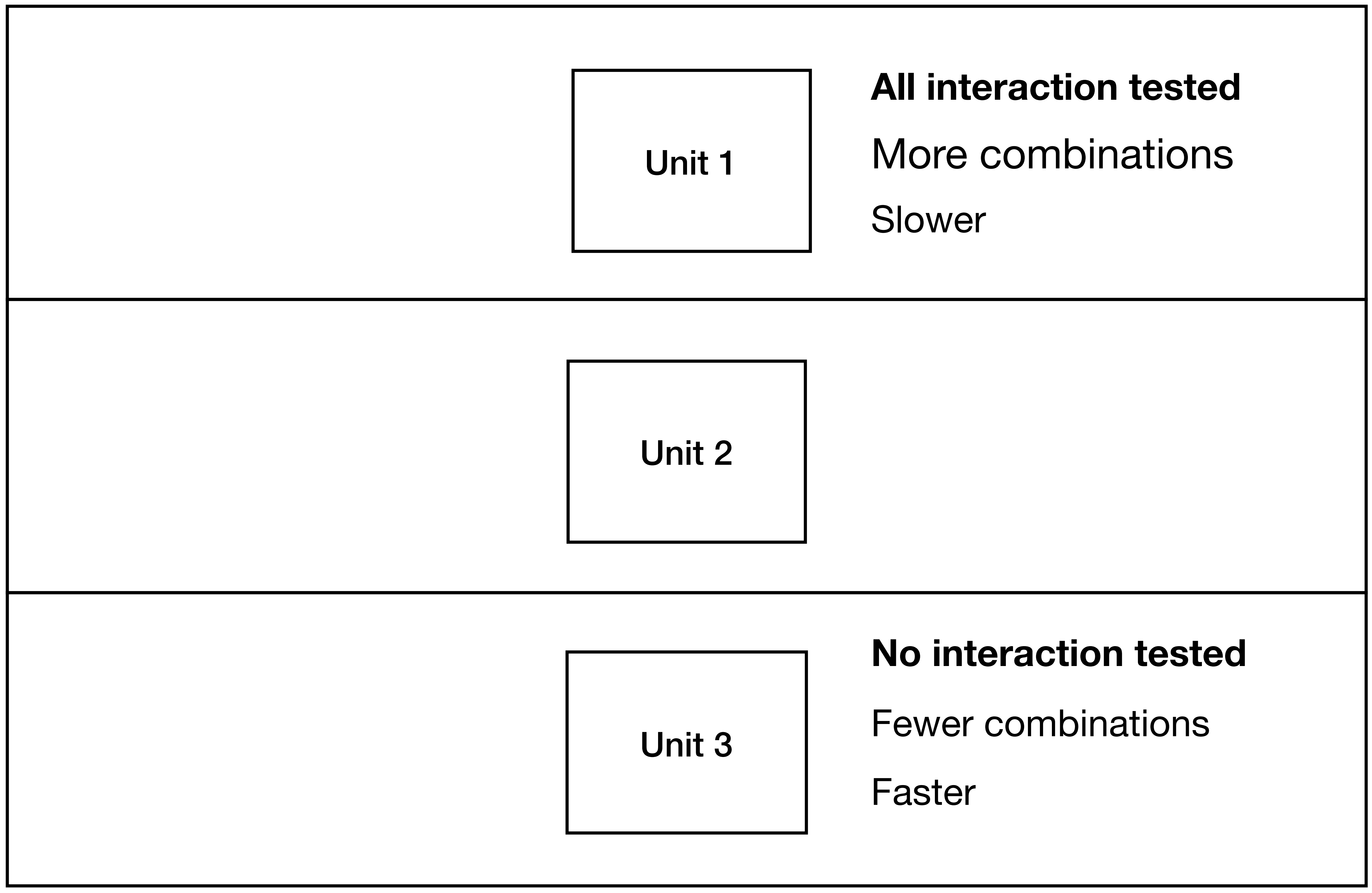
Also

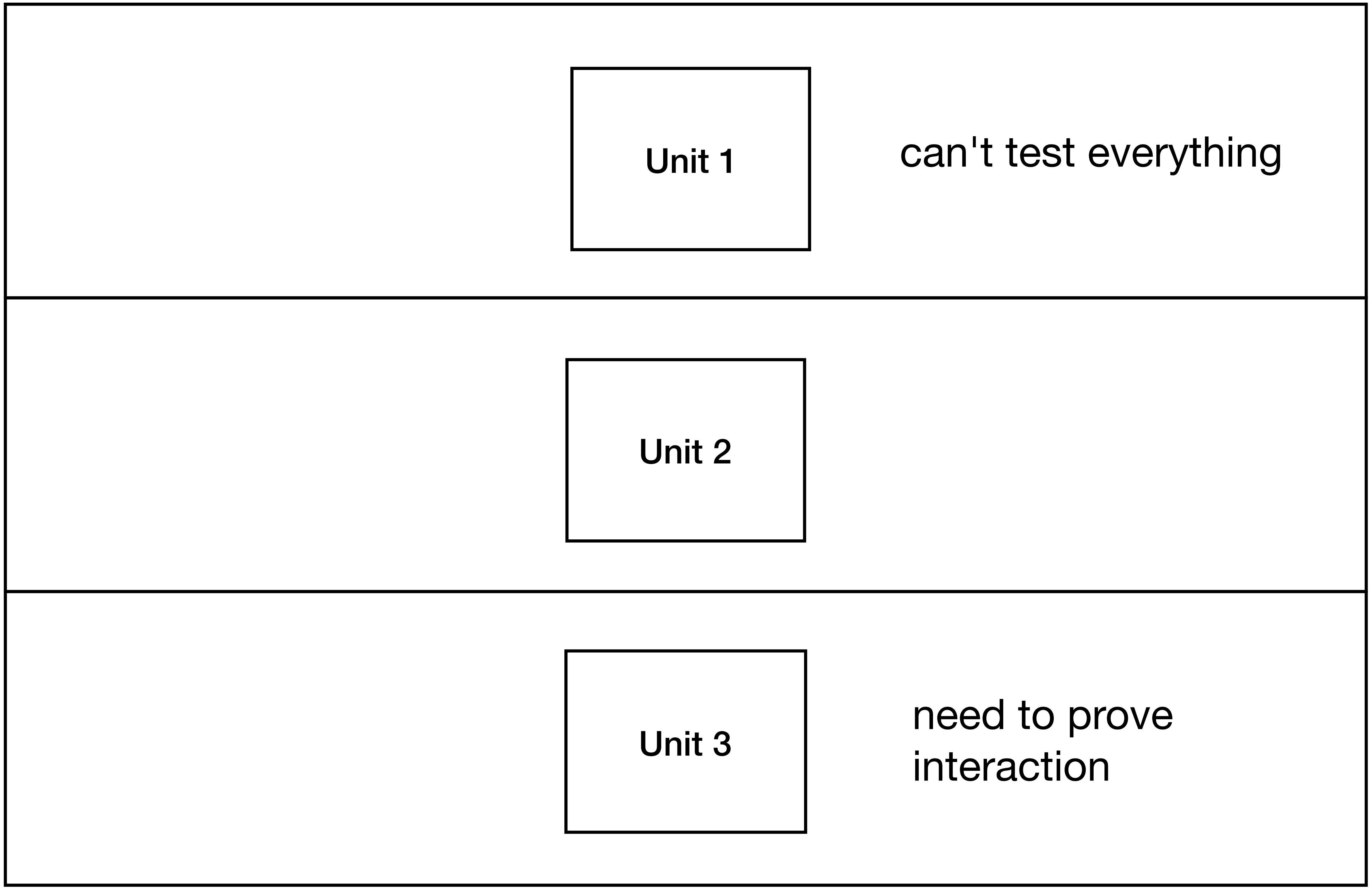


Crucial decisions

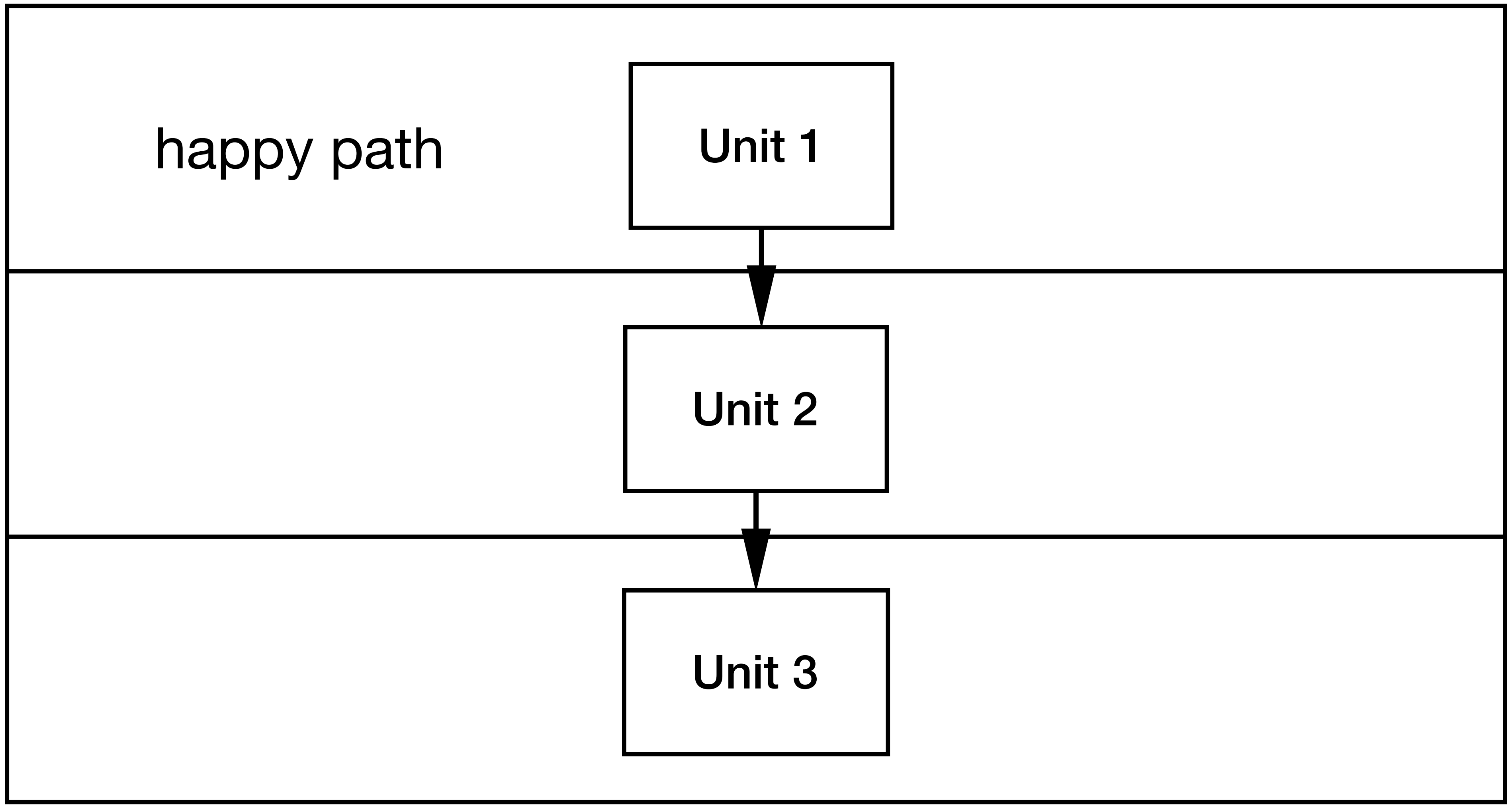
1. Where do I test


RAILS
Browser
View
Controller
Model
feature
request spec
model spec

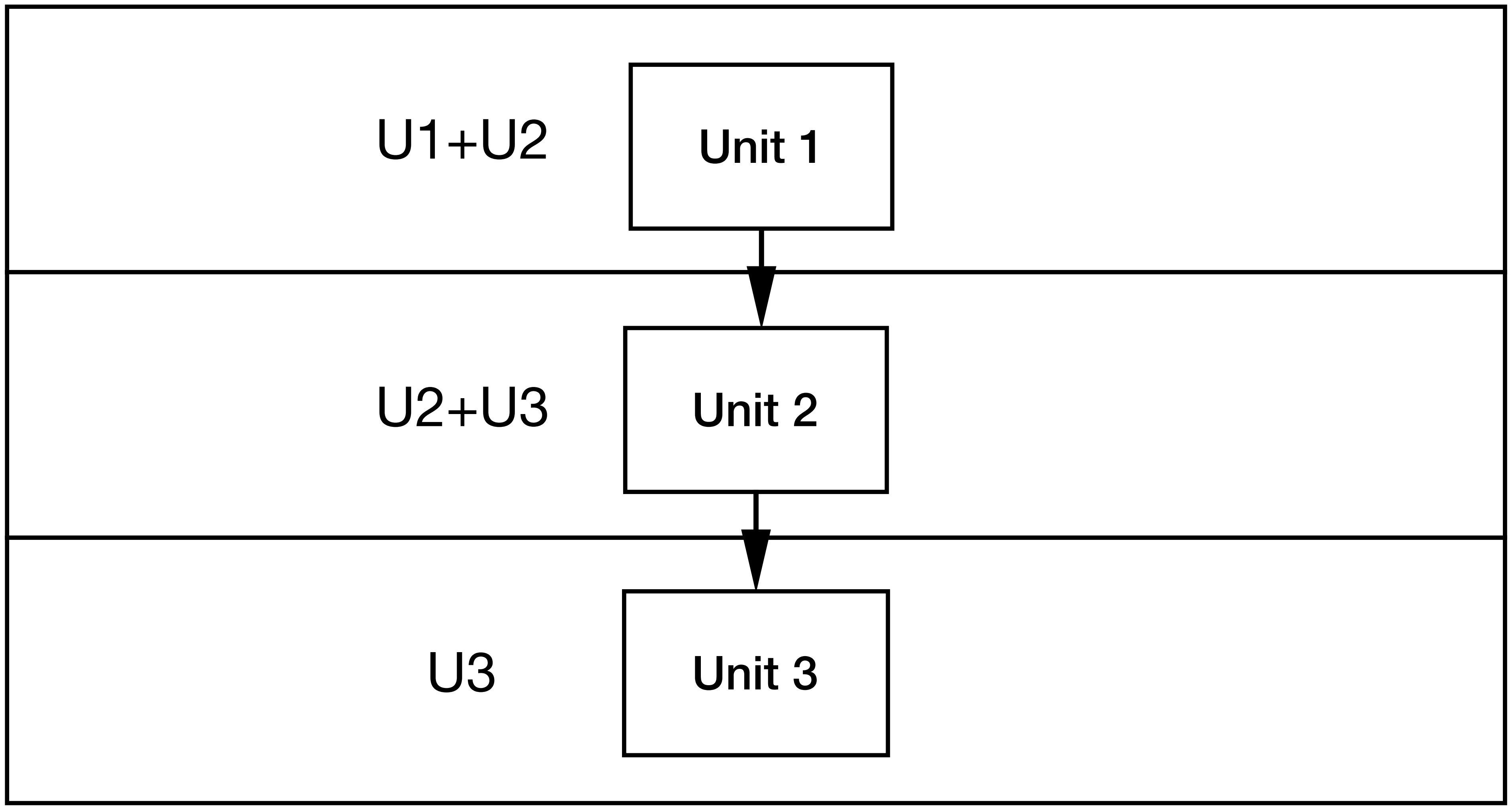

Unit 1
All interaction tested
More combinations
Slower
Unit 2
Unit 3
No interaction tested
Fewer combinations
Faster

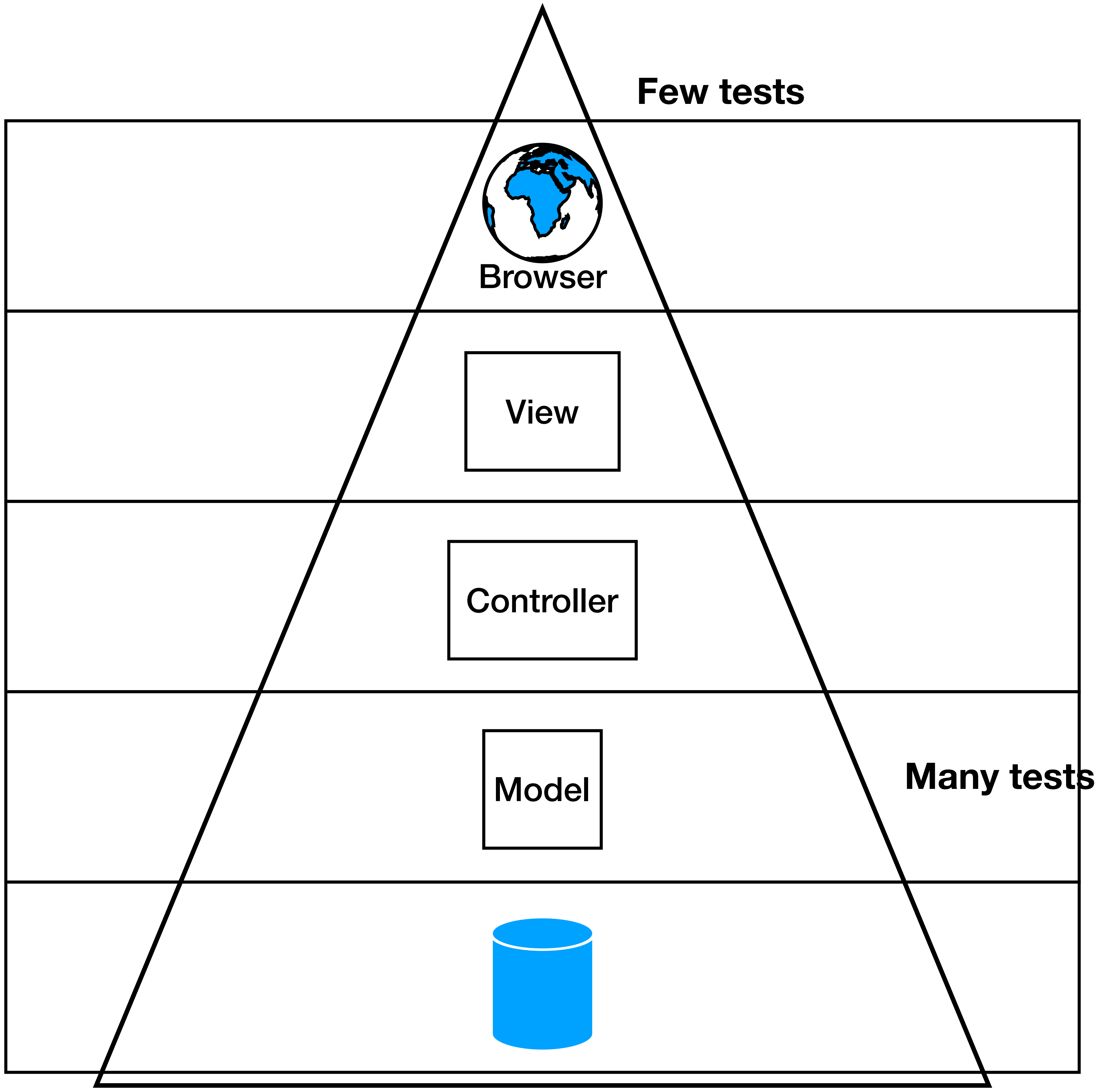

Unit 1
can't test everything
Unit 2
Unit 3
need to prove interaction

Verify vertical slice of interaction once



Test in detail with less collaboration




Few tests
Browser
View
Controller
Model
Many tests

2. Dependencies affect tests

The code

dependency

```ruby
def my_method(user)
  user.street_address
end
```

dependency's dependency

```ruby
class User
  def street_address
    account.address.street
  end
end
```

The test it needs

```
it 'street address of the user' do
  user = ...
  user.account = ...
  user.account.address = ...
  user.account.address.street = "Hope Str."

  expect(my_method(user)).to eq('Hope Str.')
end
```

Creating the real objects

Assembly Required

```
it 'street address of the user' do
  user = ...                          ← dependency
  user.account = ...                  ← dependency's dependency
  user.account.address = ...          ← ...
  user.account.address.street = "Hope Str."

  expect(my_method(user)).to eq('Hope Str.')
end
```

Warning

Changes

```
it 'street address of the user' do
  user = ...          ←  dependency
  user.account = ...          ←  dependency's dependency
  user.account.address = ...          ←  ...
  user.account.address.street = "Hope Str."

  expect(my_method(user)).to eq('Hope Str.')
end
```

Changes

Mitigation: factory_bot

```
it 'gets the street address of the user' do
  user = ...
  user.account = ...
  user.account.address = ...
  user.account.address.street = "Hope Str."

  expect(my_method(user)).to eq('Hope Str.')
end
```

Mitigation: factory_bot

```ruby
it 'street address of the user' do
  user = create(:user, :with_address)
  user.address.street = 'Hope Str.'

  expect(my_method(user)).to eq('Hope Str.')
end
```

Mitigation: factory_bot

Advantages:

- hide unnecessary knowledge about how to create things
- change all uses of (e.g.) **Account** or **User** in one place only
- sensible defaults

Warning

Discipline

Keep factories valid and minimal

https://thoughtbot.com/blog/factories-should-be-the-bare-minimum
https://thoughtbot.com/blog/mystery-guest

Starting from here

```
it 'street address of the user' do
  user = ...
  user.account = ...
  user.account.address = ...
  user.account.address.street =
"Hope Str."

 expect(my_method(user)).to
     eq('Hope Str.')
end
```

Mitigation: DRY Rspec

```
before do
  user = ...
  user.account = ...
  user.account.address = ...
  @address = user.account.address
end

it 'street address' do
  @address.street = 'Hope Str.'
  ...

it 'street number' do
  @address.street_number = '99B'
 ...
```

Requires discipline

```
describe MyClass do
  describe '#my_method' do
    context 'when there is an X' do
      before do
        # lines
        # lines
      end

      context 'and there is a Y' do
        before do
          # more lines
        end

        context 'and there is a Z with a P and a Q' do
          it 'does something' do
          end
        end
      end
    end
  end

  describe '#another_method' do
    ...
  end
end
```

X 20

Setup is like comments

- Comments can be trustworthy or they can be misleading
- But these comments have side effects

Test Doubles

Instead of this

```
user = ...
user.account = ...
user.account.address = ...
user.account.address.street = "Hope Str."

expect(my_method(user)).to eq('Hope Str.')
```

This

```
user = double(
   'User', street_address: 'Hope Str.'
)

expect(my_method(user)).to
 eq('Hope Str.')
```

Or this

```
user = User.new
allow(:user).to receive(
  :street_address
).and_return('Hope Str.)

expect(my_method(user)).to eq('Hope Str.')
```

Test Double Advantages

- avoid code you don't want to run
- avoid dependency's dependencies

Warning: False Negatives

```ruby
def my_method(user)
  user.street_address
end

class User
  def street_address
    user.account.street.address
  end
end
```

Warning: False Negatives

```ruby
user = double(
  'User', street_address: 'Hope Str.'
)

expect(
  my_method(user)
).to eq('Hope Str.')
```

Warning: False Negatives

```
def my_method(user)
  user.street_address
end

class User
  def street_address
    nil
  end
end
```

broken!!!

Test still passes

```
user = double(
  'User', street_address: 'Hope Str.'
)

expect(
  my_method(user)
).to eq('Hope Str.')
```

Test still passes

```
user = double(
    'User', street_address: 'Hope Str.'
  )
  ...
```

no implementation check

indicates another test

Warning: Tight Coupling

- have to change test doubles if the dependency's implementation changes
- requires practice and expertise
- works well when dependencies are stable

3. How my code interacts with dependencies

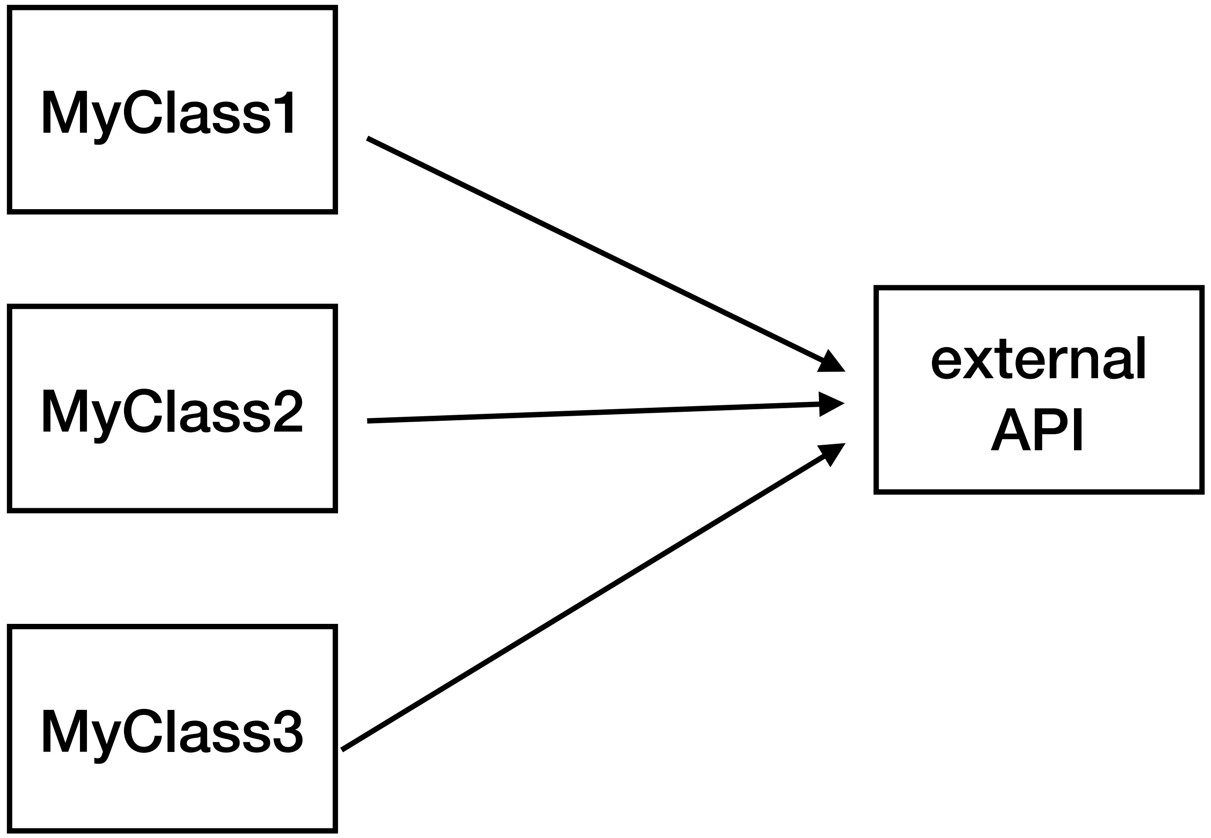
All about dependencies

- dependencies make tests difficult
- every solution comes with warnings
- can I write code with easier-to-use dependencies?

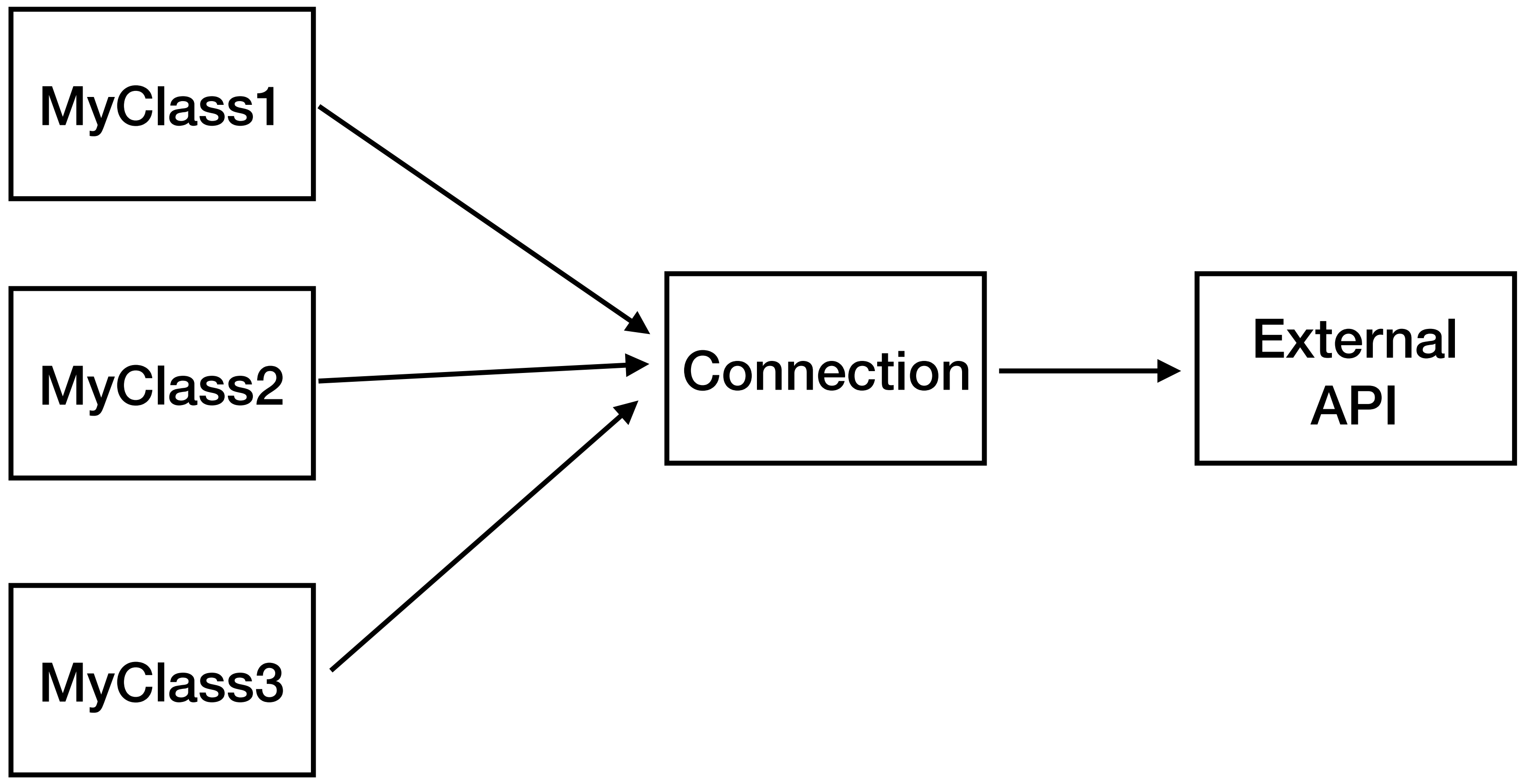
Déjà Vu

Tests are like clients

Separation of concerns



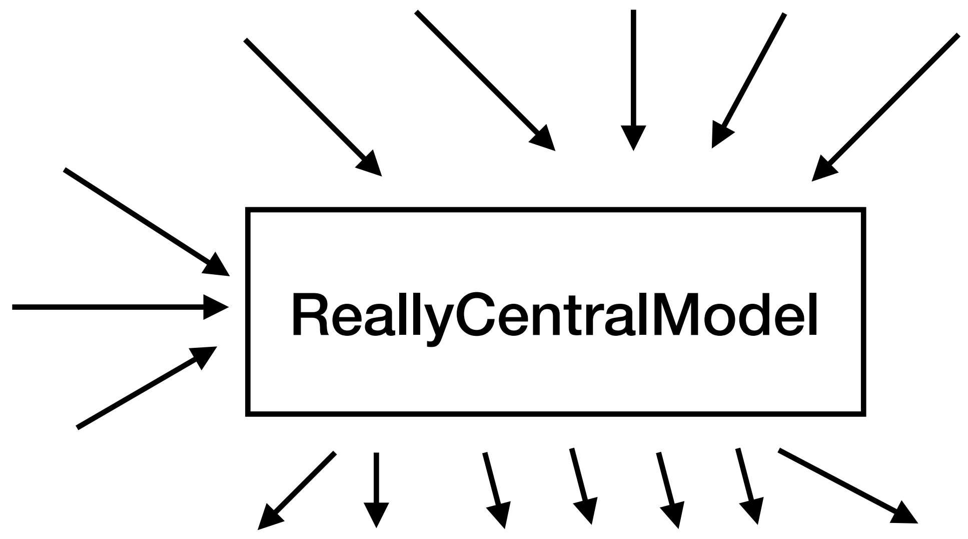
Isolate external thing



"Refactoring code that accesses external Systems"

https://www.martinfowler.com/articles/refactoring-external-service.html

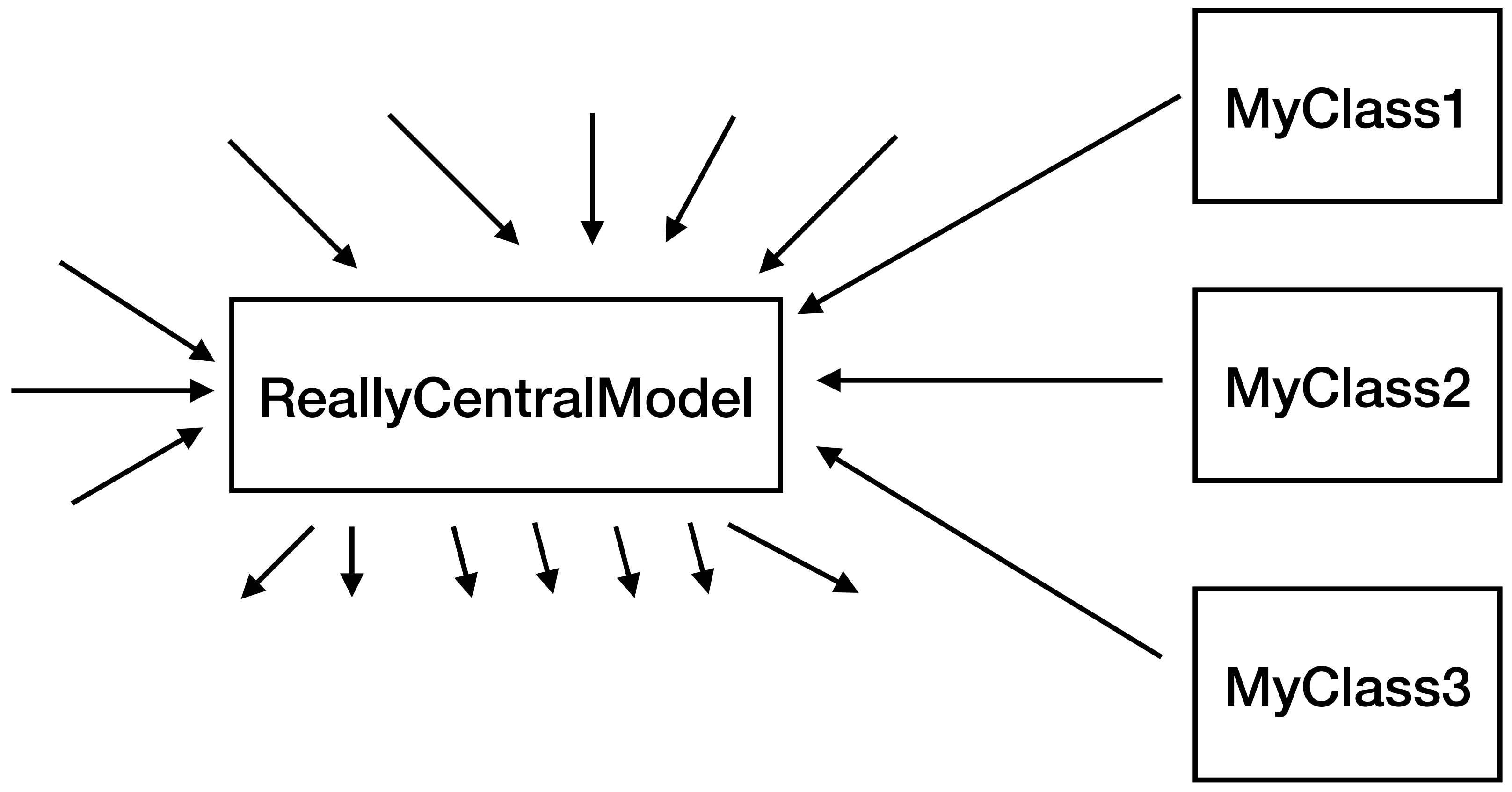
Disconnect from code with lots of dependencies



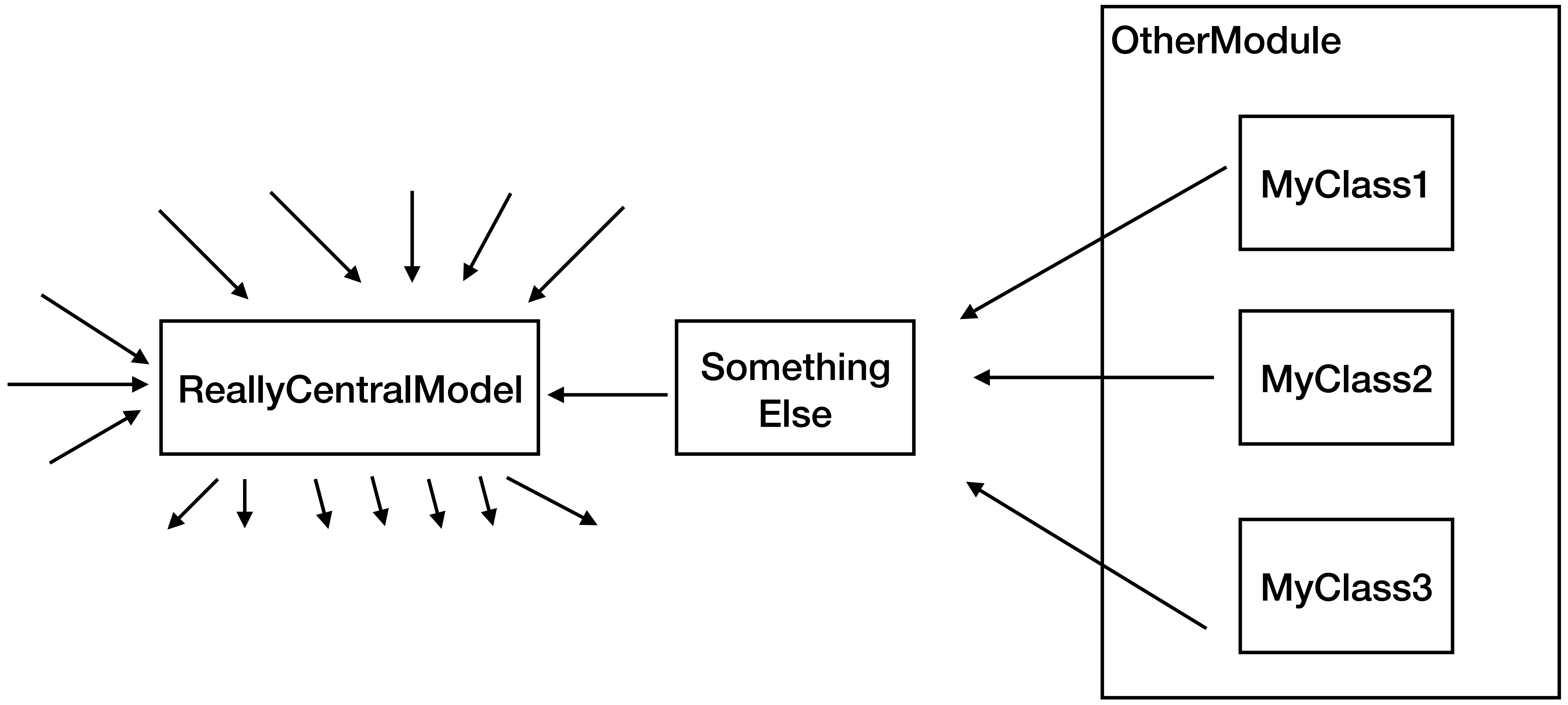
Why disconnect

- if I touch this code, it could affect lots of other code
- if someone else touches this code, it will affect me

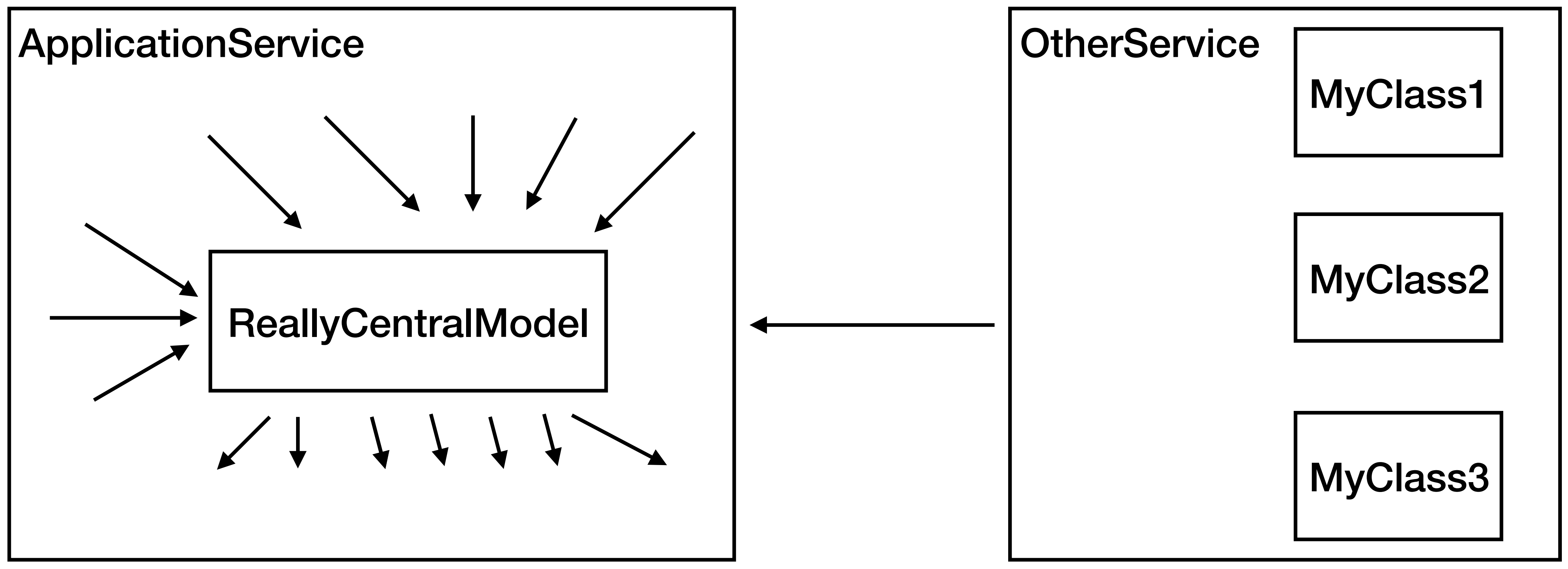
Not this



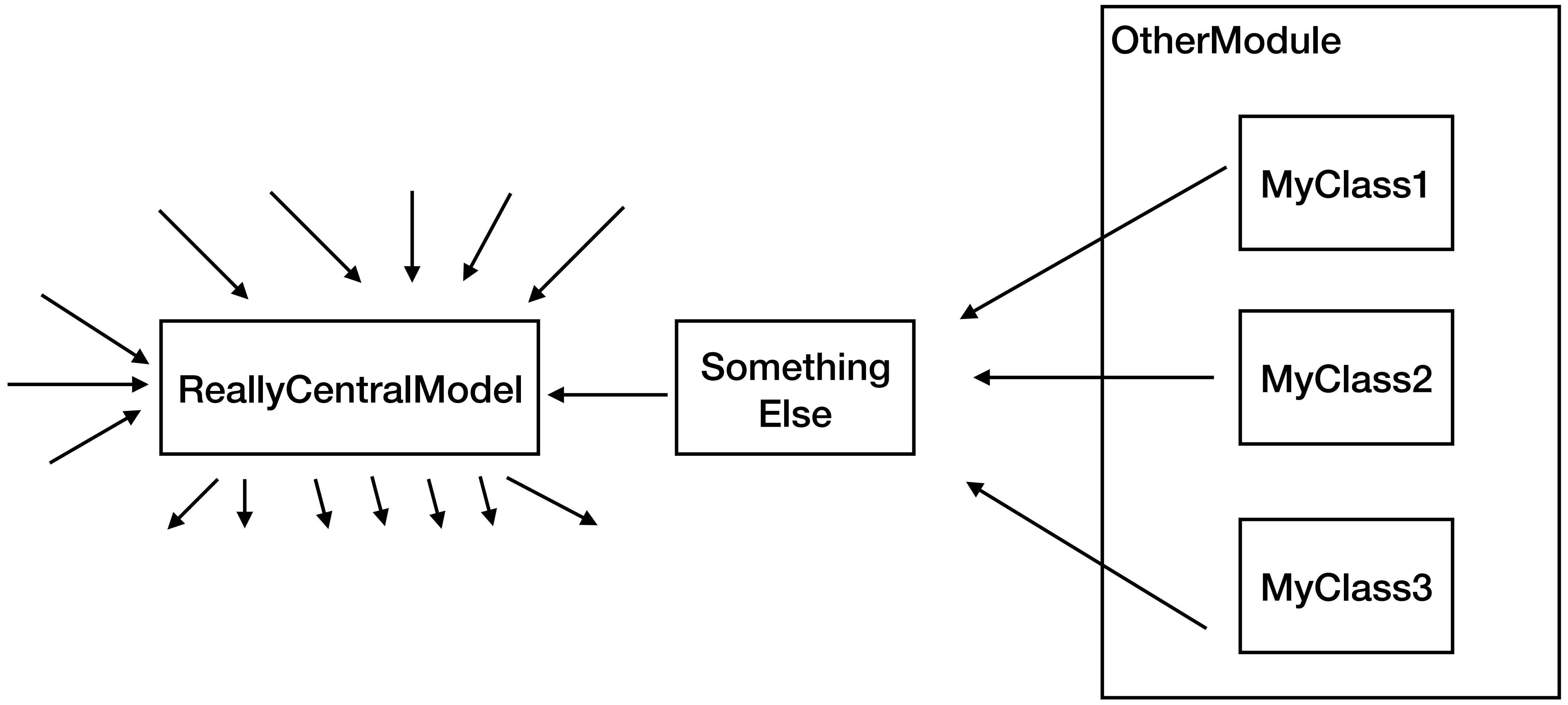
This


OtherModule
MyClass1
MyClass2
MyClass3
ReallyCentralModel
Something Else

Or even this


ApplicationService
ReallyCentralModel
OtherService
MyClass1
MyClass2
MyClass3

Easier tests



:fritzcited

Closing thoughts

Dependencies

- Lots of dependencies make tests difficult to write
- There are ways of dealing with dependencies more easily
- Can write code with easier dependencies

Fewer dependencies

- Less I don't wanna
- More safety and flexibility

Teammates

TDD?

Useful reading

Testing Rails Applications
https://guides.rubyonrails.org/testing.html

ThoughtBot TDD Ebook
https://books.thoughtbot.com/assets/testing-rails.pdf

Sandi Metz - the magic tricks of testing
https://www.youtube.com/watch?v=URSWYvyc42M

Sandi Metz - POODR
https://www.poodr.com/

JB Rainsberger
https://blog.jbrains.ca/

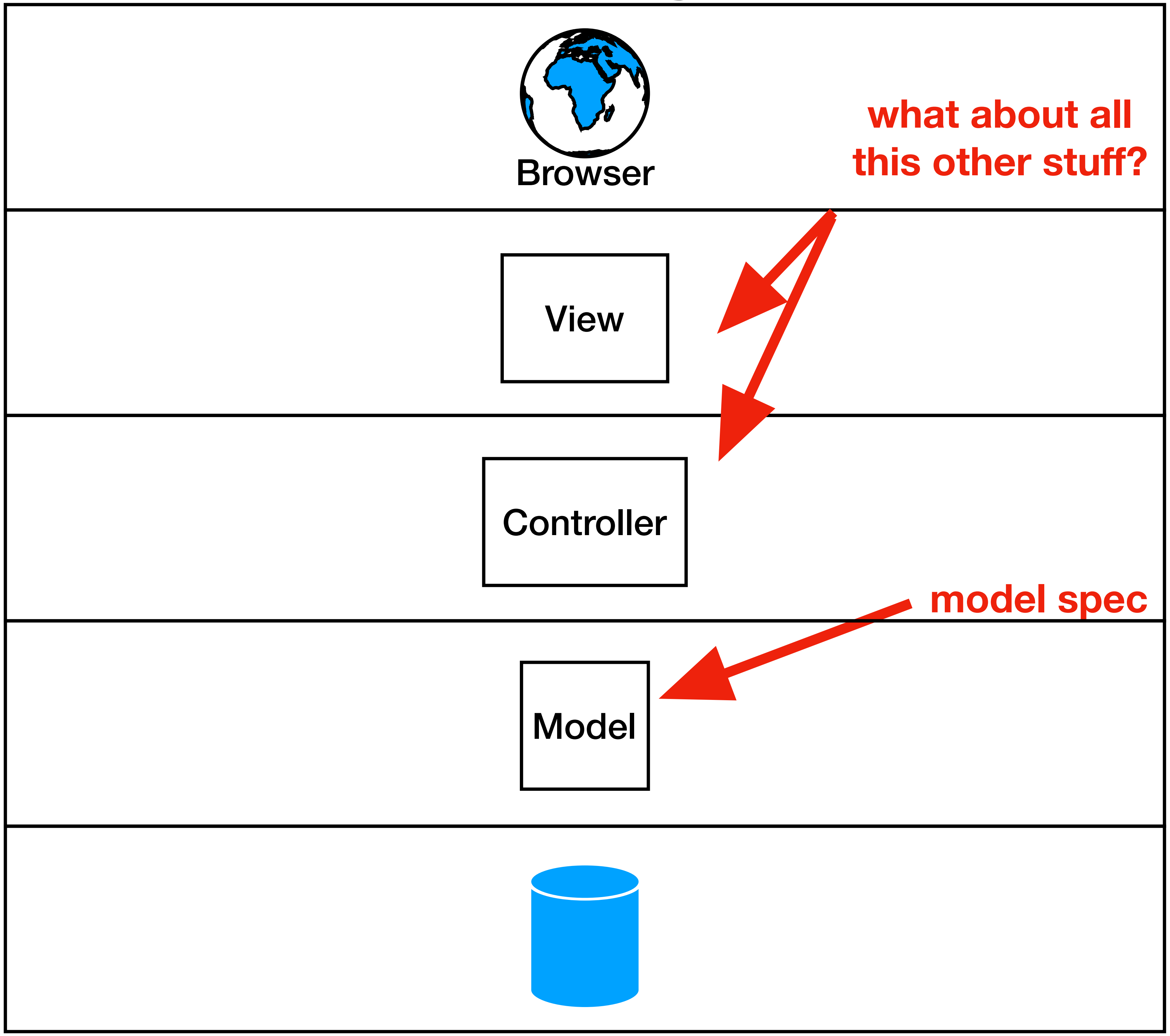
Martin Fowler Mocks aren't Stubs
https://martinfowler.com/articles/mocksArentStubs.html

Questions?

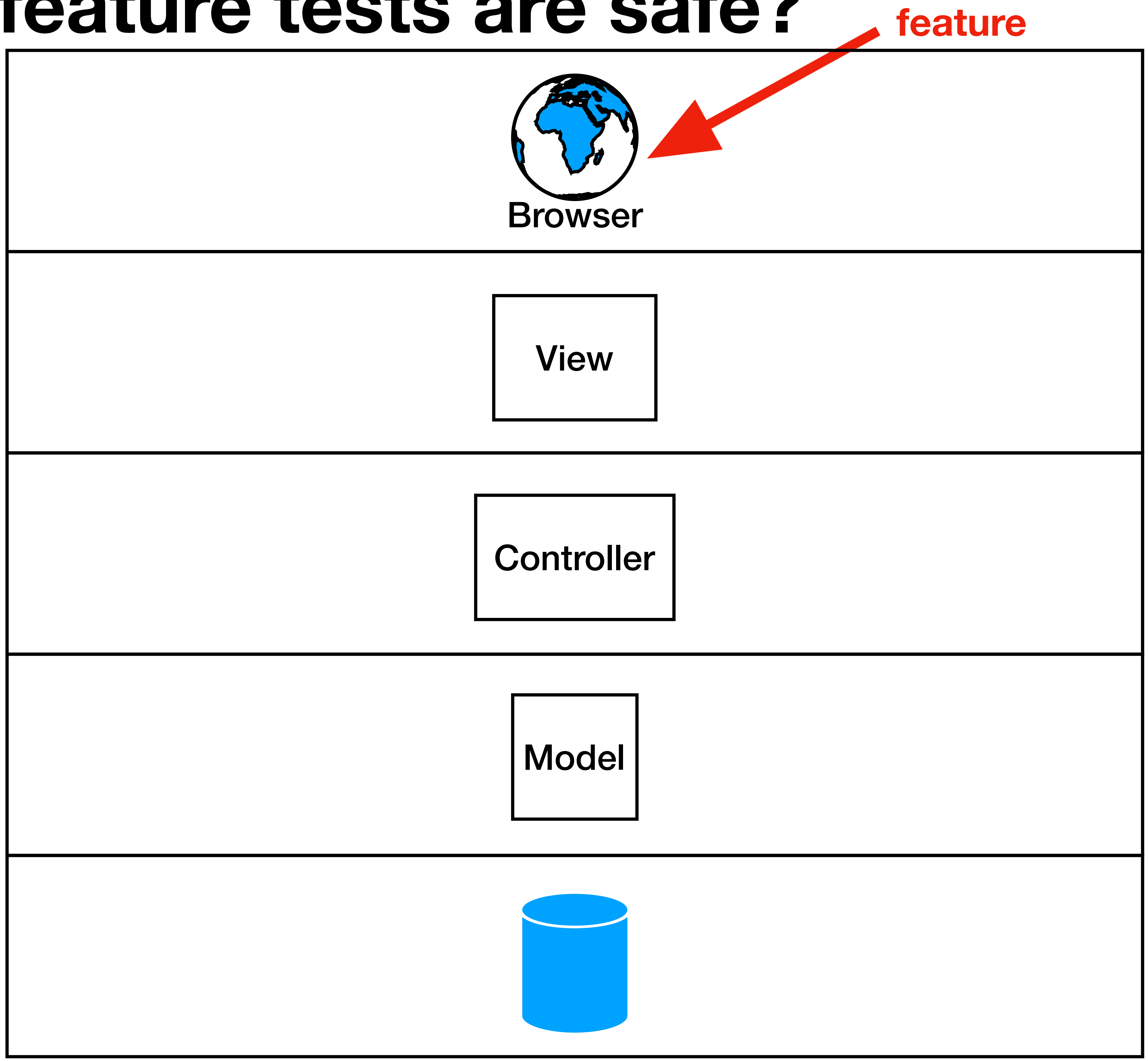
Thank you

Early Fritz: feature tests

Model tests are easy to write



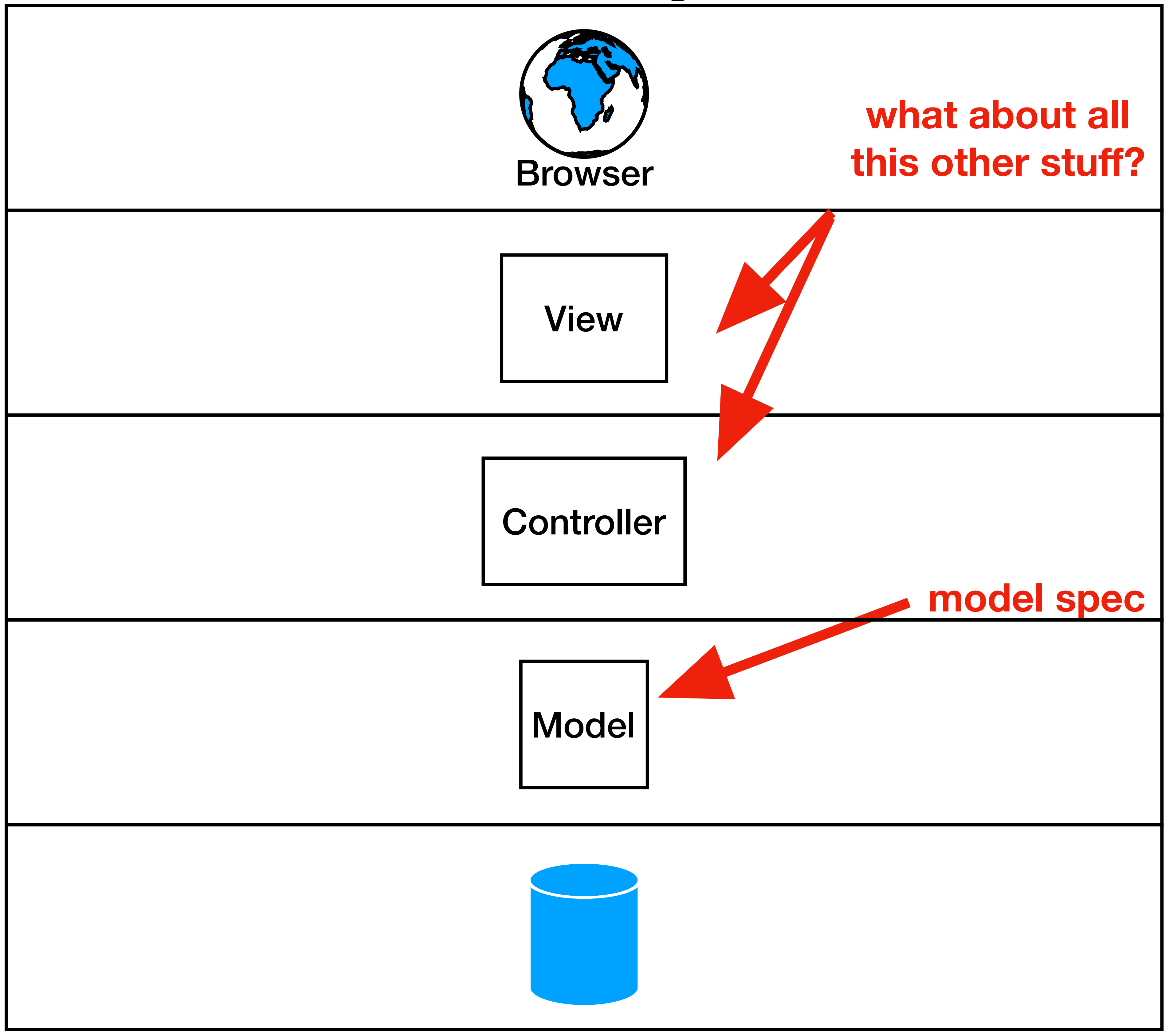
But feature tests are safe?



A few months later

30 minutes of feature tests
Still bad coverage
I don't wanna

Model tests are easy to write



Consistency in setup

```
describe MyClass do
  describe '#my_method' do
    context 'when there is an X' do
      before do
        @x = X.new
      end

      context 'and there is a Y' do
        before do
          @y = Y.new
        end

        context 'and there is a Z with a P and a Q' do
          it 'does something' do
             z = Z.new(p: P.new, Q: Q.new)
             MyClass.new(x, y, z).my_method
          end
        end
      end
    end
  end
end
```

this

and this

say something
about this