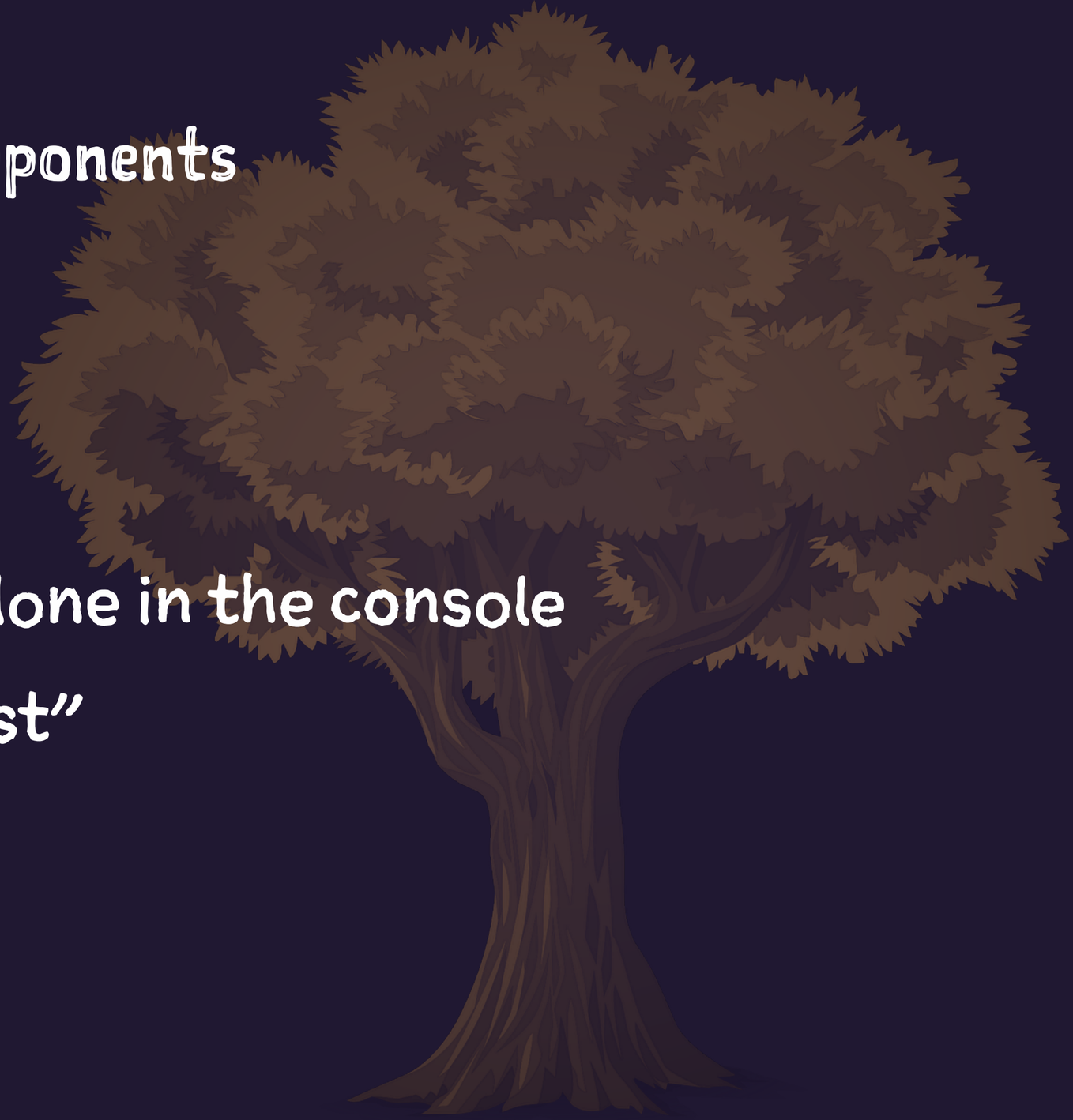
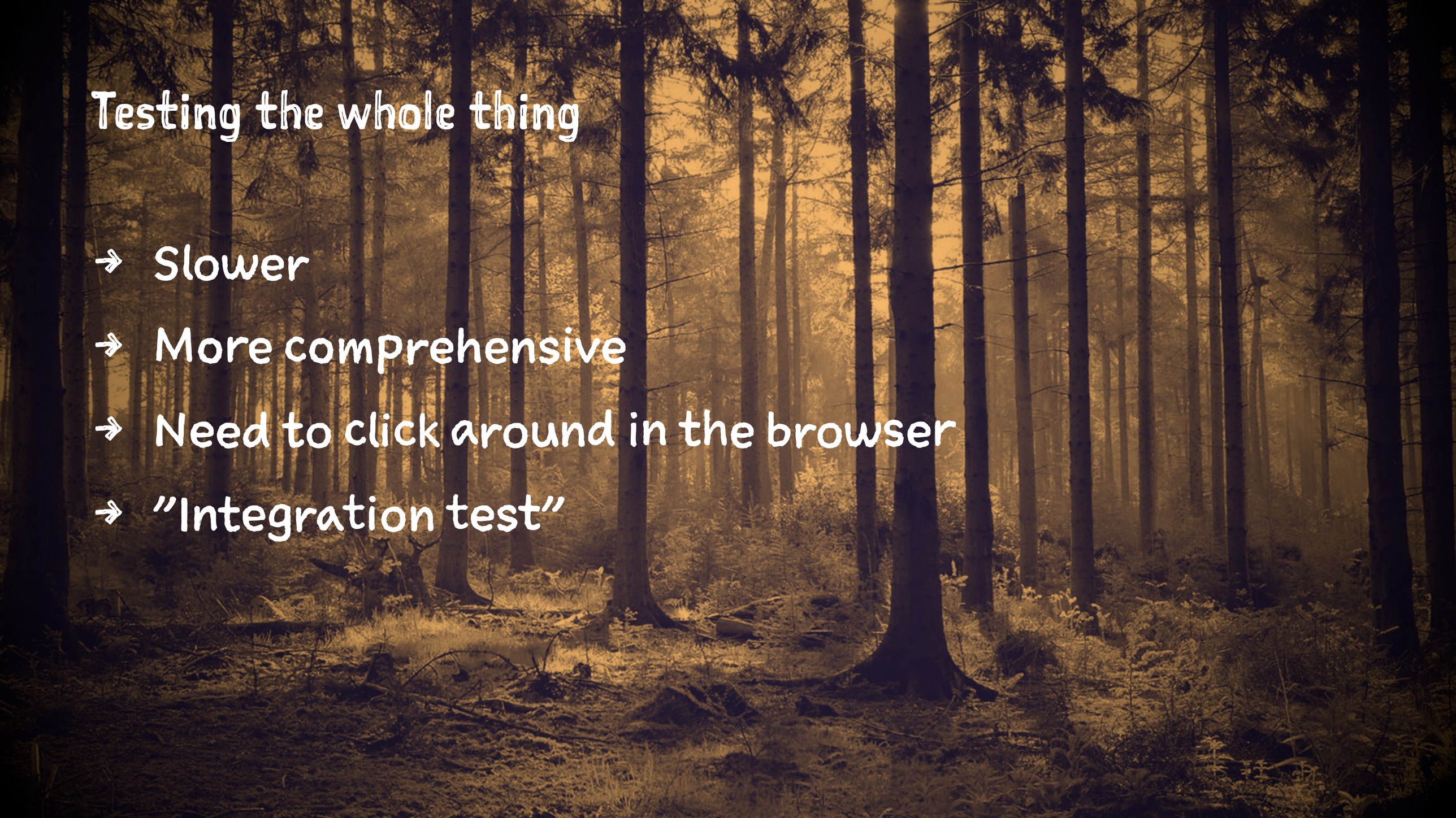
Why do we test?


Two approaches

Testing Components



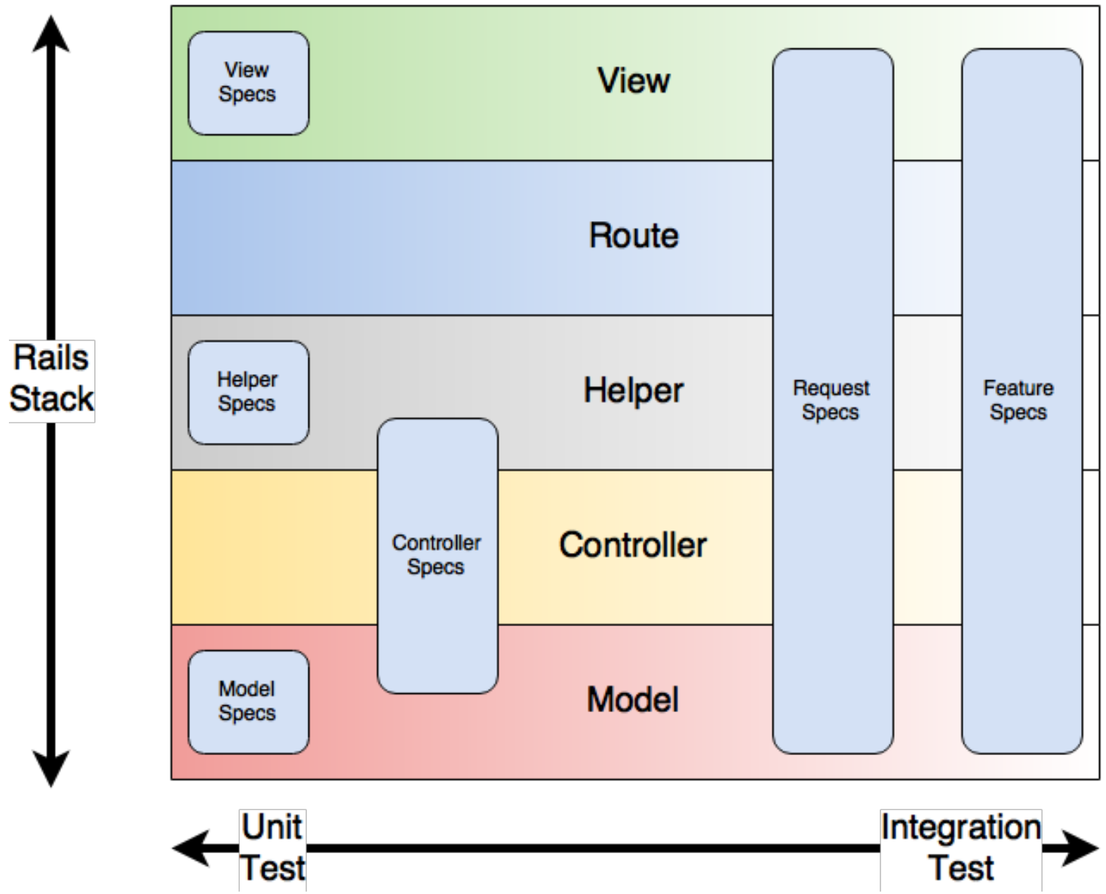
- Fast
- Simple
- Can be done in the console
- "Unit test"



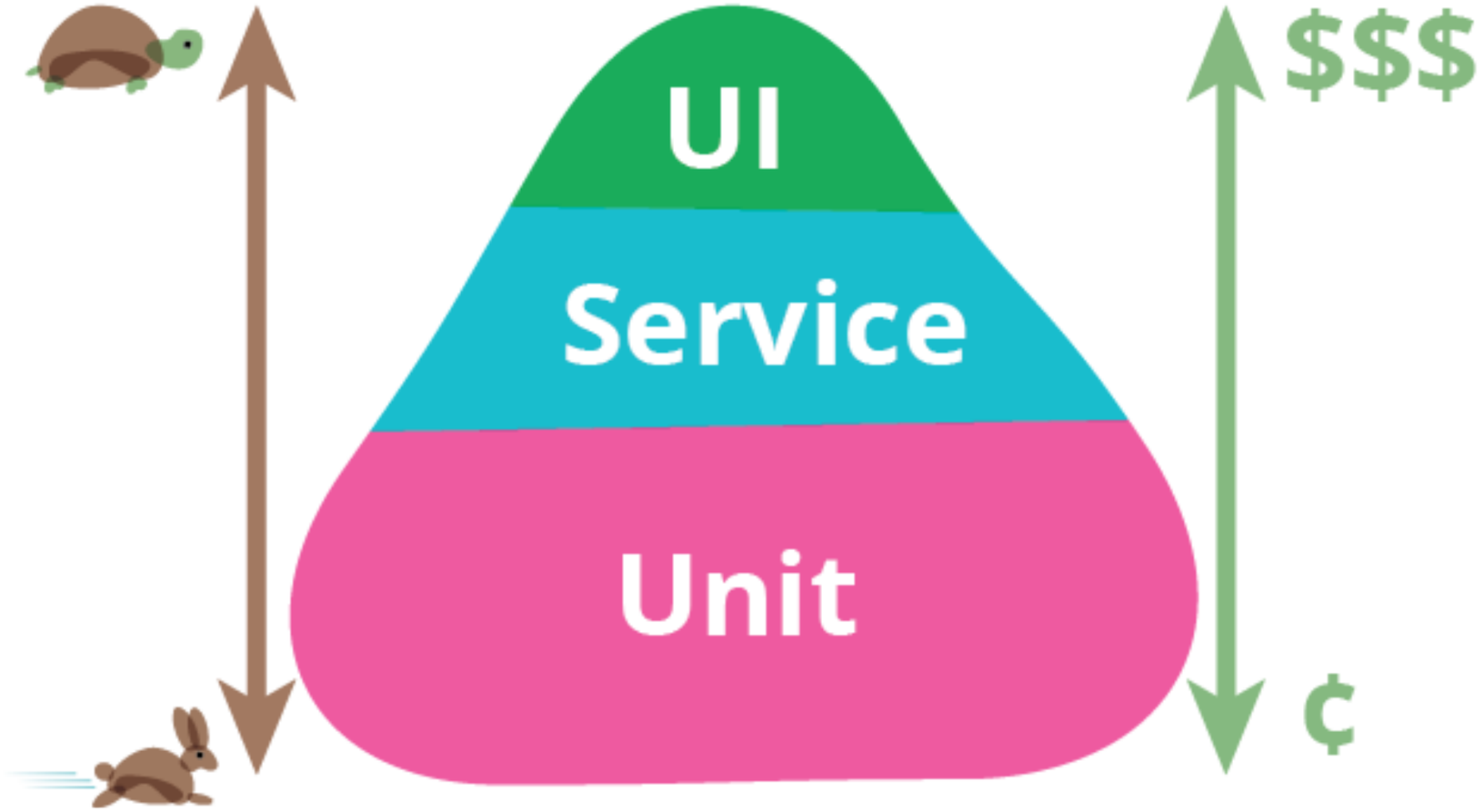
Testing the whole thing

- Slower
- More comprehensive
- Need to click around in the browser
- "Integration test"

Rails test types



Testing pyramid

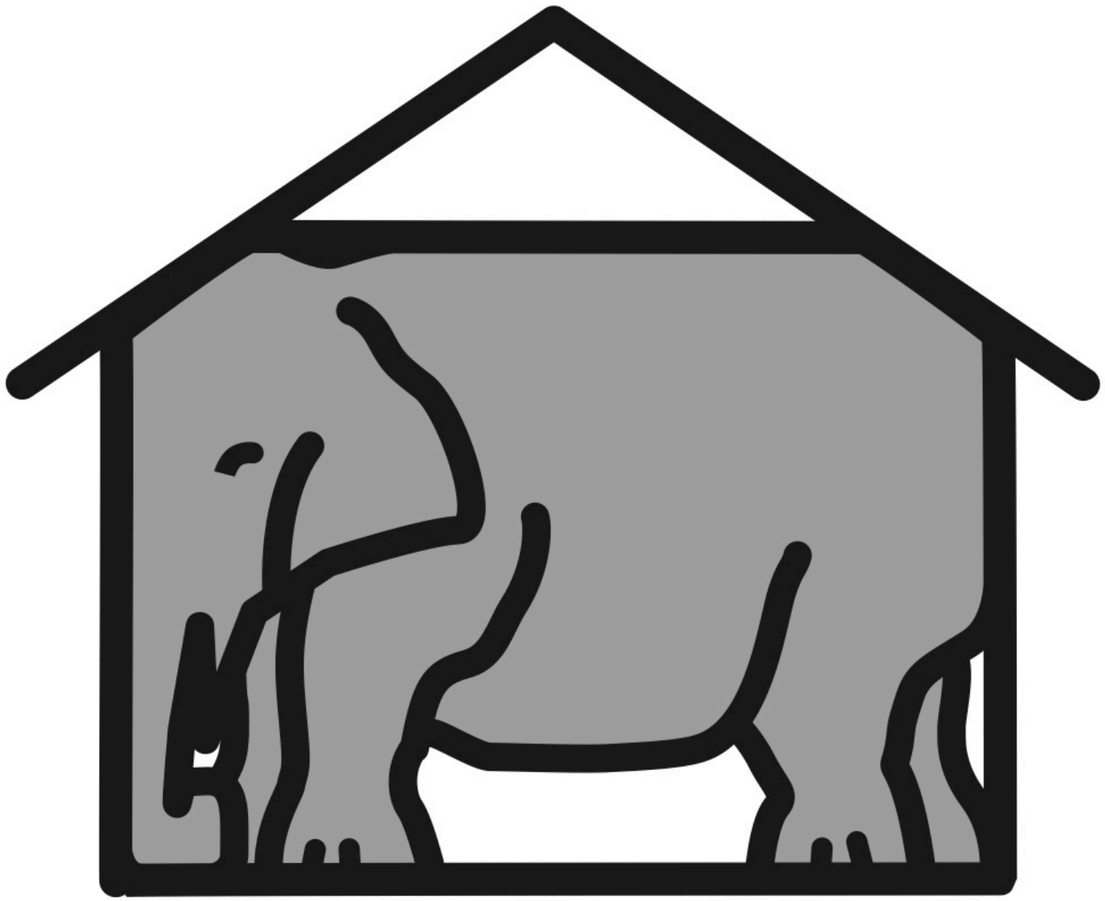


Testing in isolation


easy

No dependencies

```
it "does something" do
  formatter = Formatter.new

  formatted_text = formatter.format("foo")

  expect(formatted_text).to eq "FOO"
end
```

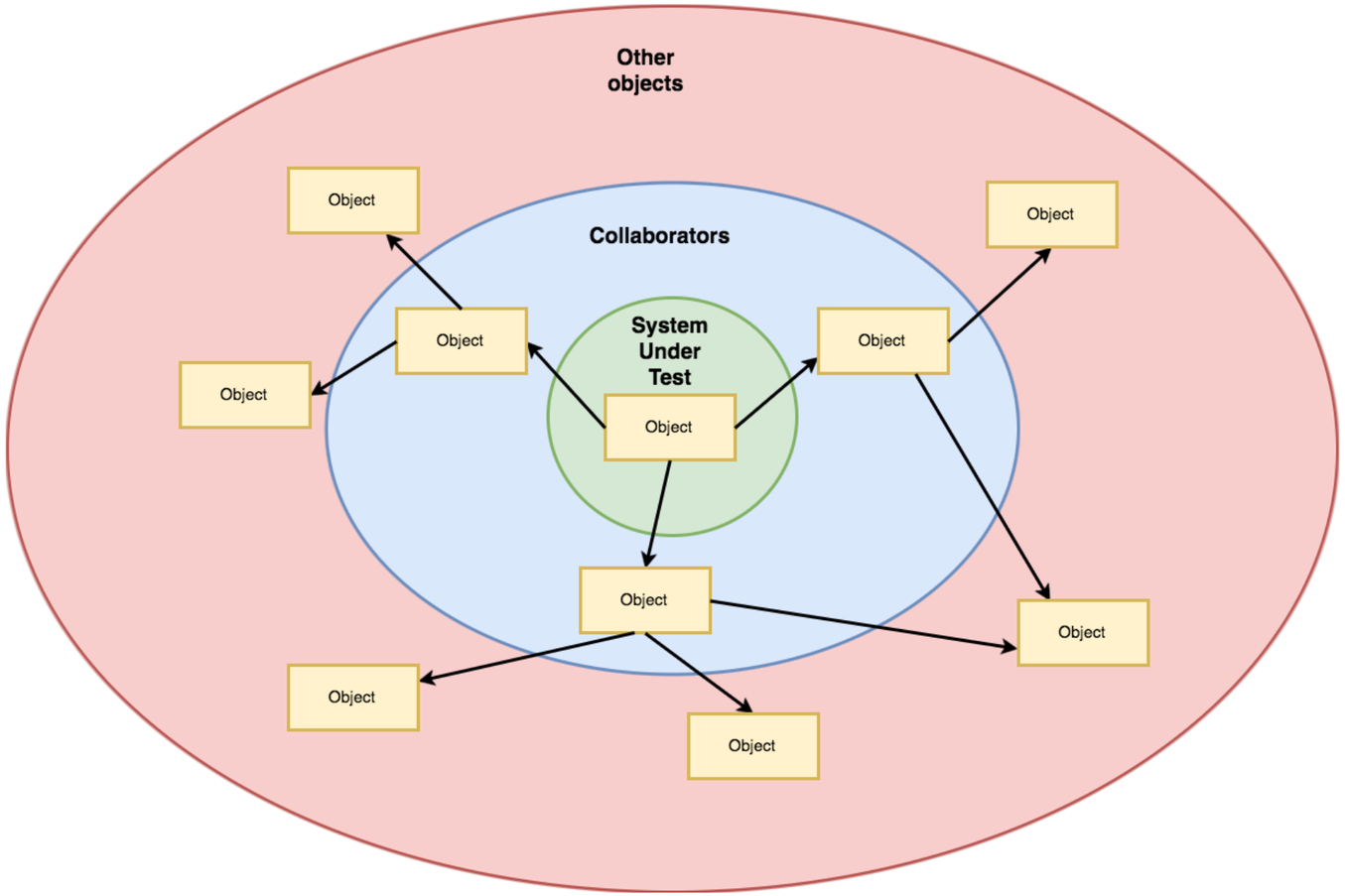


ELEPHANT
IN THE ROOM


Public-Private Partnerships

Test the what, not the how

Object graphs



Introducing a collaborator

```
class Formatter
  def format_name(user)
    "#{user.first_name} #{user.last_name}"
  end
end
```


duck typing

Duck typing

```
class User < ActiveRecord::Base
 validates :first_name, presence: true
 validates :last_name, presence: true
end
```

Duck typing

```
class Guest
  def first_name
    "First"
  end

  def last_name
    "Last"
  end
end
```

Duck typing

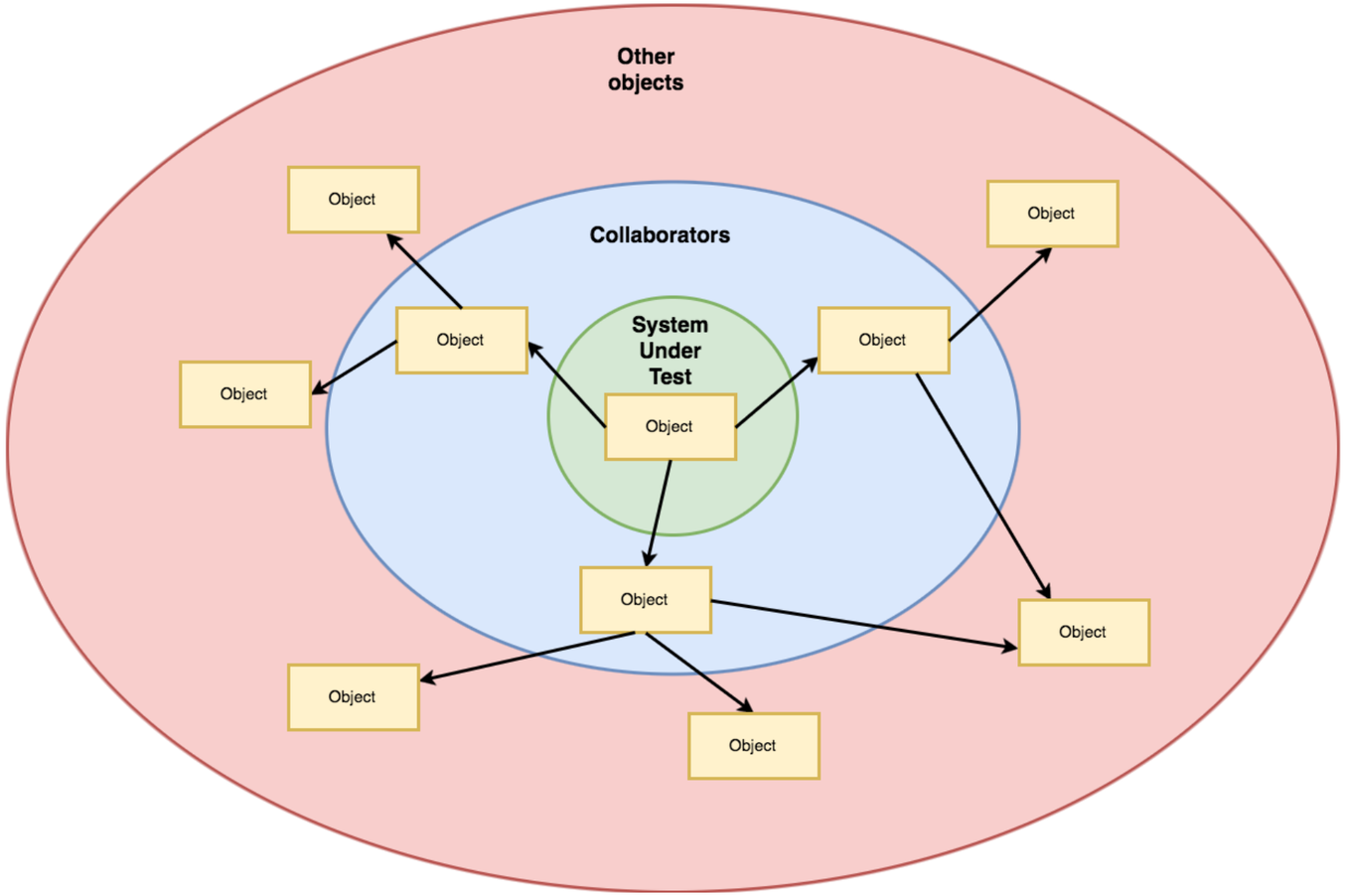
```
Duck = Struct.new(:first_name, :last_name)
```

Test doubles

```
it "formats a user's name" do
  formatter = Formatter.new
  user = double(first_name: "Joël", last_name: "Quenneville")

  formatted_text = formatter.format_name(user)

  expect(formatted_text).to eq "Joël Quenneville"
end
```


Dependency injection

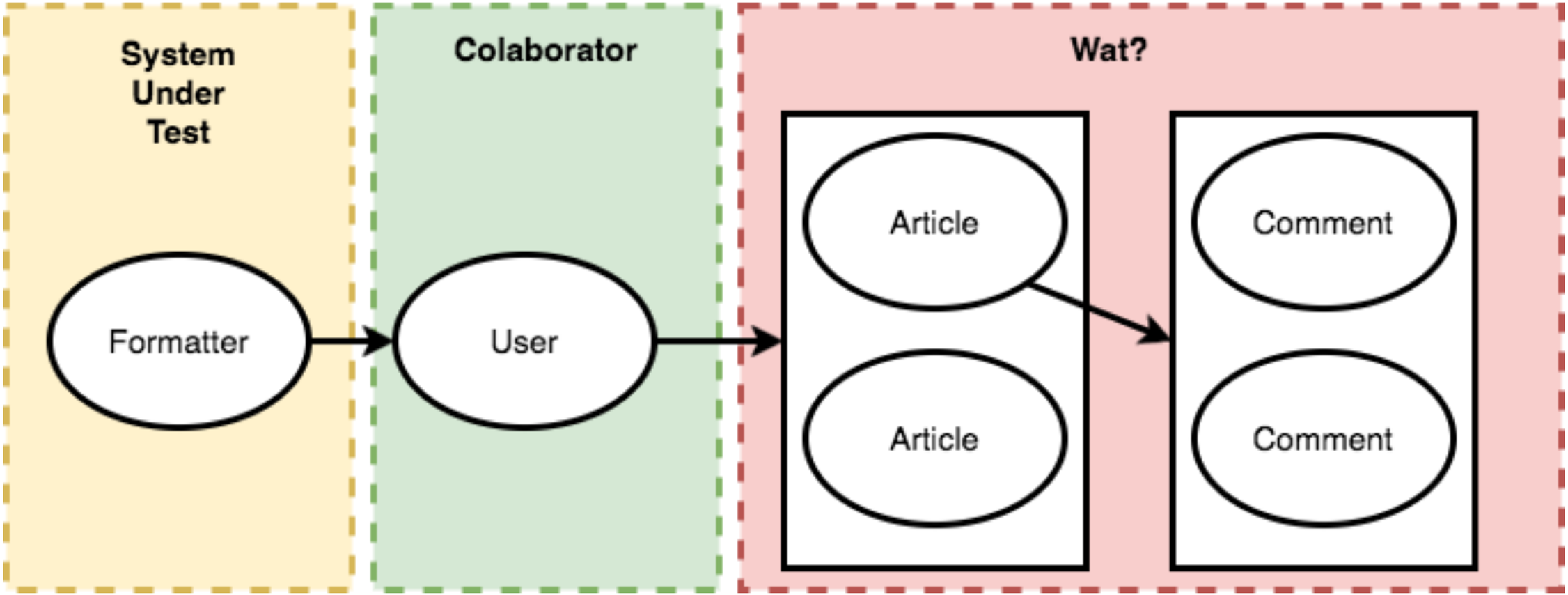
Remember this object graph?



Collaborators of collaborators

```
class Formatter
  def comment_blurb(user)
    "Most recent comment:\n #{user.articles.comments.most_recent}"
  end
end
```

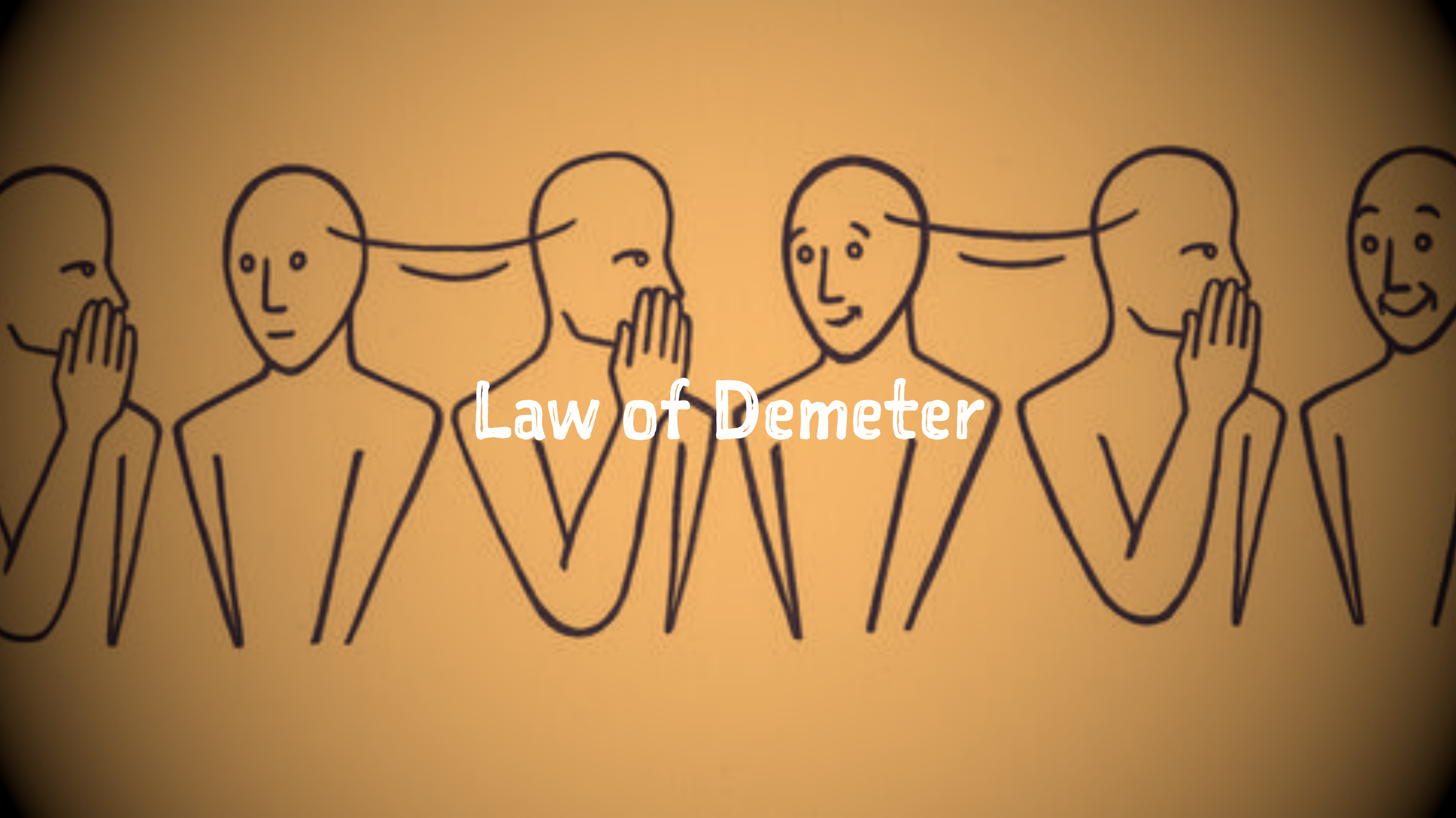
Graphical representation



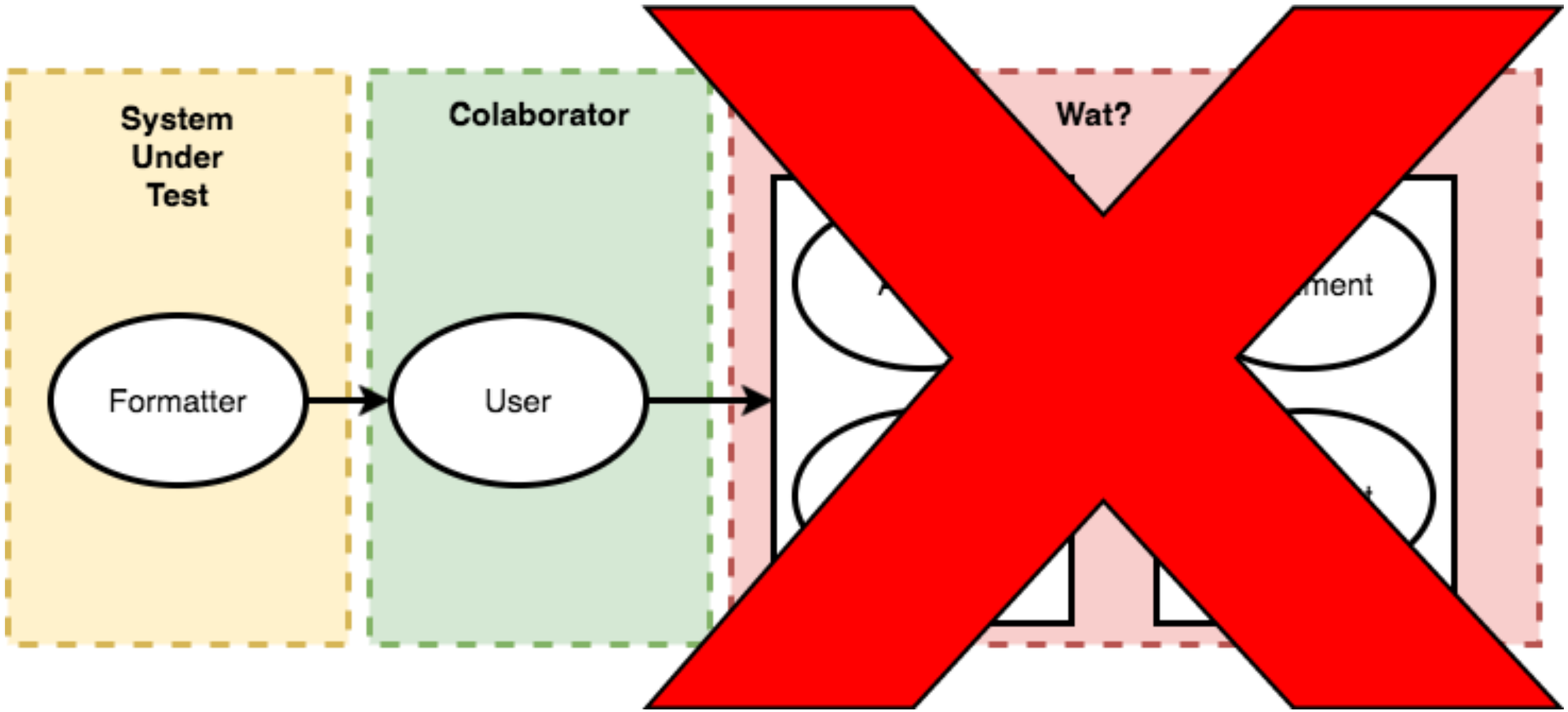
Test

```
it "shows a blurb with most recent comment" do
  formatter = Formatter.new
  comment = double(body: "great post!")
  article = double(comments: [comment])
  user = double(articles: [article])

  formatted_text = formatter.comment_blurb(user)

  expect(formatted_text).to eq "Most recent comment:\n - great post!"
end
```


Test pain often points to architecture smells


Law of Demeter

Desired result



Simpler Test...

```
it "formats a user's name with their latest comment" do
  formatter = Formatter.new
  user = double(most_recent_comment: "great_post")

  formatted_text = formatter.comment_blurb(user)

  expect(formatted_text).to eq "Most recent comment: \n - great post!"
end
```

... leads to simpler code

```
class Formatter
  def comment_blurb(user)
    "Most recent comment:\n #{user.most_recent_comment}"
  end
end
```


Something harder

There are two types of methods...

```ruby
# query
def format(user)
  "#{user.first_name} #{user.last_name}"
end

# command
def save(data)
  HTTParty.post(ENDPOINT, data)
end
```

Stubbing

```
it "posts to the API" do
  allow(HTTParty).to recieve(:post)
  data = double
  user = User.new

  user.save(data)

  expect(HTTParty).to have_recieved(:post).with(User::ENDPOINT, data)
end
```

Temptation...

```
class Sasquatch
  def to_s
    "#{full_name} - Sasquatch"
  end
end
```


JACK LINK'S
BEEF JERKY
MESSIN' WITH
SASQUATCH
JACK LINK'S
ORIGINAL
BEEF
JERKY

stubbing the system under test

```
it "renders a textual representation of the user" do
  sasquatch = Sasquatch.new
  allow(sasquatch).to recieve(:full_name).and_return("First Last")

  expect(sasquatch.to_s).to eq "First Last - Sasquatch"
end
```

The know it all

```
class KnowItAll
  def do_a_thing
    # a lot of stuff
  end

  def do_another_thing
    do_a_thing
    # do some more stuff
  end
end
```

Extract Class

```
class AnotherThing
  def do_another_thing(know_it_all)
    know_it_all.do_a_thing
    # do some more stuff
  end
end
```

Single Responsibility

Prefer composition to inheritance

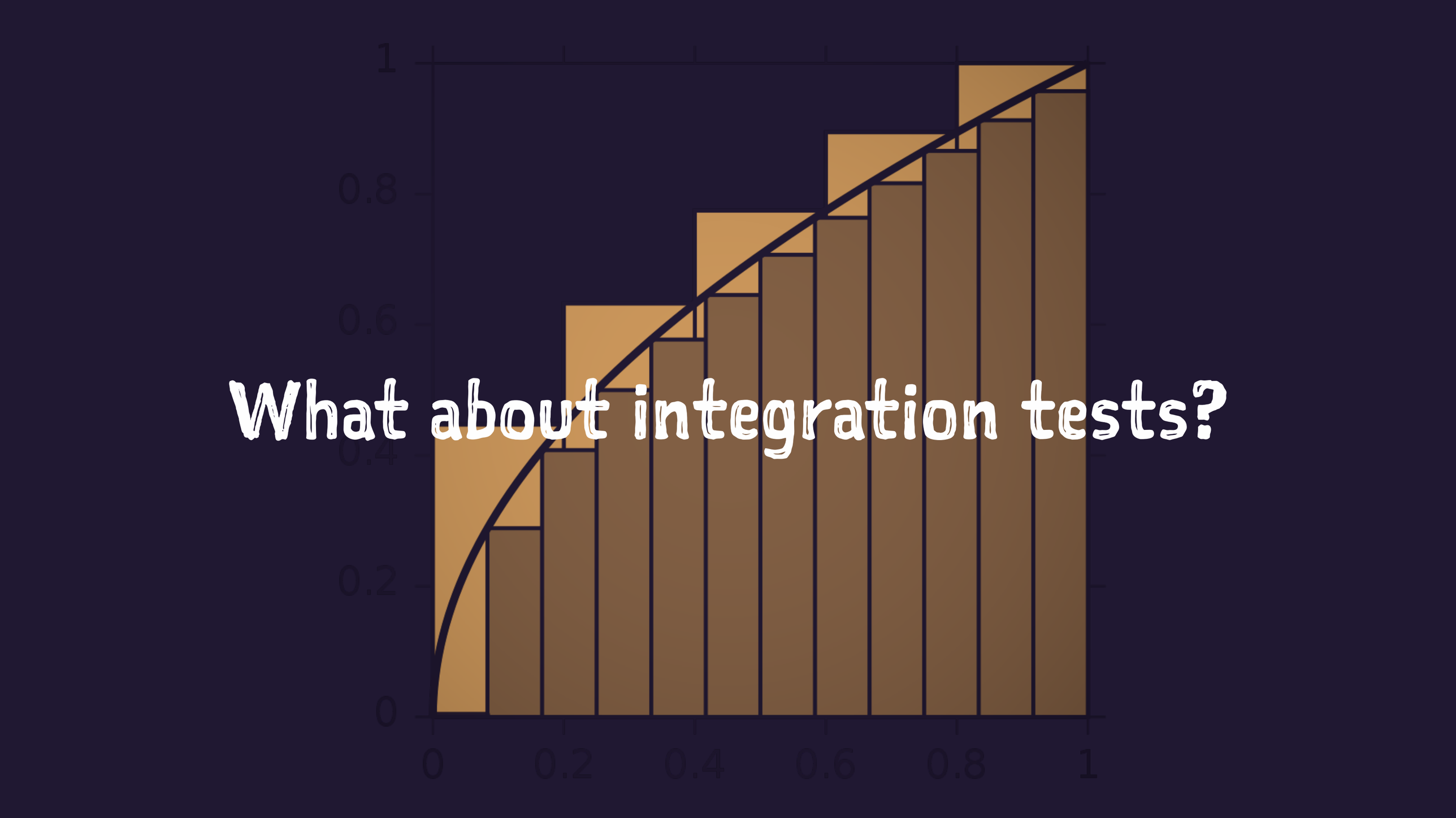
Command/Query separation (not!)

```
class User
  def self.create(data)
    response = HTTParty.post(data)
    if response.success?
      new(response.body)
    else
      false
    end
  end
end
```

Working with your framework

Principles of testing

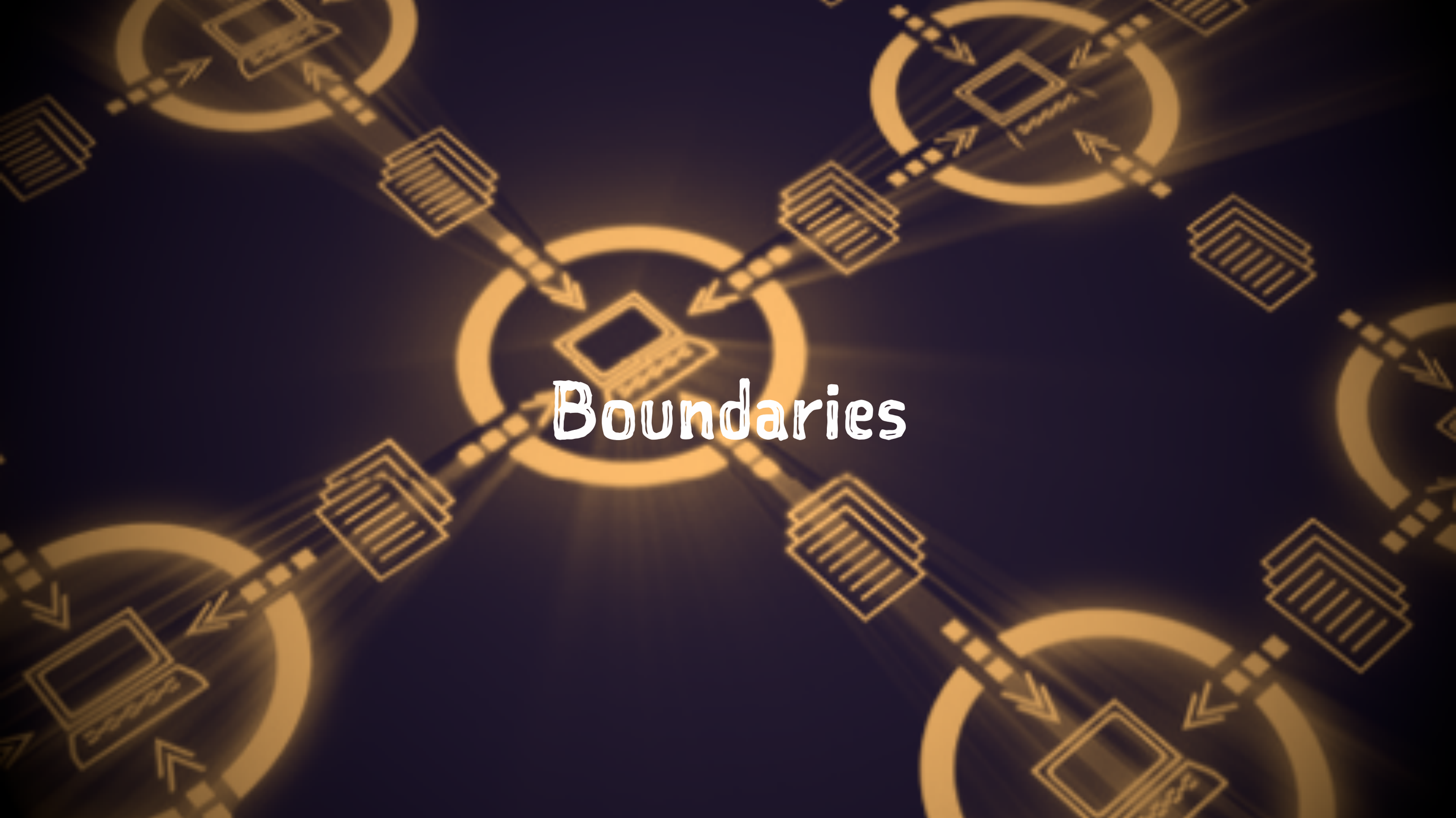
- Isolate your subject
- Don't mess with the subject
- If you feel pain, maybe rethink your implementation



What about integration tests?

Stubbing the system under test


JACK LINK'S
BEEF JERKY
MESSIN' WITH
SASQUATCH
JACK LINK'S
ORIGINAL
BEEF
JERKY


Boundaries

Principles of testing

- Isolate your subject
- Don't mess with the subject
- If you feel pain, maybe rethink your implementation

Questions?

Slides:

http://github.com/JoelQ/testing-in-isolation