AN INTRODUCTION TO MOCKING IN RUBY

Brian Guthrie
bguthrie@thoughtworks.com
Twitter: bguthrie

ThoughtWorks®

What is mocking?

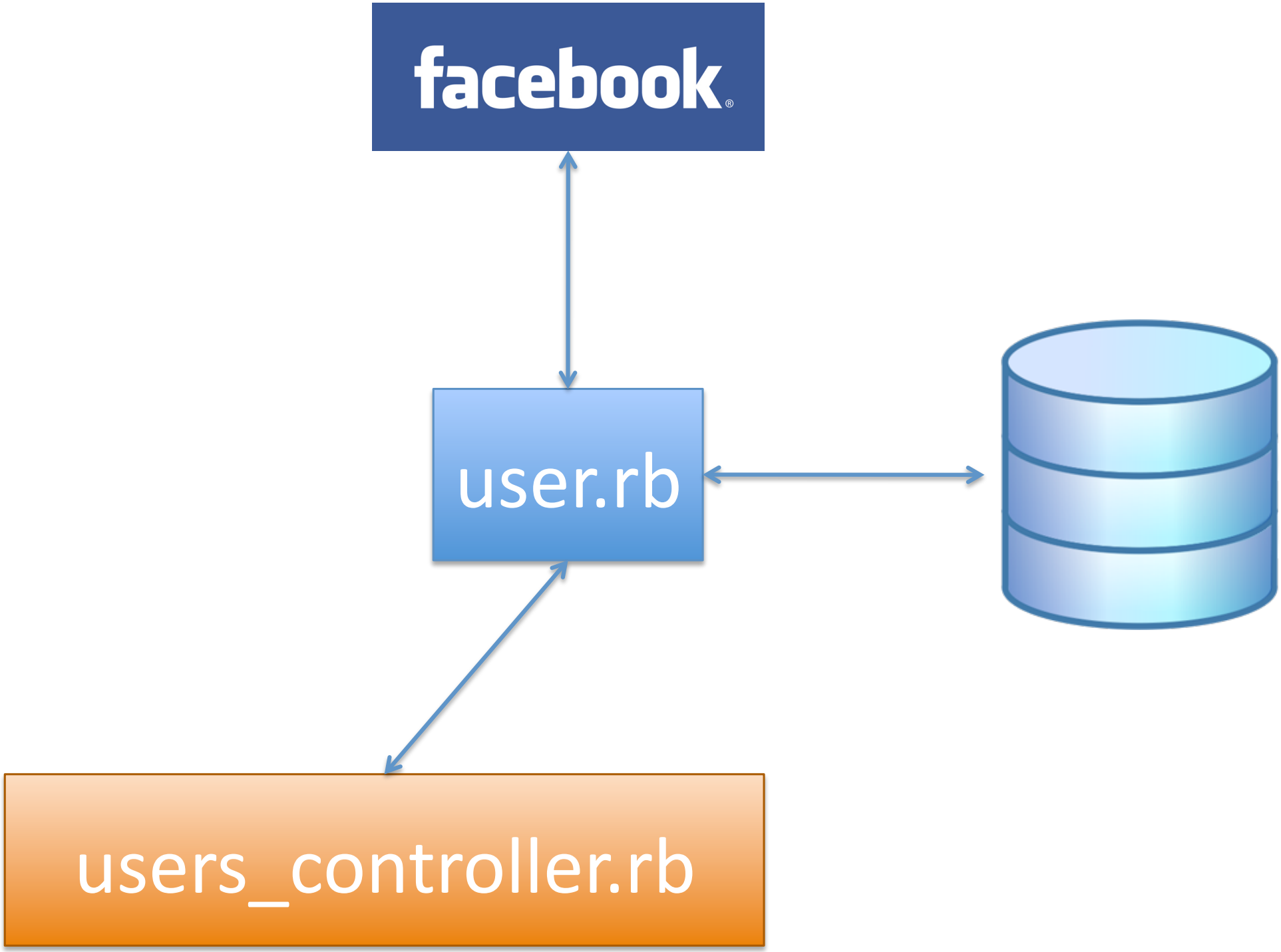
heckle.rb

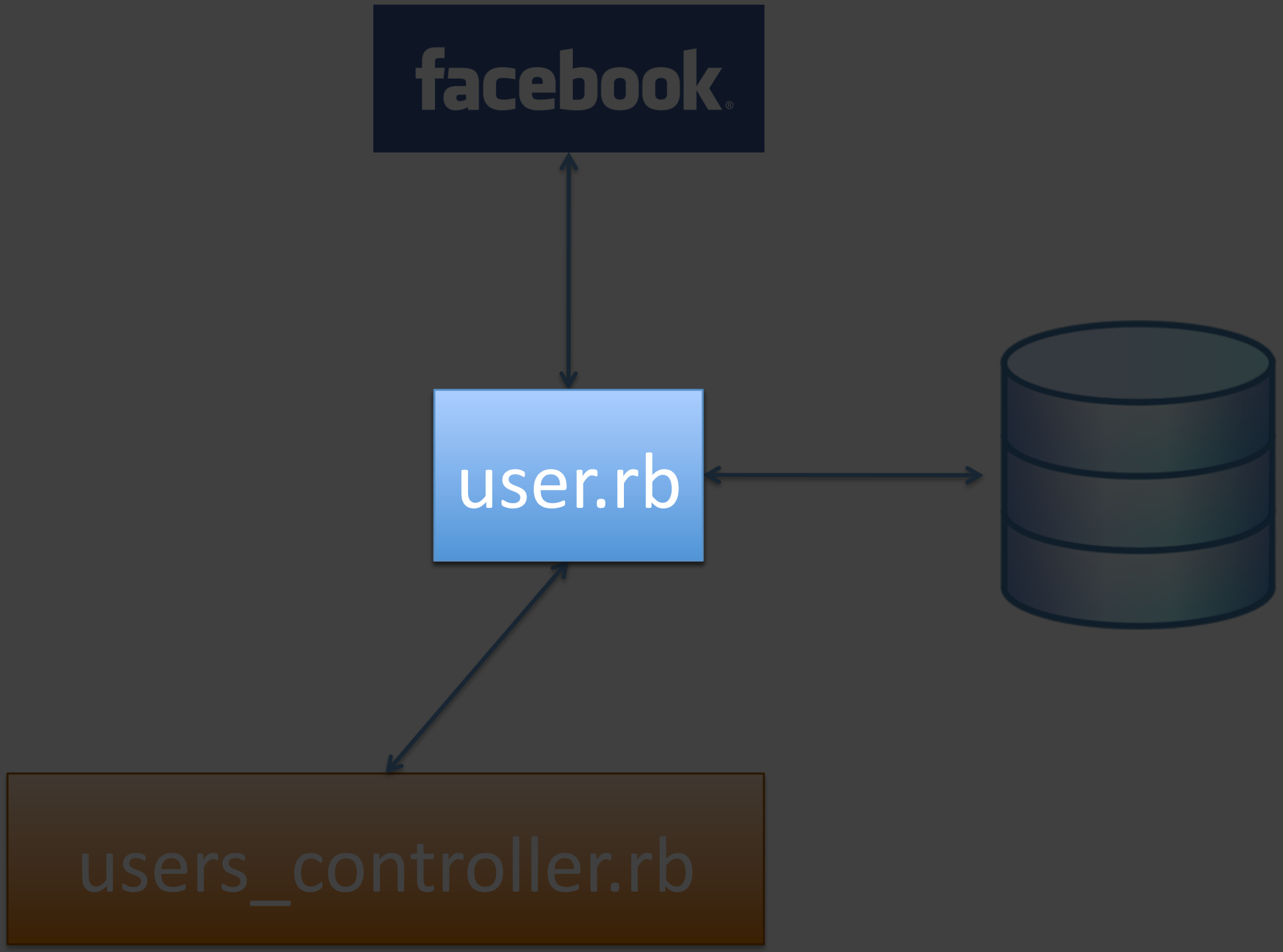
Mocks and stubs **pretend to be something else** for the duration of a test.


Worth1000.com

```ruby
class User < ActiveRecord::Base
  has_many :addresses
  belongs_to :company
  validate :some_extremely_complicated_thingamajig

  class << self
    def authenticate(username, password)
      Facebook.authenticate ...
    end
  end
end
```


facebook
user.rb
users_controller.rb


facebook
user.rb
users_controller.rb

Mocks replace existing methods and fail your test unless those methods are called.

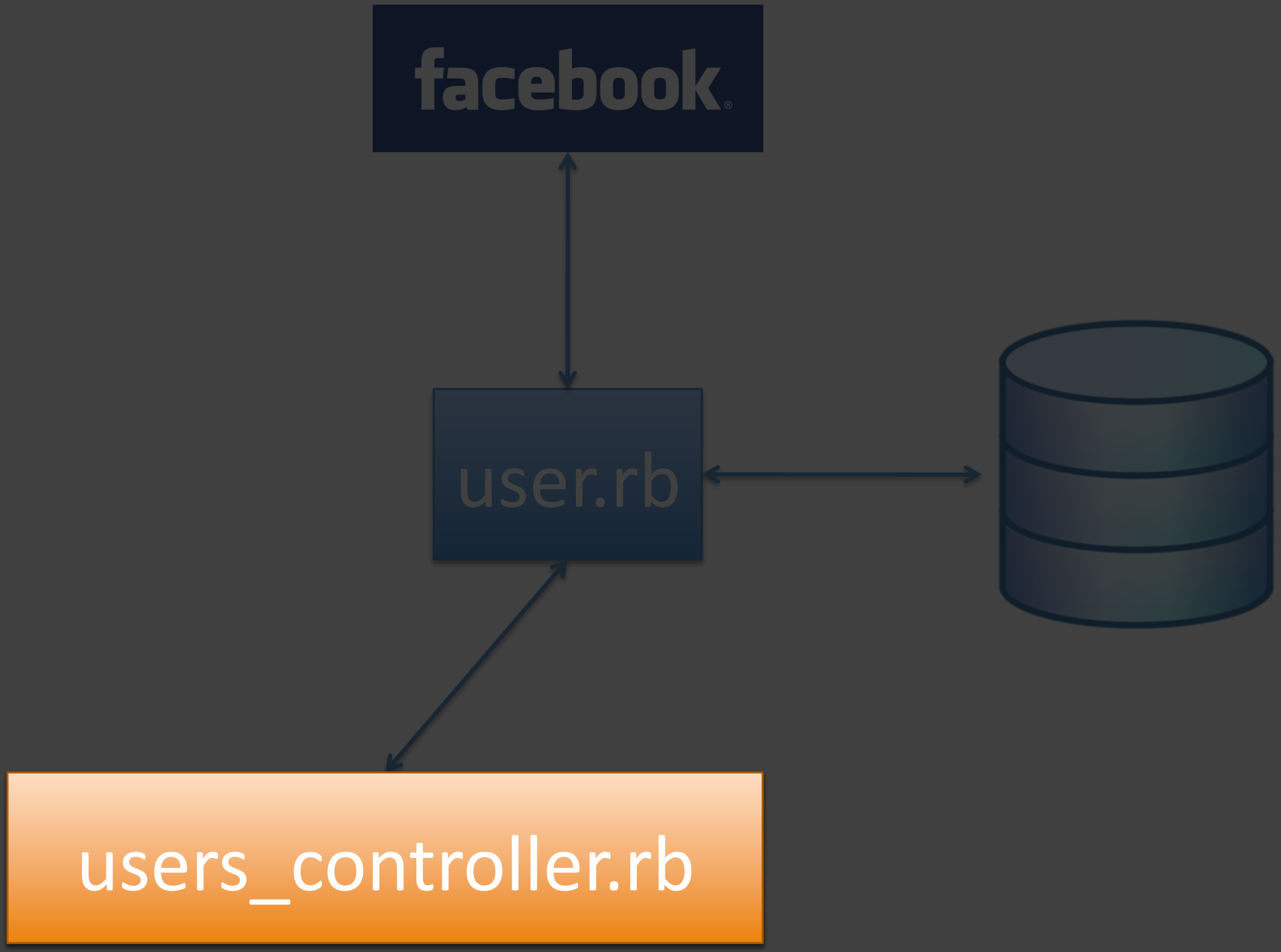
```ruby
describe User do

  it "should authenticate with the Facebook module" do
    Facebook::Connect.expects(:authenticate).with(
      :username => "bguthrie",
      :password => "secret"
    ).returns(true)

    User.authenticate("bguthrie", "secret").should be_true
  end

end
```

Stubs simply replace methods.


facebook
user.rb
users_controller.rb

```ruby
describe UsersController do

  it "should render successfully if the user is authenticated" do
    Facebook::Connect.stubs(:authenticate).returns true
    post :login, :username => "bguthrie", :password => "secret"
    response.should be_success
  end

end
```

Use **mocks** to test **interaction with dependencies.**

Use **stubs** to **ignore dependencies.**

Don’t use either if you don’t have to.

Controversy alert!

```ruby
def setup
  @gem = Rails::GemDependency.new "hpricot"
end

def test_gem_adds_load_paths
  @gem.expects(:gem).with(Gem::Dependency.new(@gem.name, nil))
  @gem.add_load_paths
end
```

ails/railties/test/gem_dependency_test.rb

```ruby
def test_gem_loading
  @gem.expects(:gem).with(Gem::Dependency.new(@gem.name, nil))
  @gem.expects(:require).with(@gem.name)
  @gem.add_load_paths
  @gem.load
end
```

ails/railties/test/gem_dependency_test.rb

```ruby
describe "ResourceFull::Dispatch", :type => :controller do
  describe "POST create" do
    controller_name "resource_full_mocks"

    it "assigns @resource_full_mock based on the default creator" do
      ResourceFullMock.stubs(:create).returns stub(
        :valid? => true,
        :id => :mock
      )

      post :create, :format => 'html'
      assigns(:resource_full_mock).id.should == :mock
    end
  end
end
```

esource_full/spec/resource_full/dispatch_spec.rb

```ruby
describe "ResourceFull::Dispatch", :type => :controller do
  before(:each) do
    ResourceFullMock.stubs(:find).returns stub(:id => 1)
  end

  describe "based on request format" do
    it "dispatches to index_xml render method if xml is requested" do
      controller.expects(:index_xml)
      get :index, :format => 'xml'
    end
  end
end
```

esource_full/spec/resource_full/dispatch_spec.rb

Should this use mocks?

```
describe WidgetImporter do
  describe "execute" do
    before :each do
      Widget.should_receive(:transaction).and_yield
      @importer = WidgetImporter.new
    end

    it "should create a record given a unit measurement" do
      Widget.should_receive(:create).with(
        :name => 'Fancy widget',
        :size => '2'
      )

      @importer.execute('Fancy widget,2')
    end
  end
end
```

One possible alternative

```
describe WidgetImporter do
  describe "execute" do
    before :each do
      @importer = WidgetImporter.new
    end

    it "should create a record given a unit measurement" do
      @importer.execute('Fancy widget,2')
      Widget.should have(1).record
      Widget.find(:first).name.should == 'Fancy widget'
    end
  end
end
```

Over-mocking leads to **brittle tests.**

```ruby
module Product::Lifecycle::Retirable
  def retire!
    update_actual_retired_time
    unless sell!
      notify_buyer_of :retired
      auto_relist!
    end
  end
end
```

```
test "retire automatically relists a duplicate with same
    duration starting on the end time of retired product" do
  end_time = Time.now
  start_time = end_time - 3.days
  product = retired_product
  new_product = stub_everything(:disclosed_problems => [])
  Product.expects(:new).yields(new_product).returns(new_product)
  new_product.expects(:start_time=).with(product.end_time)
  new_product.expects(:update_duration).with(product.live_duration_time)
  new_product.expects(:status=).with(Status::Live)

  product.retire!
end
```

```
test "retire automatically relists a duplicate with same
    duration starting on the end time of retired product" do
  end_time = Time.now
  start_time = end_time - 3.days
  product = retired_product
  new_product = stub_everything(:disclosed_problems => [])
  Product.expects(:new).yields(new_product).returns(new_product)
  new_product.expects(:start_time=).with(product.end_time)
  new_product.expects(:update_duration).with(product.live_duration_time)
  new_product.expects(:status=).with(Status::Live)

  product.retire!
end
```

Over-stubbing leads to **weak tests.**

```ruby
describe SomeClass do
  it "has been refactored a bunch of times" do
    SomeDependency.stubs(:a_method)
    AnObsoleteDependency.stubs(:another_method)
    YetAnotherDependency.stubs(:a_method_that_no_longer_exists)
    SomeClass.new.foo!
  end
end
```

```
Mocha::Configuration.prevent :stubbing_non_existent_methods
```

So why mock?

Databases are still **slower than the alternative**.

mock_model: build mocks that look like ActiveRecord objects

unit_record: explode if your test tries to touch the database

Remote systems **aren’t always accessible or performant**.

Writing them encourages **better dependency management.**

Other tips

```
mock_user = mock("a user",
   :first_name => "Guybrush",
   :last_name => "Threepwood"
)
```

```ruby
User.any_instance.expects(:authenticate)
```

```
user.expects(:authenticate).never

user.expects(:authenticate).once

user.expects(:authenticate).times(6)
```

Questions?

Brian Guthrie
bguthrie@thoughtworks.com

ThoughtWorks®