
Thanks for making it!


The magic hook

New Event

Name

Location

Date

2019 February 1

Crew arrives at

2019 February 1 — 15 : 58

Performers arrive at

2019 February 1 — 15 : 58

Open at

2019 February 1 — 15 : 58

Starts at

2019 February 1 — 15 : 58

Ends at

2019 February 1 — 15 : 58

Create Event

Back

Event Form

New Event

Name

Location

Date

2019 February 1

Crew arrives at

2019 February 1 — 15 : 58

Performers arrive at

2019 February 1 — 15 : 58

Open at

2019 February 1 — 15 : 58

Starts at

2019 February 1 — 15 : 58

Ends at

2019 February 1 — 15 : 58

Create Event

Back

Repetitive

New Event

Name

Location

Date

2019 February 1

Crew arrives at

15 : 58

Performers arrive at

15 : 58

Open at

15 : 58

Starts at

15 : 58

Ends at

15 : 58

Create Event

Back

Much better!

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

Datetime fields

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

```ruby
event =
  Event.new(
    name: "Ruby On Ice",
    location: "Tegernsee",
    date: "24.02.2019",
    crew_arrives_at: "6:45",
    performers_arrive_at: "9:30",
    open_at: "9:30",
    starts_at: "10:00",
    ends_at: "16:00"
  )

event.valid?
```

It works!

```
#<Event:
 name: "Ruby On Ice",
 location: "Tegernsee",
 date: Sun, 24 Feb 2019,
 crew_arrives_at: Sun, 24 Feb 2019 6:45:00,
 performers_arrive_at: Sun, 24 Feb 2019 9:00:00,
 open_at: Sun, 24 Feb 2019 9:00:00,
 starts_at: Sun, 24 Feb 2019 9:30:00,
 ends_at: Sun, 24 Feb 2019 20:00:00>
```

```ruby
let(:event) do
  build :event,
        ends_at: Time.zone.local(2042, 1, 1, 15, 45)
end
```

```ruby
let(:event) do
  build :event,
        ends_at: Time.zone.local(2042, 1, 1, 15, 45)
end
```

Have fun debugging!

```ruby
let(:event) do
  build :event,
        ends_at: Time.zone.local(2042, 1, 1, 15, 45)
end

it "works" do
  p event.ends_at # Wed, 01 Jan 2042 15:45:00
  event.save!
  p event.ends_at # Sun, 24 Feb 2019 15:45:00
end
```

Have fun debugging!

```
let(:event) do
  create :event,
         ends_at: Time.zone.local(2042, 1, 1, 15, 45)
end

it "retrieves the right events" do
  query = FutureEvents.new
  expect(query.call(23.years)).to include(event)
end
```

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

Smell

```ruby
class Event < ApplicationRecord
  before_validation :set_datetimes_to_date

  def set_datetimes_to_date
    base_date = date.to_datetime

    DATE_TIME_FIELDS.each do |time_attribute|
      original = public_send(time_attribute)
      if original
        adjusted_time =
          base_date.change hour: original.hour,
                           min:  original.min
        self.public_send("#{time_attribute}=", adjusted_time)
      end
    end
  end
end
```

Why does the model clean up

Validations

gitlab/user

```
validate :unique_email, if: :email_changed?
validate :owns_notification_email,
         if: :notification_email_changed?
validate :owns_public_email, if: :public_email_changed?
validate :owns_commit_email, if: :commit_email_changed?
```

gitlab/user

```
validate :unique_email, if: :email_changed?
validate :owns_notification_email,
         if: :notification_email_changed?
validate :owns_public_email, if: :public_email_changed?
validate :owns_commit_email, if: :commit_email_changed?
```


Why the effort?


Practice open?


Practice open?
No overlap?


Practice open?
No overlap?
Right skills?


Practice open?
No overlap?
Right skills?
Patient can be contacted?


Practice open?
No overlap?
Right skills?
Patient can be contacted?
Associated models


Practice open?
No overlap?
Right skills?
Patient can be contacted?
Associated models
...


Expensive Test Setup

gitlab/user

```
validate :unique_email, if: :email_changed?
validate :owns_notification_email, if: :notification_email_changed?
validate :owns_public_email, if: :public_email_changed?
validate :owns_commit_email, if: :commit_email_changed?

before_validation :set_notification_email, if: :new_record?
before_validation :set_public_email, if: :public_email_changed?
before_validation :set_commit_email, if: :commit_email_changed?
# in case validation is skipped
before_save :set_public_email, if: :public_email_changed?
# in case validation is skipped
before_save :set_commit_email, if: :commit_email_changed?
before_save :skip_reconfirmation!, if: ->(user) {
  user.email_changed? && user.read_only_attribute?(:email)
}
before_save :check_for_verified_email, if: ->(user) {
  user.email_changed? && !user.new_record?
}
```

gitlab/user

```ruby
validate :unique_email, if: :email_changed?
validate :owns_notification_email, if: :notification_email_changed?
validate :owns_public_email, if: :public_email_changed?
validate :owns_commit_email, if: :commit_email_changed?

before_validation :set_notification_email, if: :new_record?
before_validation :set_public_email, if: :public_email_changed?
before_validation :set_commit_email, if: :commit_email_changed?
# in case validation is skipped
before_save :set_public_email, if: :public_email_changed?
# in case validation is skipped
before_save :set_commit_email, if: :commit_email_changed?
before_save :skip_reconfirmation!, if: ->(user) {
  user.email_changed? && user.read_only_attribute?(:email)
}
before_save :check_for_verified_email, if: ->(user) {
  user.email_changed? && !user.new_record?
}
```

Ways to change?

```
validate :unique_email, if: :email_changed?
validate :owns_notification_email, if: :notification_email_changed?
validate :owns_public_email, if: :public_email_changed?
validate :owns_commit_email, if: :commit_email_changed?

before_validation :set_notification_email, if: :new_record?
before_validation :set_public_email, if: :public_email_changed?
before_validation :set_commit_email, if: :commit_email_changed?
# in case validation is skipped
before_save :set_public_email, if: :public_email_changed?
# in case validation is skipped
before_save :set_commit_email, if: :commit_email_changed?
before_save :skip_reconfirmation!, if: ->(user) {
  user.email_changed? && user.read_only_attribute?(:email)
}
before_save :check_for_verified_email, if: ->(user) {
  user.email_changed? && !user.new_record?
}
```

gitlab/user

```
validate :unique_email, if: :email_changed?
validate :owns_notification_email, if: :notification_email_changed?
validate :owns_public_email, if: :public_email_changed?
validate :owns_commit_email, if: :commit_email_changed?

before_validation :set_notification_email, if: :new_record?
before_validation :set_public_email, if: :public_email_changed?
before_validation :set_commit_email, if: :commit_email_changed?
# in case validation is skipped
before_save :set_public_email, if: :public_email_changed?
# in case validation is skipped
before_save :set_commit_email, if: :commit_email_changed?
before_save :skip_reconfirmation!, if: ->(user) {
  user.email_changed? && user.read_only_attribute?(:email)
}
before_save :check_for_verified_email, if: ->(user) {
  user.email_changed? && !user.new_record?
}
```

gitlab/user

```
validate :unique_email, if: :email_changed?
validate :owns_notification_email, if: :notification_email_changed?
validate :owns_public_email, if: :public_email_changed?
validate :owns_commit_email, if: :commit_email_changed?

before_validation :set_notification_email, if: :new_record?
before_validation :set_public_email, if: :public_email_changed?
before_validation :set_commit_email, if: :commit_email_changed?
# in case validation is skipped
before_save :set_public_email, if: :public_email_changed?
# in case validation is skipped
before_save :set_commit_email, if: :commit_email_changed?
before_save :skip_reconfirmation!, if: ->(user) {
  user.email_changed? && user.read_only_attribute?(:email)
}
before_save :check_for_verified_email, if: ->(user) {
  user.email_changed? && !user.new_record?
}
```

Why are we doing this?


Affordance


Wants to be cuddled


Wants to be fed

OOP Affordance

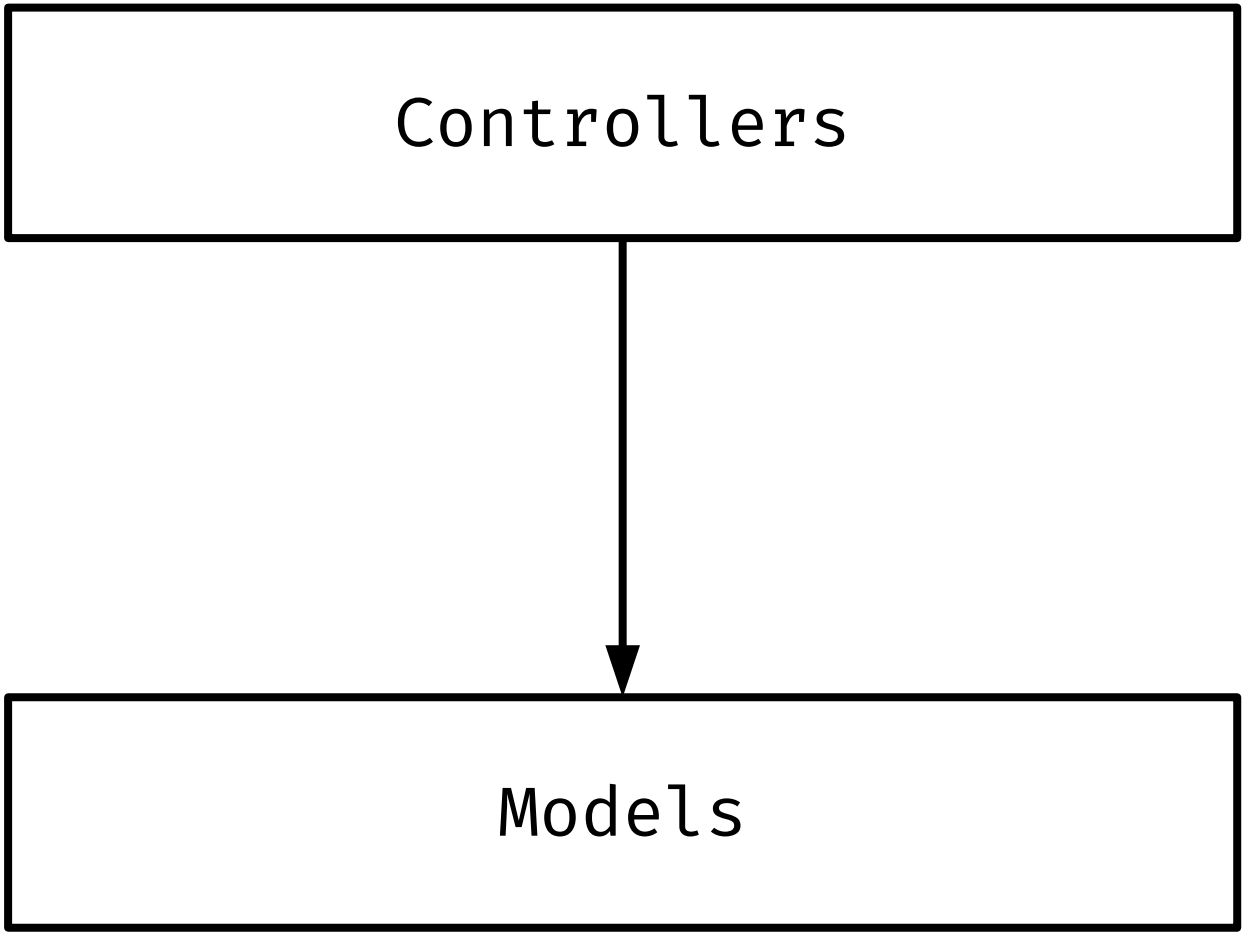
```
class MySolution
  def do_thing(argument)
  end
end
```

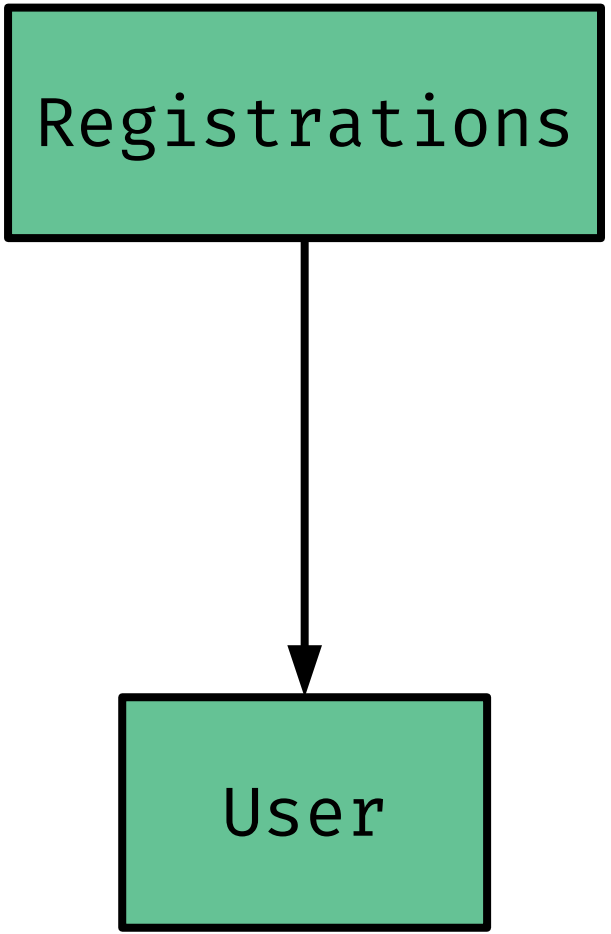
Model
View
Controller

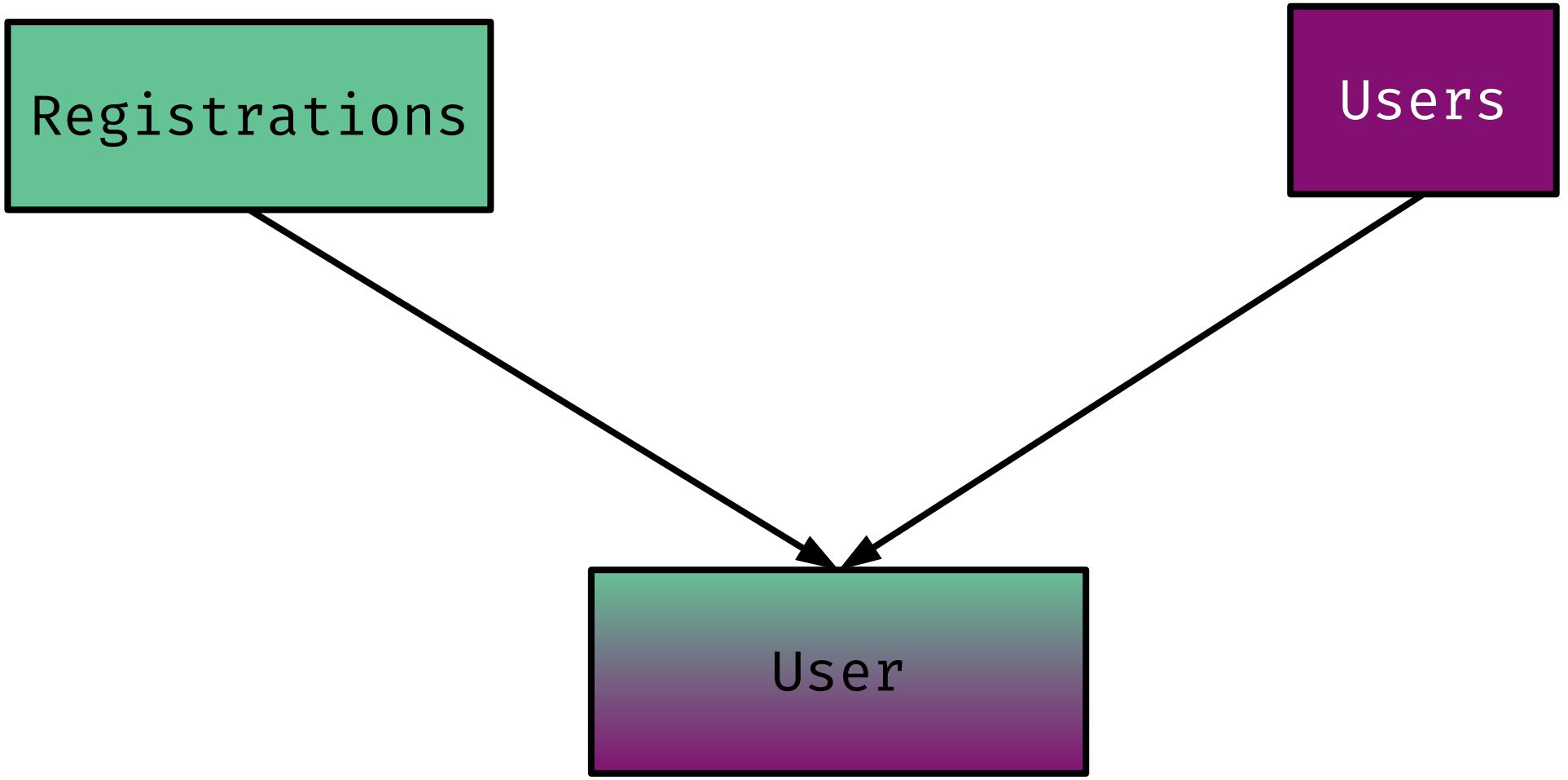
Rails Affordance

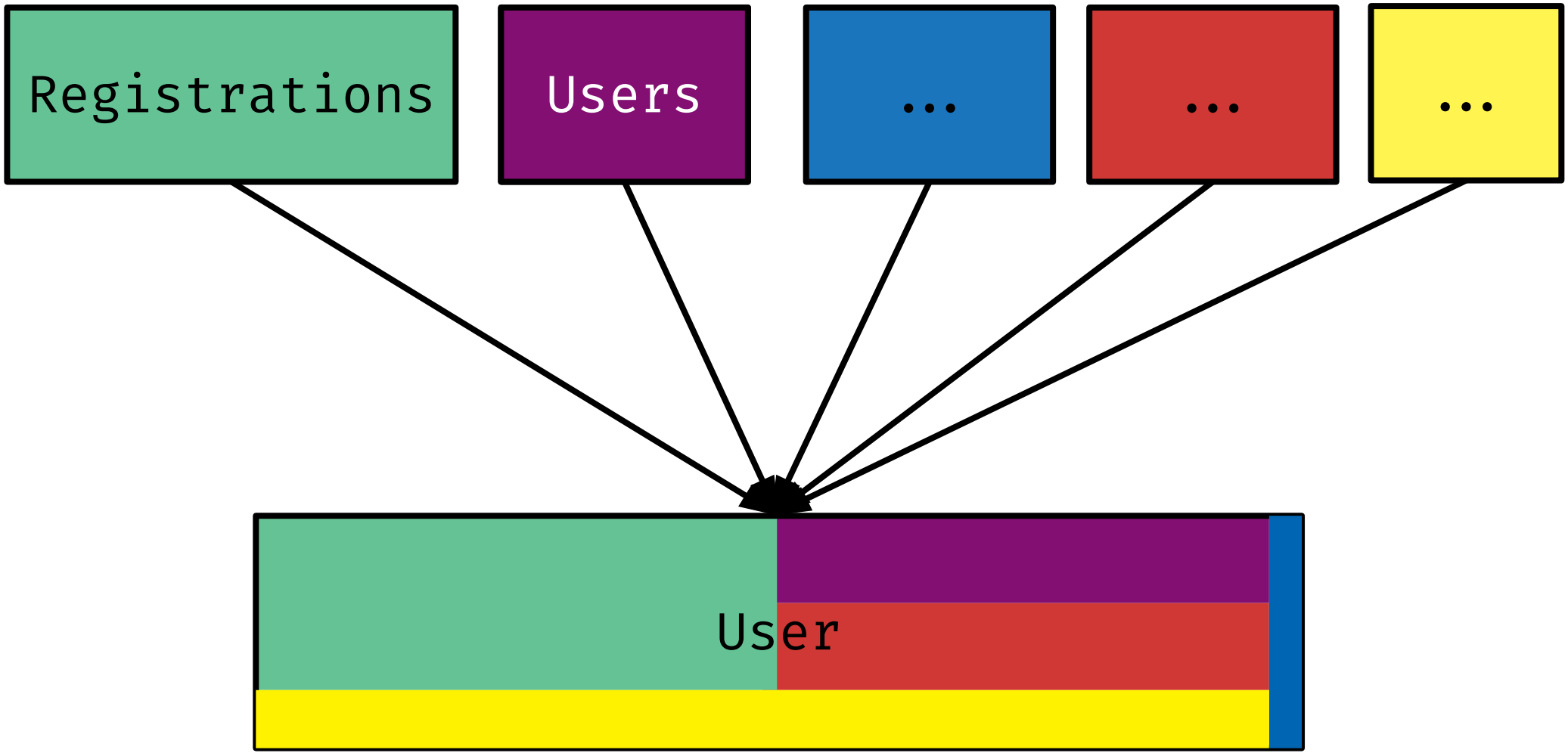
“Fat Models
Skinny Controllers”

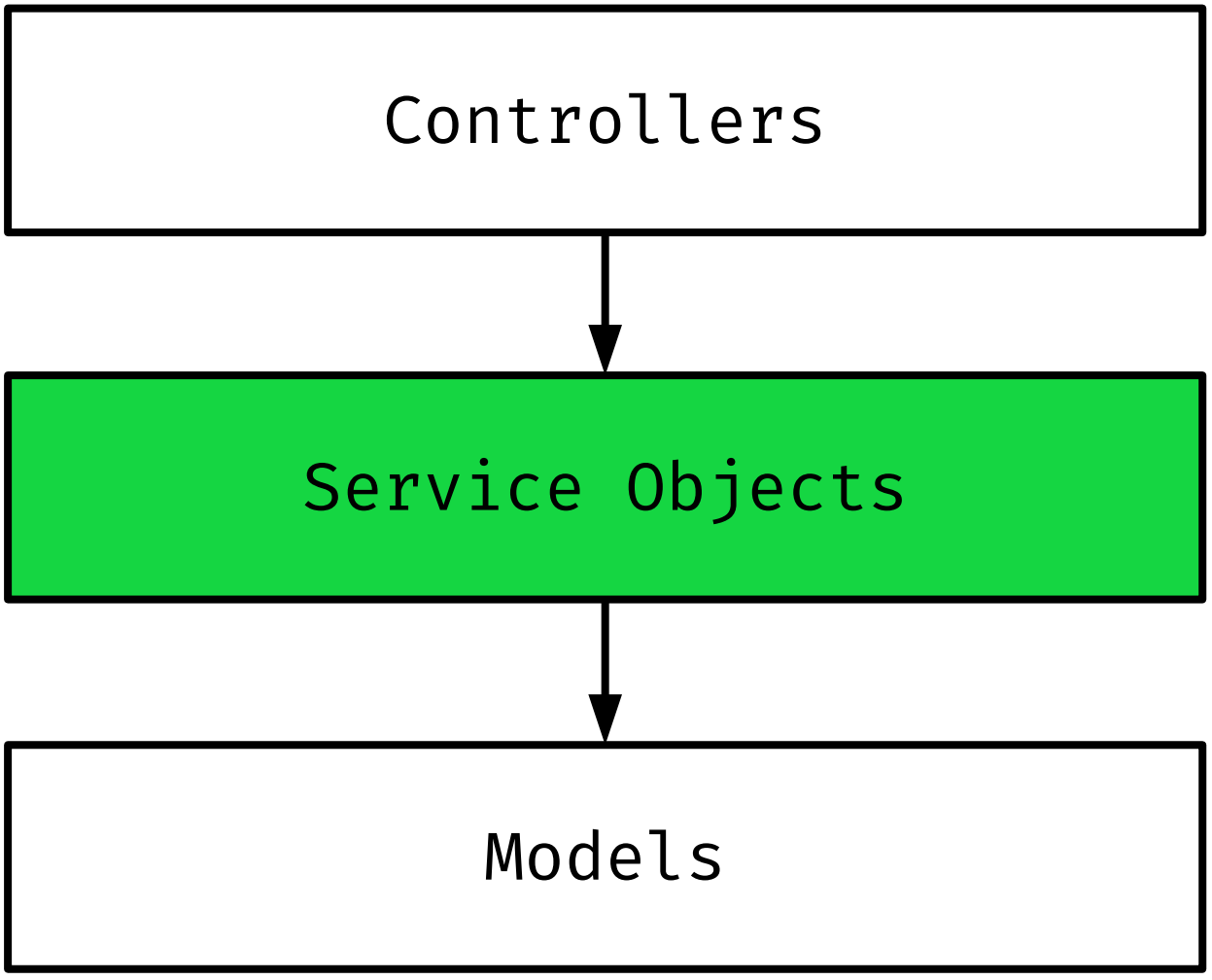
1 or 2 use cases
stuck on **every model**

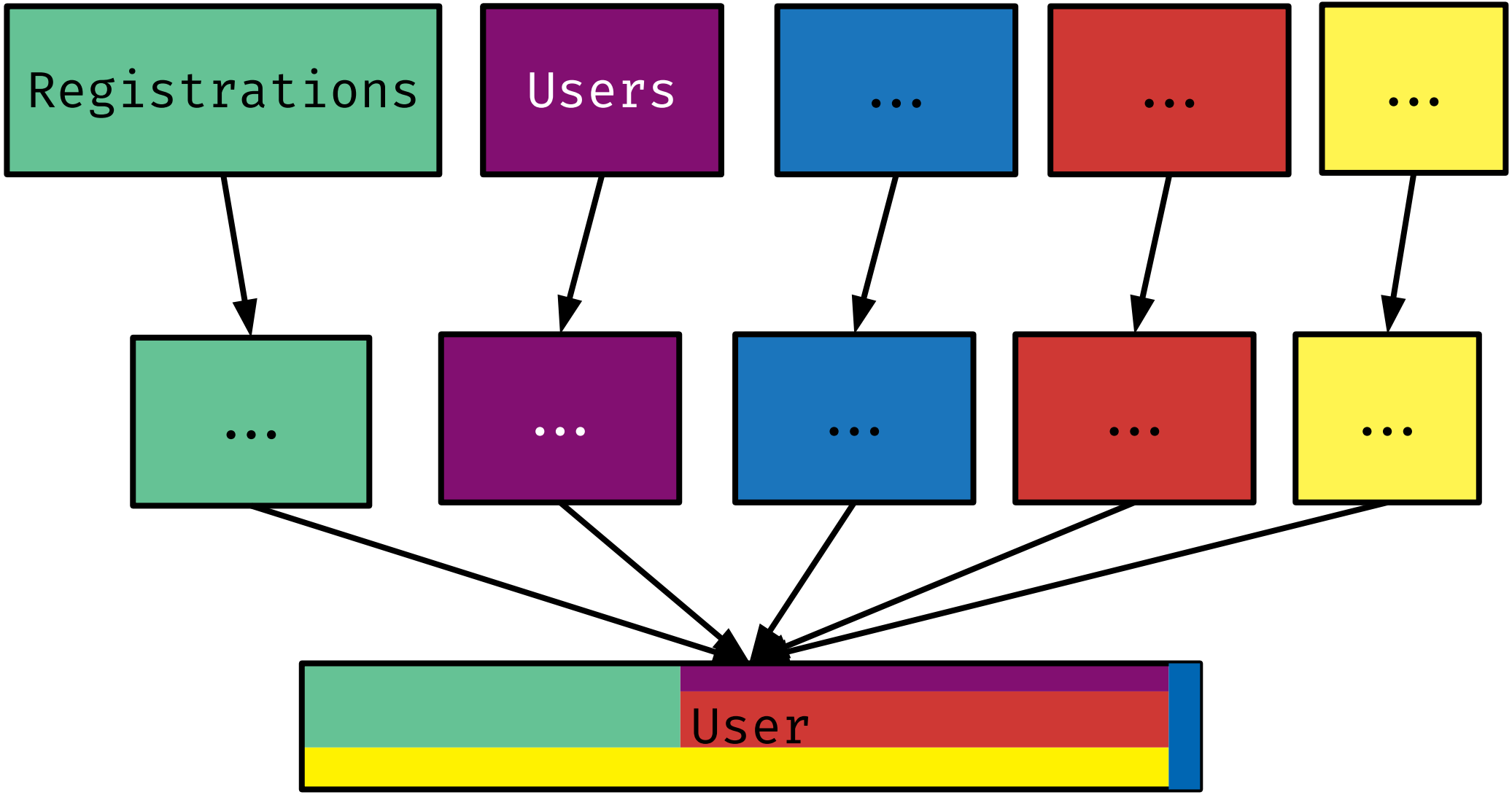

Controllers
Models

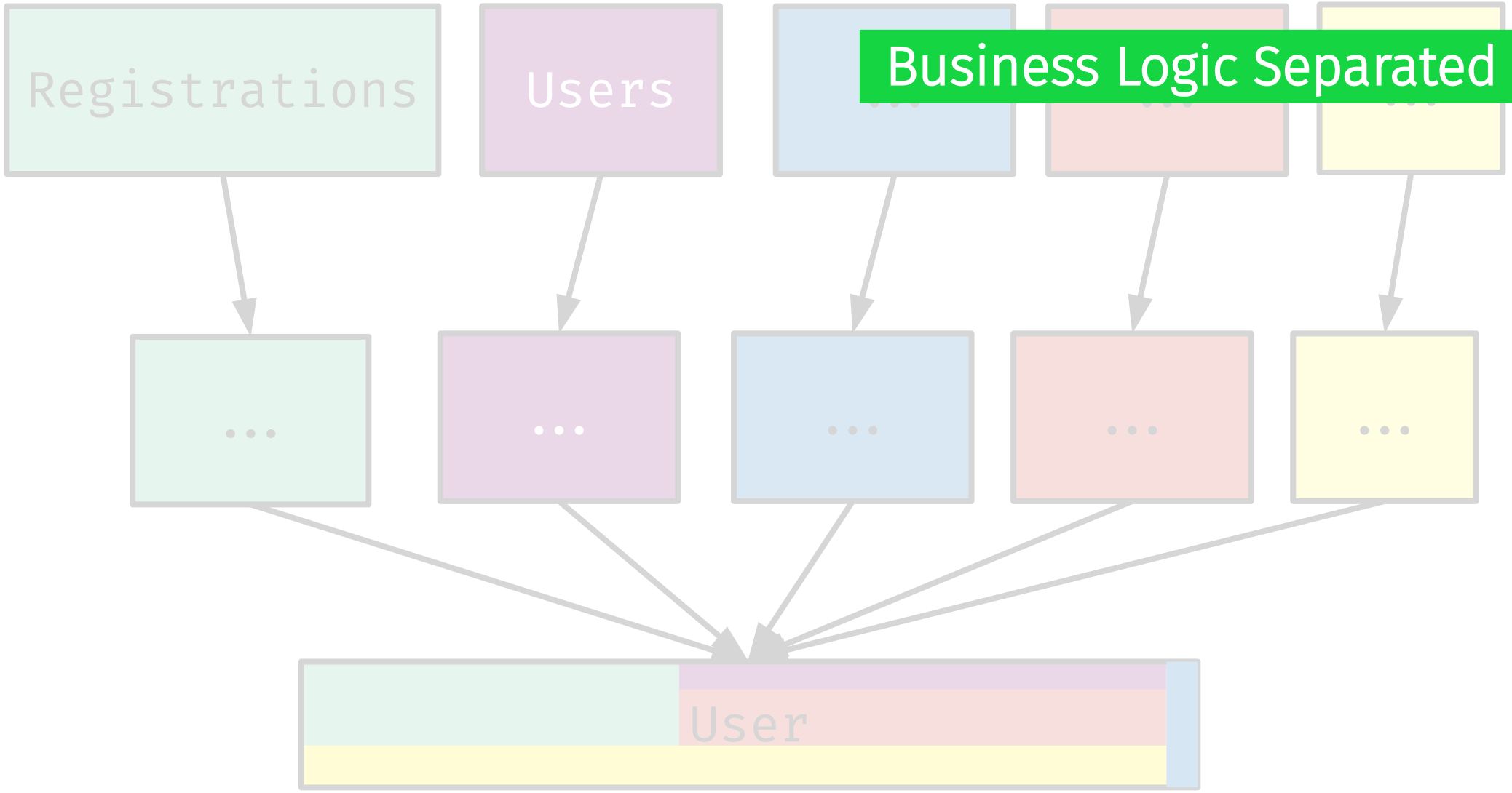

Registrations
User

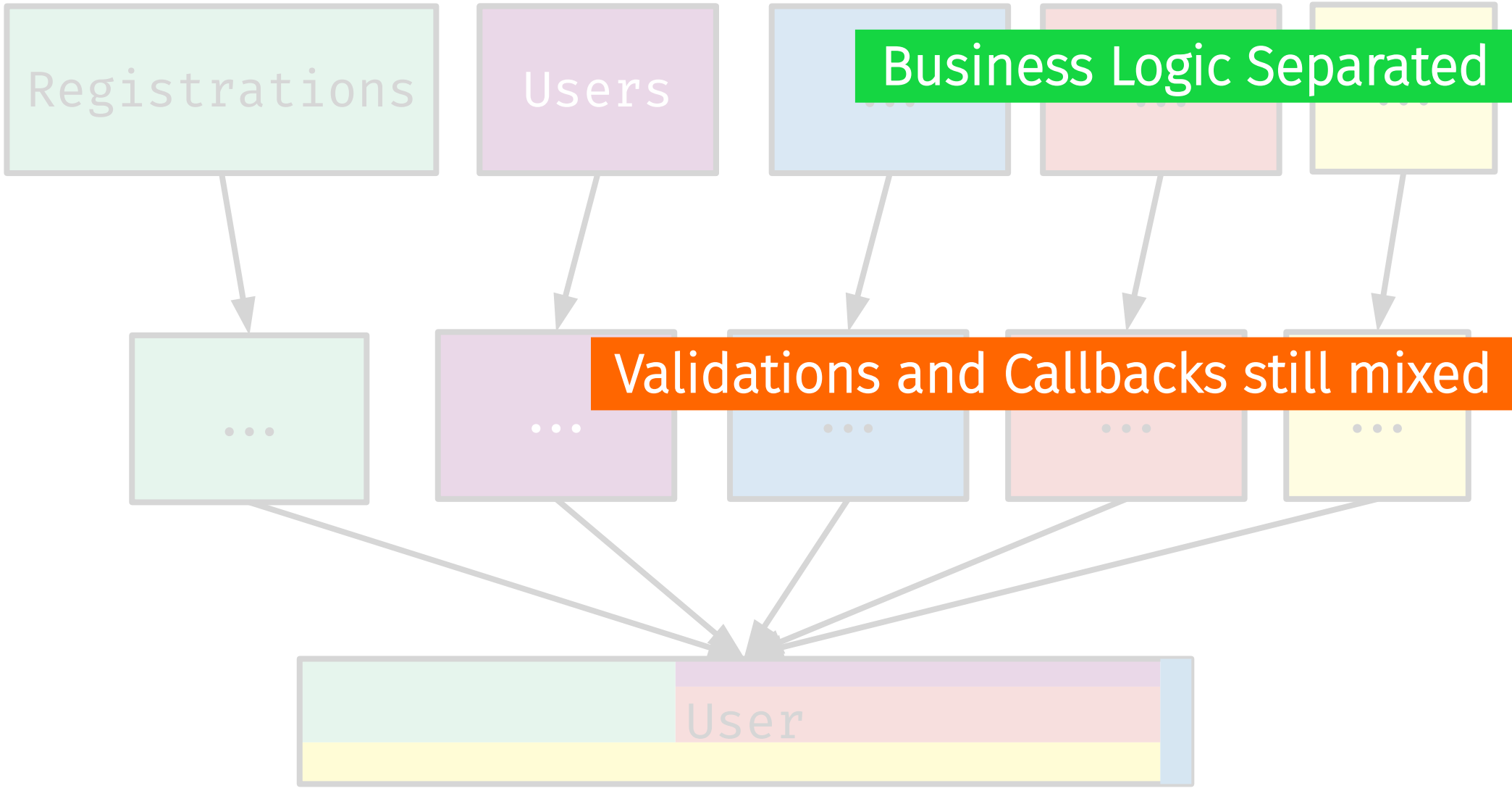

Registrations
Users
User

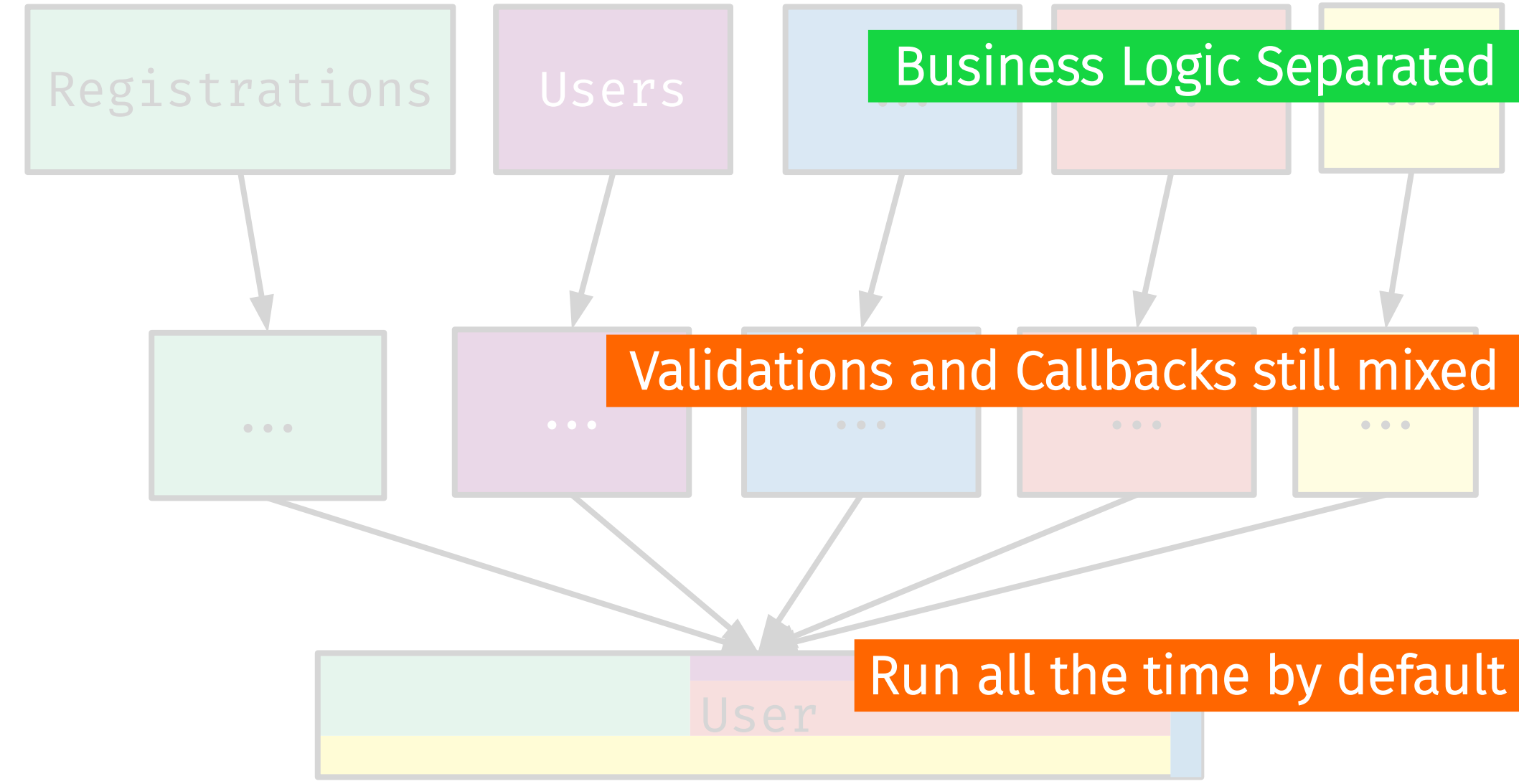

Registrations
Users
...
...
...
User

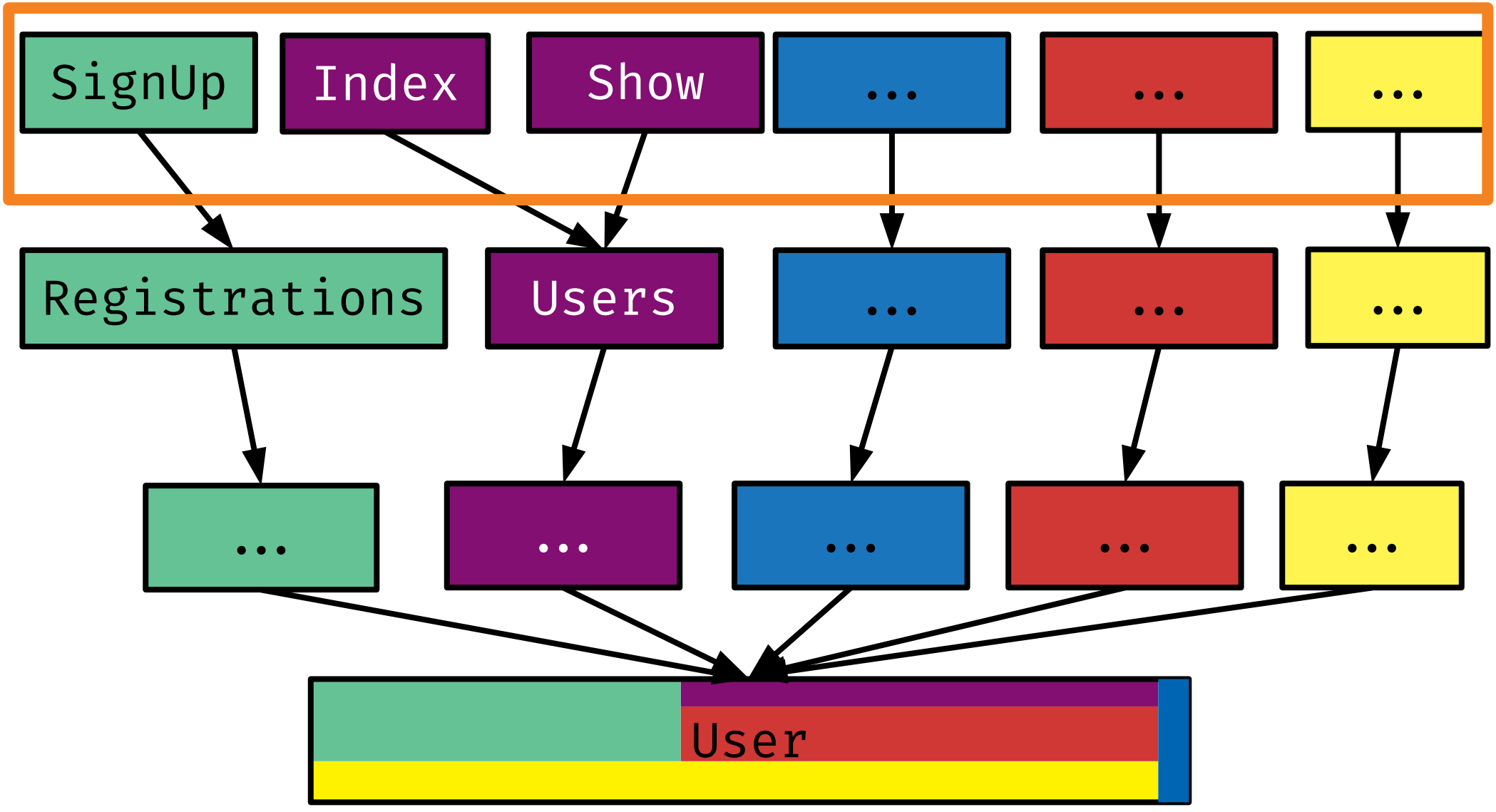

Controllers
Service Objects
Models

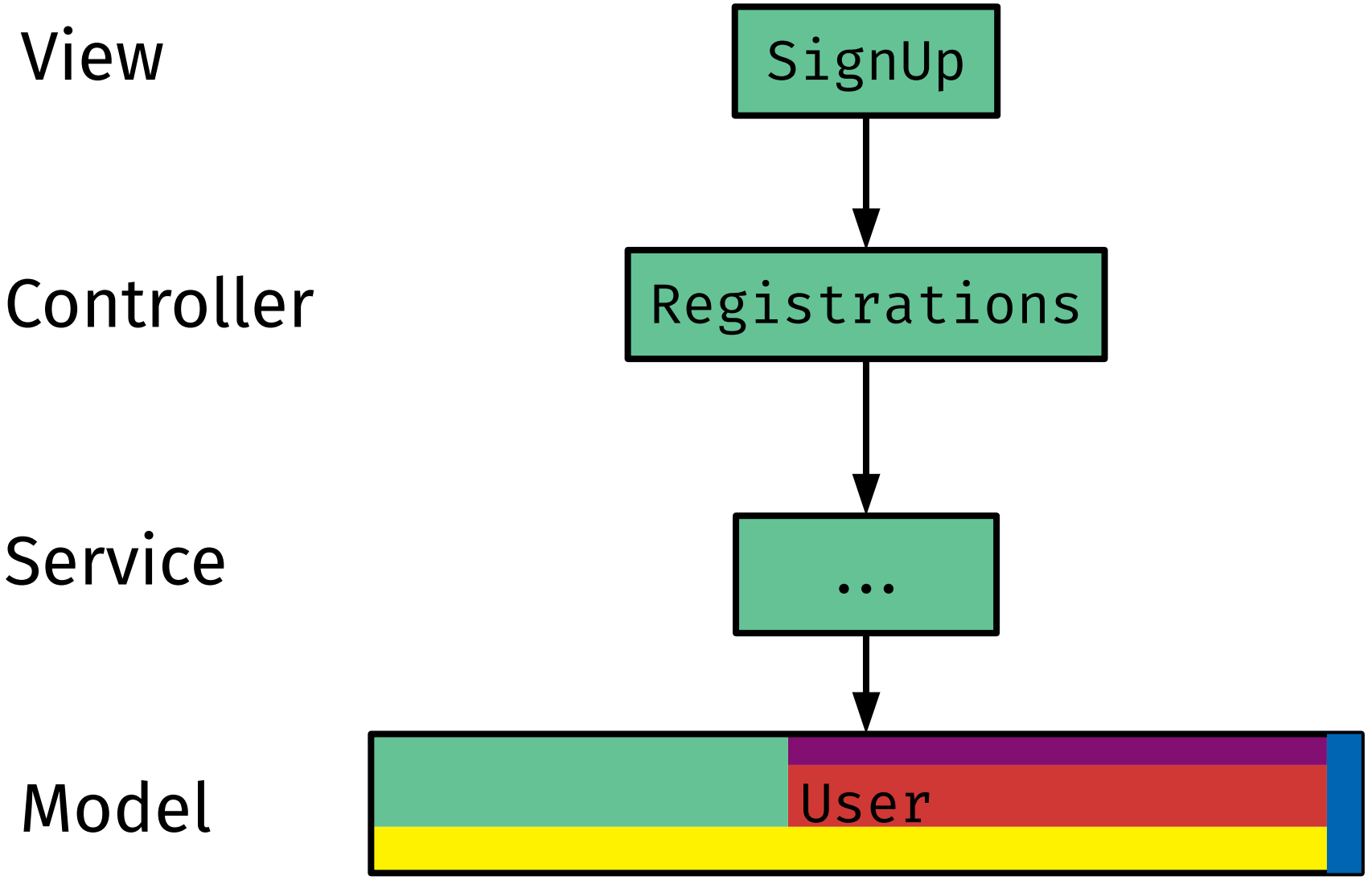

Registrations
Users
...
...
...
...
...
...
...
...
...
User


Registrations
Users
Business Logic Separated
...
...
...
...
...
User


Registrations
Users
...
...
...
...
...
...
...
...
...
Business Logic Separated
Validations and Callbacks still mixed
User


Registrations
Users
...
...
...
...
...
...
...
...
...
User
Business Logic Separated
Validations and Callbacks still mixed
Run all the time by default


SignUp
Index
Show
...
...
...
Registrations
Users
...
...
...
...
...
...
...
...
User


View
SignUp
Controller
Registrations
Service
...
Model
User

A User Model

```ruby
class User < ApplicationRecord
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

```ruby
class User < ApplicationRecord
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

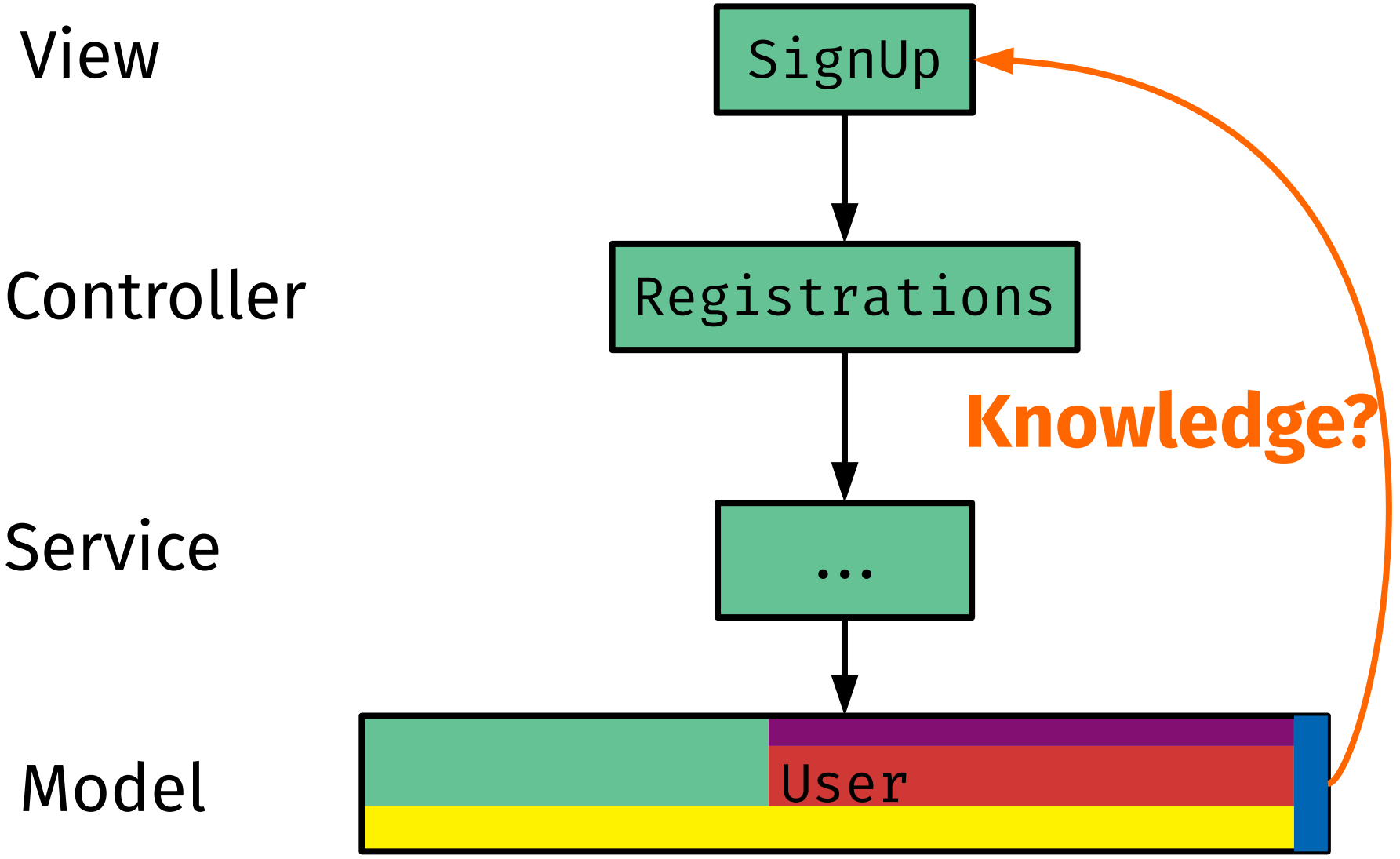
Sign Up / Edit Only

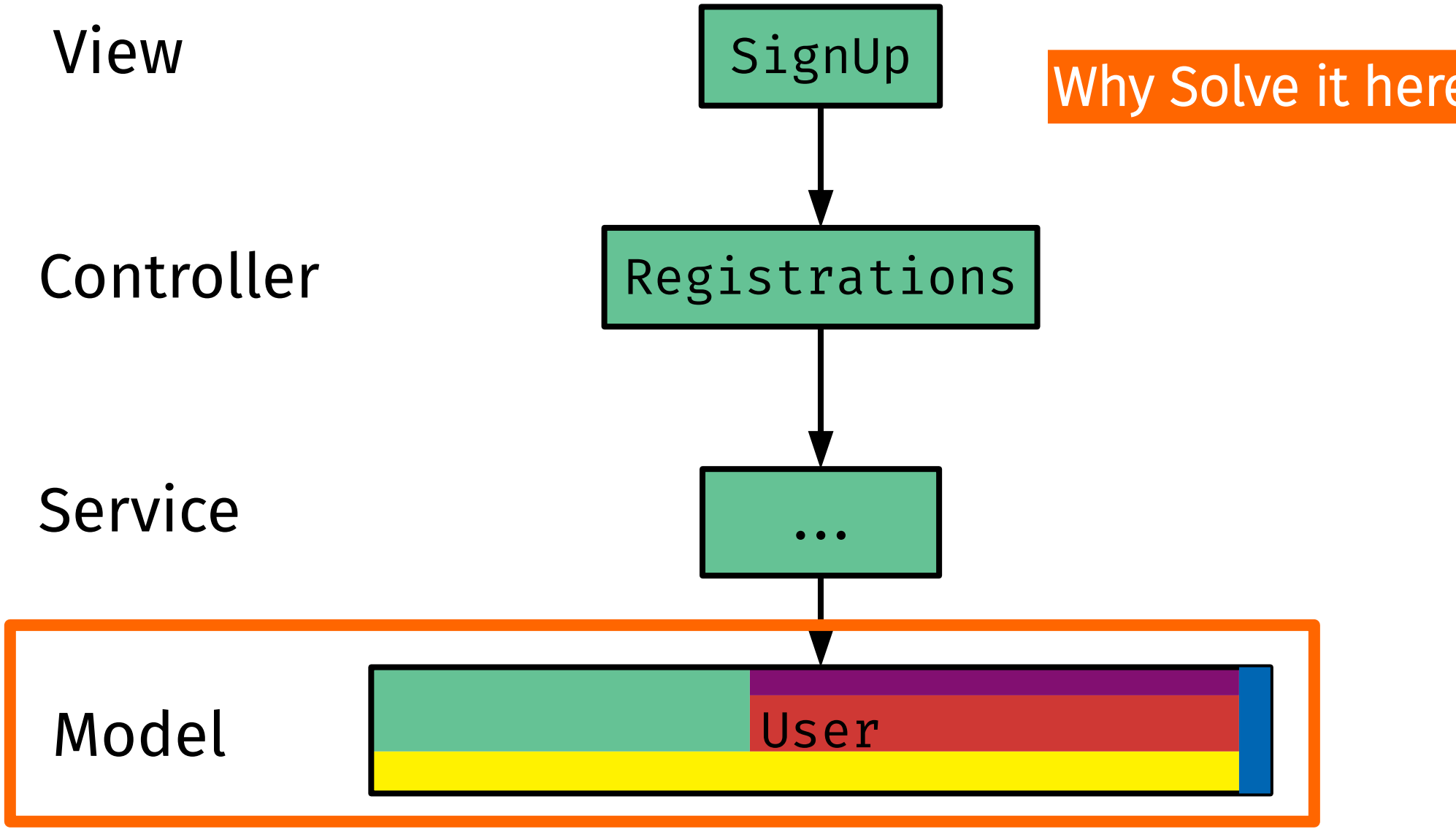
```ruby
class User < ApplicationRecord
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

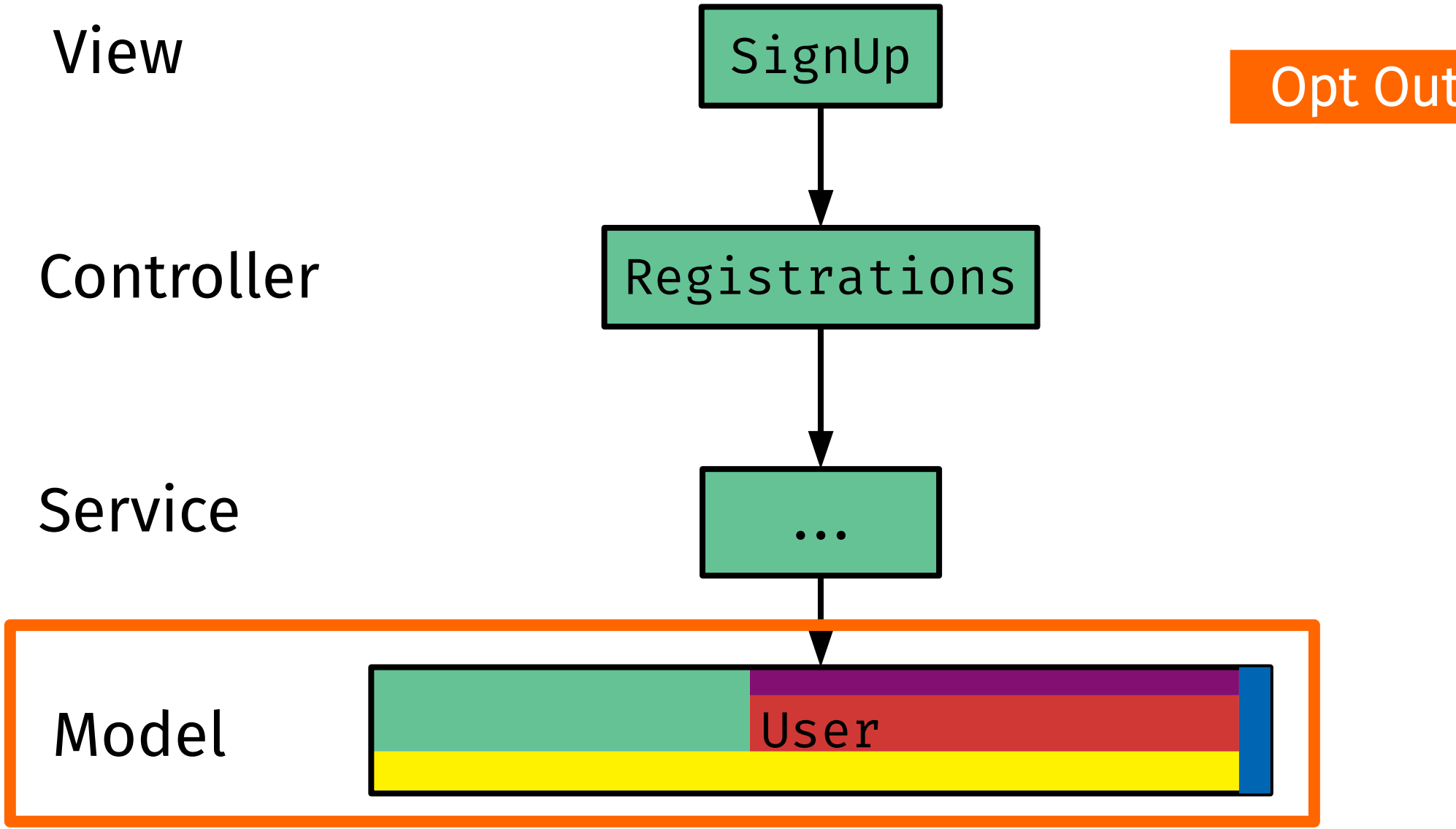
View Related

```ruby
class User < ApplicationRecord
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

WHWWHHYYY???


View
Controller
Service
Model
SignUp
Registrations
...
User
Knowledge?


View
SignUp
Why Solve it here?
Controller
Registrations
Service
...
Model
User

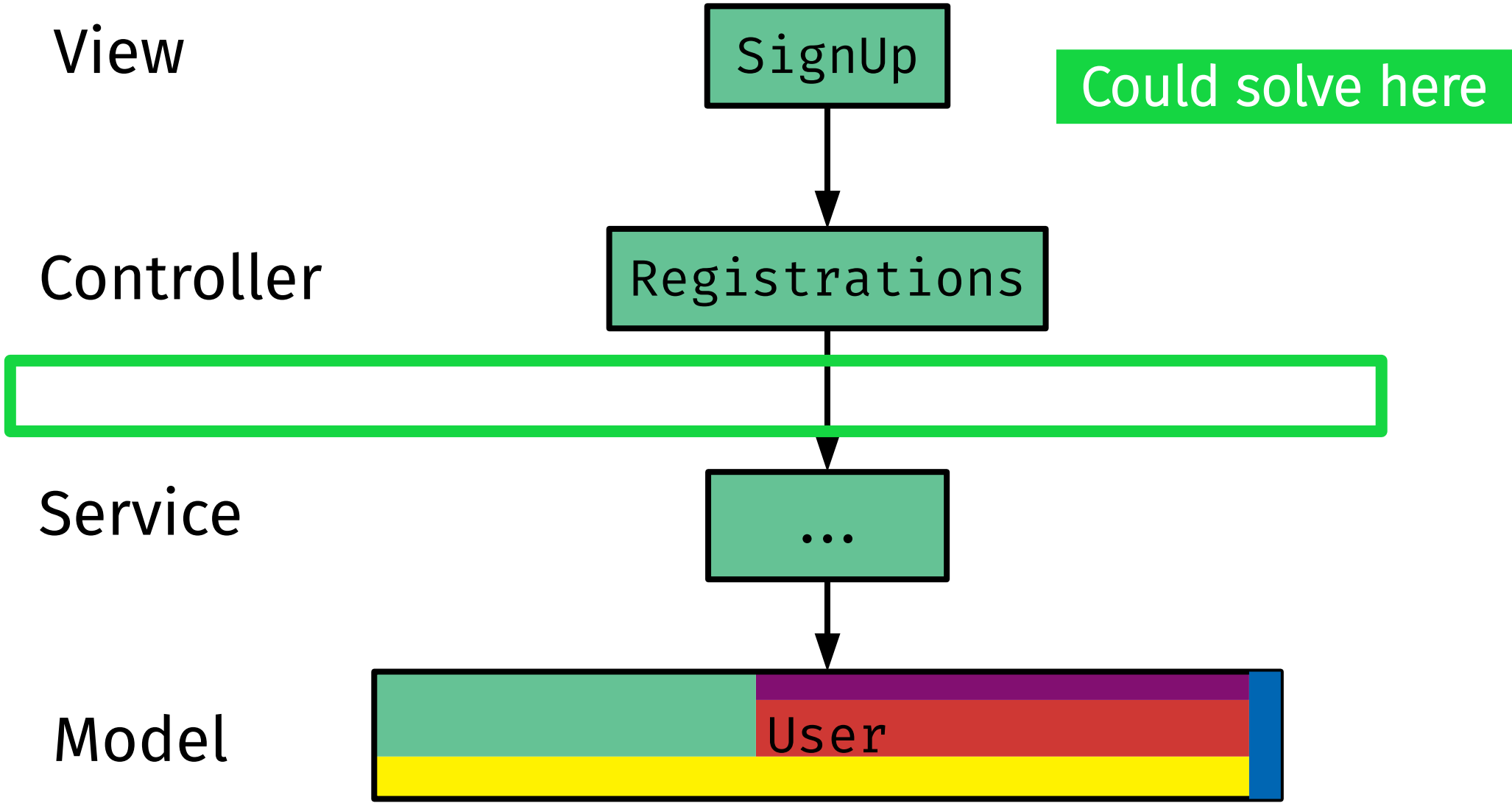

View
SignUp
Opt Out
Controller
Registrations
Service
...
Model
User

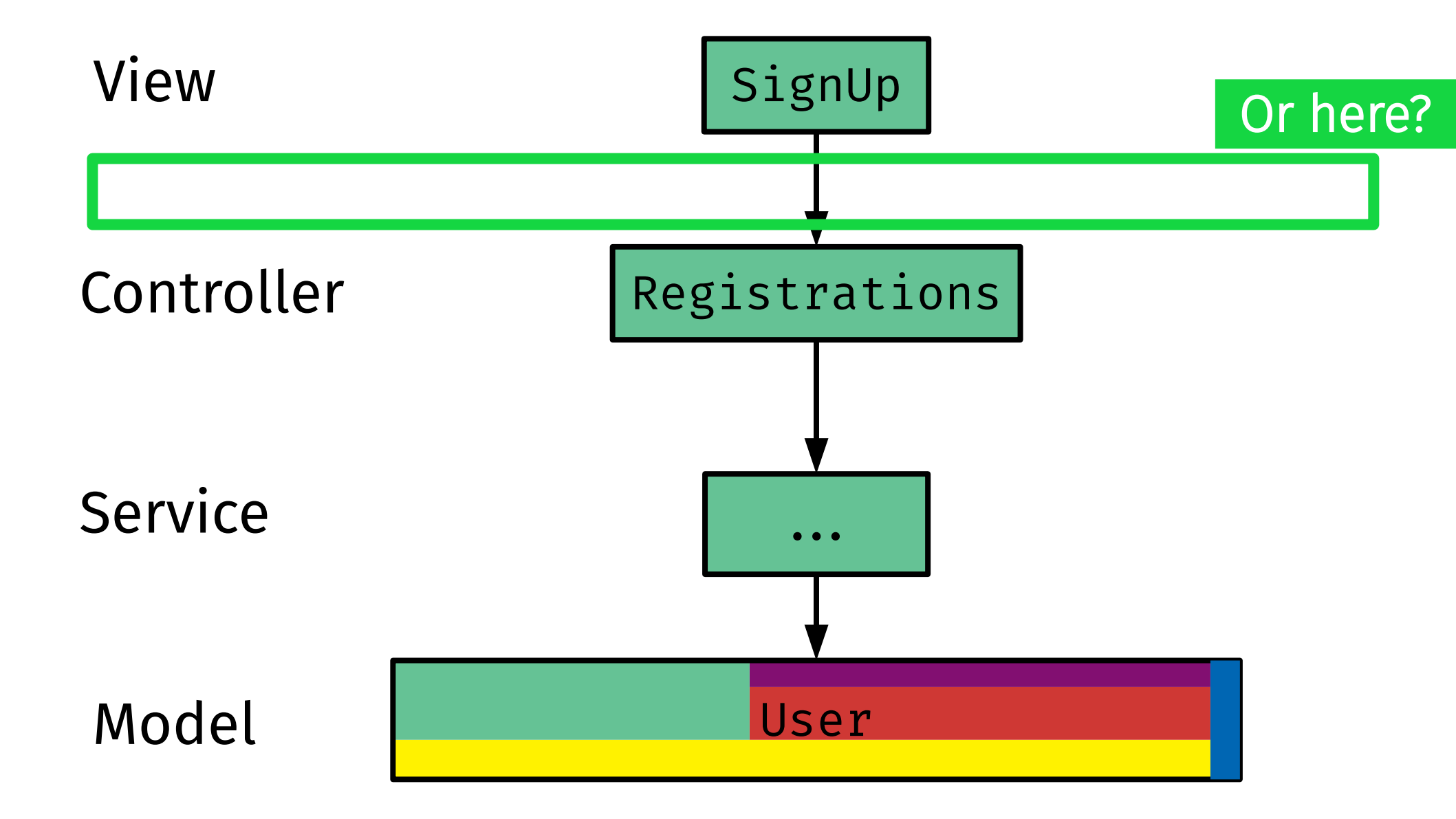
RUN ALL THE CALLBACKS

ALL THE TIME

memegenerator.net

WELL, ACTUALLY


View
SignUp
Could solve here
Controller
Registrations
Service
...
Model
User


View
SignUp
Or here?
Controller
Registrations
Service
...
Model
User

RUN EXACTLY THE CALLBACKS

THAT I WANT TO RUN

You've seen:

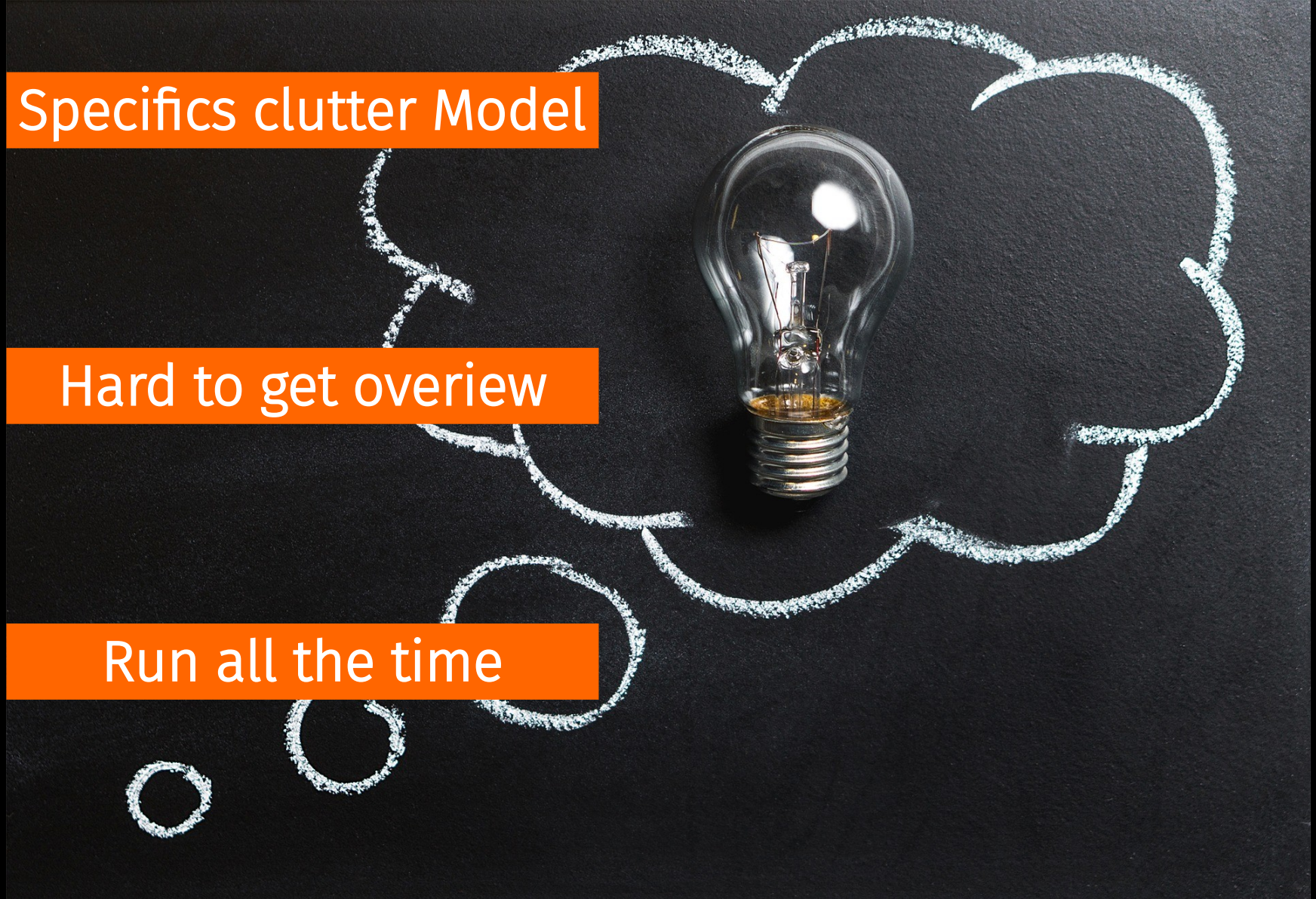
Tobi complaining about validations and callbacks

Do You Need That Validation? Let Me Call You Back About It

Tobias Pfeiffer
@PragTob
pragtob.info

Specifics clutter Model


Specifics clutter Model
Hard to get overiew


Specifics clutter Model
Hard to get overiew
Run all the time

ActiveRecord Original

```ruby
class User < ApplicationRecord
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```


What does Rails offer?

Concerns

```ruby
module UserRegistration
  extend ActiveSupport::Concern

  included do
    validates :email,
              presence: true,
              confirmation: true
    validates :password,
              confirmation: true,
              length: { minimum: 8 }
    validates :terms, acceptance: true

    attr_accessor :password

    before_save :hash_password
    after_commit :send_welcome_email, on: :create
  end
end
```

Suppress

```ruby
module Copyable
  def copy_to(destination)
   Notification.suppress do
     # Copy logic that creates new
     # comments that we do not want
     # triggering notifications.
   end
  end
end
```

Custom Contexts

```
class Person < ApplicationRecord
  validates :email,
            uniqueness: true,
            on: :account_setup
  validates :age,
            numericality: true,
            on: :account_setup
end
```


What’s out there?

Form Objects

Form Objects

```
class Registration
  include ActiveModel::Model

  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :email, :password

  def save
    if valid?
      user = BaseUser.new(email: email, password_digest: hash_password)
      user.save!
      send_welcome_email
      true
    else
      false
    end
  end
end
```

Plain ActiveModel

```ruby
class Registration
  include ActiveModel::Model

  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :email, :password

  def save
    if valid?
      user = BaseUser.new(email: email, password_digest: hash_password)
      user.save!
      send_welcome_email
      true
    else
      false
    end
  end
end
```

Validations

```ruby
class Registration
  include ActiveModel::Model

  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :email, :password

  # ...
end
```

Attributes

```
class Registration
  include ActiveModel::Model

  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :email, :password

  # ...
end
```

```ruby
class Registration
  # ...

  def save
    if valid?
      user = BaseUser.new(
                email: email,
                password_digest: hash_password
              )
      user.save!
      send_welcome_email
      true
    else
      false
    end
  end
end
```

Map to ActiveRecord

```ruby
class Registration
  # ...

  def save
    if valid?
      user = BaseUser.new(
                email: email,
                password_digest: hash_password
              )
      user.save!
      send_welcome_email
      true
    else
      false
    end
  end
end
```

Interface

Callbacks

```
class Registration
  # ...

  def save
    if valid?
      user = BaseUser.new(
                email: email,
                password_digest: hash_password
              )
      user.save!
      send_welcome_email
      true
    else
      false
    end
  end
end
```

Same Interface

```
def create
  @user = Registration.new(registration_params)

  if @user.save
    # ...
  else
    # ...
  end
end
```


Inheritance!

```ruby
class User::AsSignUp < ActiveType::Record[User]
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

Inheritance!

```ruby
class User::AsSignUp < ActiveType::Record[User]
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

Inheritance!

```ruby
class User::AsSignUp < ActiveType::Record[User]
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

ActiveType

Original

```ruby
class User < ApplicationRecord
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

```ruby
class User::AsSignUp < ActiveType::Record[User]
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

Almost the same!

```ruby
class User::AsSignUp < ActiveType::Record[User]
  validates :email,
            presence: true,
            confirmation: true
  validates :password,
            confirmation: true,
            length: { minimum: 8 }
  validates :terms, acceptance: true

  attr_accessor :password

  before_save :hash_password
  after_commit :send_welcome_email, on: :create
  # ...
end
```

Handle STI, Routes etc.


Changesets
change

Changesets

```elixir
defmodule ValidationShowcase.Accounts.User do
  # ...

  def registration_changeset(user, attrs) do
    user
    |> cast(attrs, [:email, :password, :terms_of_service])
    |> validate_required([:email, :password])
    |> validate_confirmation(:email)
    |> validate_confirmation(:password)
    |> validate_length(:password, min: 8)
    |> validate_acceptance(:terms_of_service)
    |> hash_password()
  end
end
```

Elixir

```elixir
defmodule ValidationShowcase.Accounts.User do

  # ...

  def registration_changeset(user, attrs) do
    user
    |> cast(attrs, [:email, :password, :terms_of_service])
    |> validate_required([:email, :password])
    |> validate_confirmation(:email)
    |> validate_confirmation(:password)
    |> validate_length(:password, min: 8)
    |> validate_acceptance(:terms_of_service)
    |> hash_password()
  end
end
```

Pipe

```elixir
defmodule ValidationShowcase.Accounts.User do

  # ...

  def registration_changeset(user, attrs) do
    user
    |> cast(attrs, [:email, :password, :terms_of_service])
    |> validate_required([:email, :password])
    |> validate_confirmation(:email)
    |> validate_confirmation(:password)
    |> validate_length(:password, min: 8)
    |> validate_acceptance(:terms_of_service)
    |> hash_password()
  end
end
```

```elixir
defmodule ValidationShowcase.Accounts.User do

  # ...

  def registration_changeset(user, attrs) do
    user
    |> cast(attrs, [:email, :password, :terms_of_service])
    |> validate_required([:email, :password])
    |> validate_confirmation(:email)
    |> validate_confirmation(:password)
    |> validate_length(:password, min: 8)
    |> validate_acceptance(:terms_of_service)
    |> hash_password()
  end
end
```

Context

```elixir
defmodule ValidationShowcase.Accounts.User do

  # ...

  def registration_changeset(user, attrs) do
    user
    |> cast(attrs, [:email, :password, :terms_of_service])
    |> validate_required([:email, :password])
    |> validate_confirmation(:email)
    |> validate_confirmation(:password)
    |> validate_length(:password, min: 8)
    |> validate_acceptance(:terms_of_service)
    |> hash_password()
  end
end
```

"strong parameters"

Validations

```elixir
defmodule ValidationShowcase.Accounts.User do

  # ...

  def registration_changeset(user, attrs) do
    user
    |> cast(attrs, [:email, :password, :terms_of_service])
    |> validate_required([:email, :password])
    |> validate_confirmation(:email)
    |> validate_confirmation(:password)
    |> validate_length(:password, min: 8)
    |> validate_acceptance(:terms_of_service)
    |> hash_password()
  end
end
```

```elixir
defmodule ValidationShowcase.Accounts.User do

  # ...

  def registration_changeset(user, attrs) do
    user
    |> cast(attrs, [:email, :password, :terms_of_service])
    |> validate_required([:email, :password])
    |> validate_confirmation(:email)
    |> validate_confirmation(:password)
    |> validate_length(:password, min: 8)
    |> validate_acceptance(:terms_of_service)
    |> hash_password()
  end
end
```

callback

```elixir
defmodule ValidationShowcase.Accounts.User do

  # ...

  def registration_changeset(user, attrs) do
    user
    |> cast(attrs, [:email, :password, :terms_of_service])
    |> validate_required([:email, :password])
    |> validate_confirmation(:email)
    |> validate_confirmation(:password)
    |> validate_length(:password, min: 8)
    |> validate_acceptance(:terms_of_service)
    |> hash_password()
  end
end
```

Mixing concerns

Remember this?

```
validate :unique_email, if: :email_changed?
validate :owns_notification_email, if: :notification_email_changed?
validate :owns_public_email, if: :public_email_changed?
validate :owns_commit_email, if: :commit_email_changed?

before_validation :set_notification_email, if: :new_record?
before_validation :set_public_email, if: :public_email_changed?
before_validation :set_commit_email, if: :commit_email_changed?
# in case validation is skipped
before_save :set_public_email, if: :public_email_changed?
# in case validation is skipped
before_save :set_commit_email, if: :commit_email_changed?
before_save :skip_reconfirmation!, if: ->(user) {
  user.email_changed? && user.read_only_attribute?(:email)
}
before_save :check_for_verified_email, if: ->(user) {
  user.email_changed? && !user.new_record?
}
```

```elixir
defmodule ValidationShowcase.Accounts.User do
  # ...

  def registration_changeset(user, attrs) do
    user
    |> base_changeset(attrs)
    |> cast(attrs, [:email, :password, :terms_of_service])
    |> validate_required([:email, :password])
    |> validate_confirmation(:email)
    |> validate_confirmation(:password)
    |> validate_length(:password, min: 8)
    |> validate_acceptance(:terms_of_service)
    |> hash_password()
  end
end
```

Combinable

Context

```elixir
defmodule ValidationShowcase.Accounts do
  def create_user(attrs \\ %{}) do
    %User{}
    |> User.registration_changeset(attrs)
    |> Repo.insert()
    |> send_welcome_email()
  end
end
```

Changeset

```elixir
defmodule ValidationShowcase.Accounts do
  def create_user(attrs \\ %{}) do
    %User{}
    |> User.registration_changeset(attrs)
    |> Repo.insert()
    |> send_welcome_email()
  end
end
```

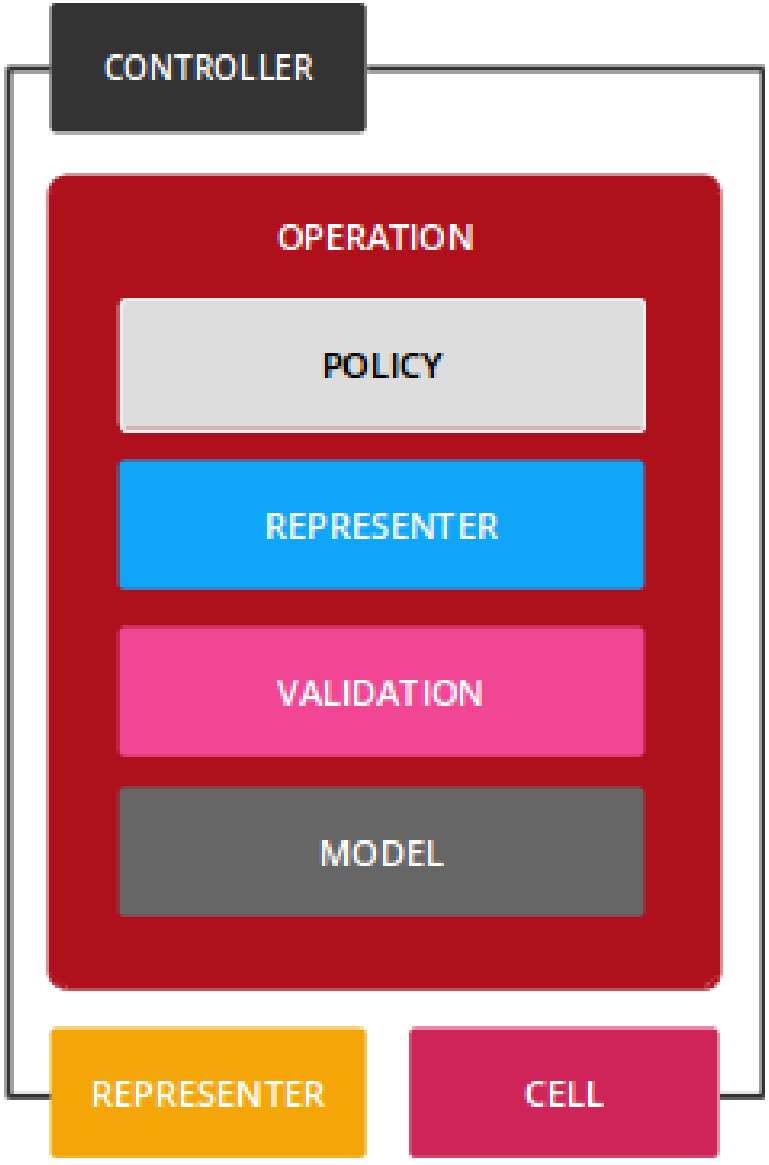
“after_commit”

```elixir
defmodule ValidationShowcase.Accounts do
  def create_user(attrs \\ %{}) do
    %User{}
    |> User.registration_changeset(attrs)
    |> Repo.insert()
    |> send_welcome_email()
  end
end
```

Controller

```elixir
defmodule ValidationShowcaseWeb.UserController do
  def create(conn, %{"user" => user_params}) do
    case Accounts.create_user(user_params) do
      {:ok, user} ->
        conn
        |> put_flash(:info, "User created successfully.")
        |> redirect(to: Routes.user_path(conn, :show, user))

      {:error, %Ecto.Changeset{} = changeset} ->
        render(conn, "new.html", changeset: changeset)
    end
  end
end
```

Separate Operations and Validators



trailblazer

```ruby
class TbRegistrationsController < ApplicationController
  def create
    result = Registration::Create.(params: params)

    if result.success?
      redirect_to "/users/", notice: 'User was created.'
    else
      @user = result["contract.default"]
      render "users/new"
    end
  end
end
```

Operation

```
class TbRegistrationsController < ApplicationController
  def create
    result = Registration::Create.(params: params)

    if result.success?
      redirect_to "/users/", notice: 'User was created.'
    else
      @user = result["contract.default"]
      render "users/new"
    end
  end
end
```

```ruby
class Registration::Create < Trailblazer::Operation
  step Model(BaseUser, :new)
  step Contract::Build(
         constant: Registration::Contract::Create
  )
  step Contract::Validate(key: :tb_registration)
  step :hash_password
  step Contract::Persist()
  step :send_welcome_email
end
```

Setup Model

```
class Registration::Create < Trailblazer::Operation
  step Model(BaseUser, :new)
  step Contract::Build(
         constant: Registration::Contract::Create
  )
  step Contract::Validate(key: :tb_registration)
  step :hash_password
  step Contract::Persist()
  step :send_welcome_email
end
```

Setup Form Object

```
class Registration::Create < Trailblazer::Operation
  step Model(BaseUser, :new)
  step Contract::Build(
         constant: Registration::Contract::Create
  )
  step Contract::Validate(key: :tb_registration)
  step :hash_password
  step Contract::Persist()
  step :send_welcome_email
end
```

```ruby
module Registration::Contract
  class Create < Reform::Form
    include Dry
    include Reform::Form::ActiveModel
    feature Coercion

    model :tb_registration

    property :email
    property :email_confirmation, virtual: true
    property :password, virtual: true
    property :password_confirmation, virtual: true
    property :terms, virtual: true, type: Types::Params::Bool

    validation do
      required(:email).filled.confirmation
      required(:password).value(min_size?: 8).confirmation
      required(:terms).value(:true?)
    end
  end
end
```

```ruby
module Registration::Contract
  class Create < Reform::Form
    include Dry
    include Reform::Form::ActiveModel
    feature Coercion

    model :tb_registration

    property :email
    property :email_confirmation, virtual: true
    property :password, virtual: true
    property :password_confirmation, virtual: true
    property :terms, virtual: true, type: Types::Params::Bool

    validation do
      required(:email).filled.confirmation
      required(:password).value(min_size?: 8).confirmation
      required(:terms).value(:true?)
    end
  end
end
```

```ruby
module Registration::Contract
  class Create < Reform::Form
    property :email
    property :email_confirmation, virtual: true
    property :password, virtual: true
    property :password_confirmation, virtual: true
    property :terms, virtual: true,
                     type: Types::Params::Bool

    validation do
      required(:email).filled.confirmation
      required(:password).value(min_size?: 8).confirmation
      required(:terms).value(:true?)
    end
  end
end
```

```ruby
module Registration::Contract
  class Create < Reform::Form
    property :email
    property :email_confirmation, virtual: true
    property :password, virtual: true
    property :password_confirmation, virtual: true
    property :terms, virtual: true,
                     type: Types::Params::Bool

    validation do
      required(:email).filled.confirmation
      required(:password).value(min_size?: 8).confirmation
      required(:terms).value(:true?)
    end
  end
end
```

Attributes

dry-validation

```ruby
module Registration::Contract
  class Create < Reform::Form
    property :email
    property :email_confirmation, virtual: true
    property :password, virtual: true
    property :password_confirmation, virtual: true
    property :terms, virtual: true,
                     type: Types::Params::Bool

    validation do
      required(:email).filled.confirmation
      required(:password).value(min_size?: 8).confirmation
      required(:terms).value(:true?)
    end
  end
end
```

validate

```
class Registration::Create < Trailblazer::Operation
  step Model(BaseUser, :new)
  step Contract::Build(
         constant: Registration::Contract::Create
  )
  step Contract::Validate(key: :tb_registration)
  step :hash_password
  step Contract::Persist()
  step :send_welcome_email
end
```

Callback

```
class Registration::Create < Trailblazer::Operation
  step Model(BaseUser, :new)
  step Contract::Build(
         constant: Registration::Contract::Create
  )
  step Contract::Validate(key: :tb_registration)
  step :hash_password
  step Contract::Persist()
  step :send_welcome_email
end
```

Persist

```
class Registration::Create < Trailblazer::Operation
  step Model(BaseUser, :new)
  step Contract::Build(
         constant: Registration::Contract::Create
  )
  step Contract::Validate(key: :tb_registration)
  step :hash_password
  step Contract::Persist()
  step :send_welcome_email
end
```

Callback

```
class Registration::Create < Trailblazer::Operation
  step Model(BaseUser, :new)
  step Contract::Build(
         constant: Registration::Contract::Create
  )
  step Contract::Validate(key: :tb_registration)
  step :hash_password
  step Contract::Persist()
  step :send_welcome_email
end
```

trailblazer

“Models are persistence-only and solely define associations and scopes. No business code is to be found here. No validations, no callbacks.”

hanami/model

The architecture eases keeping the business logic (entities) separated from details such as persistence or validations.


Takeaway


I don’t hate Rails


I don’t hate Rails
Future in Rails?


I don’t hate Rails
Future in Rails?
Affordances


I don’t hate Rails
Future in Rails?
Affordances
Alternatives

Form Objects


Inheritance!


Changesets
change

Separate Operations and Validators


I don’t hate Rails
Future in Rails?
Affordances
Alternatives
Careful with validations and callbacks


Thank you!