Functional programming for fun and profit!!

@jenny-codes

The ReadME Project

Artwork: Susan Haejin Le

Functional programming is finally going mainstream

OBJECT-ORIENTED AND IMPERATIVE PROGRAMMING AREN'T GOING AWAY, BUT FUNCTIONAL PROGRAMMING IS FINDING ITS WAY INTO MORE CODEBASES.

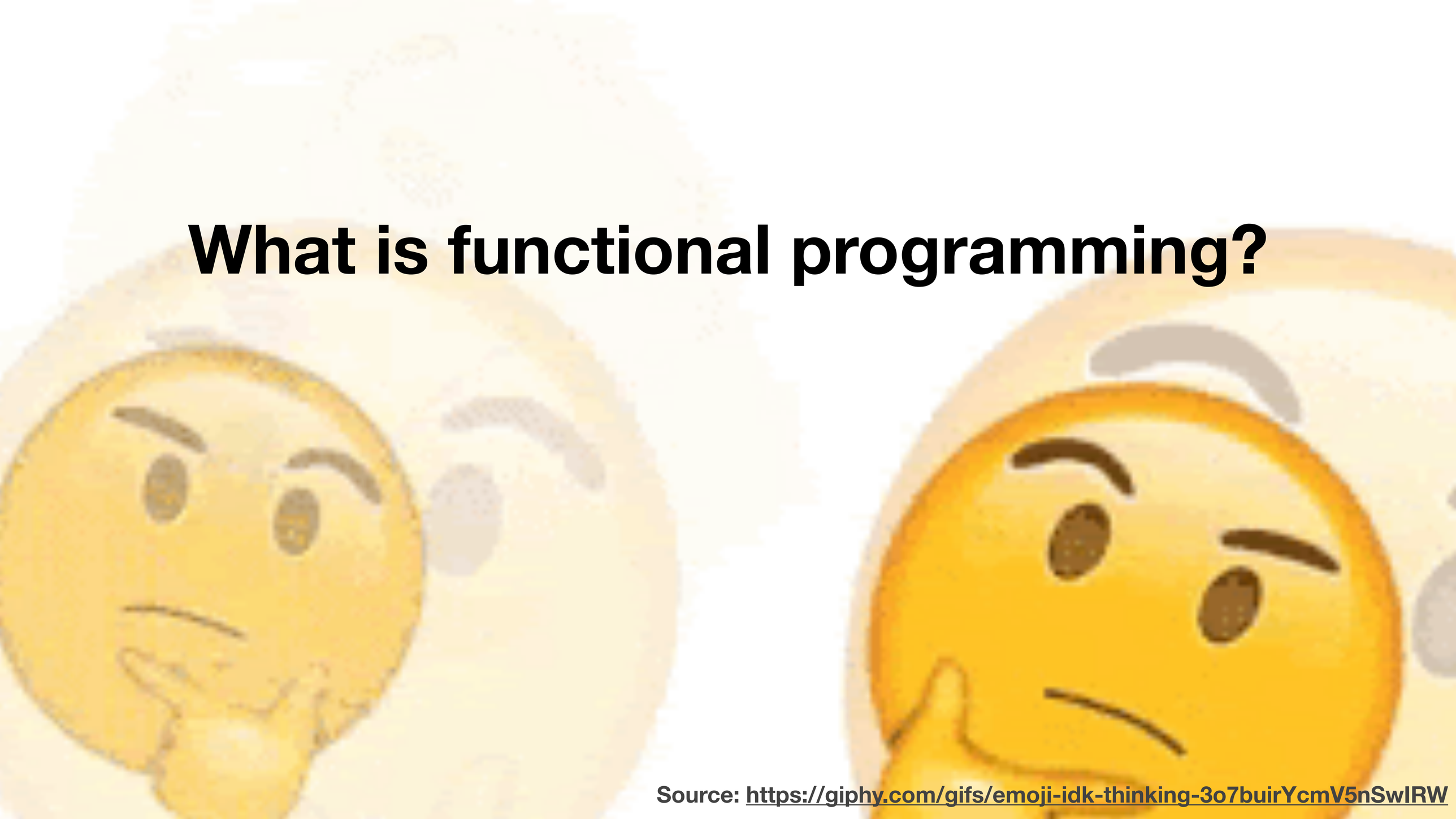
KLINT FINLEY // JULY 12, 2022

https://github.com/readme/featured/functional-programming

@jenny_codes

Source: https://youtu.be/ADqLBc1vFwI

What is functional programming?



Source: https://giphy.com/gifs/emoji-idk-thinking-3o7buirYcmV5nSwIRW

What is functional programming?

Should I care about it if I just want to write Ruby code?

Source: https://giphy.com/gifs/emoji-idk-thinking-3o7buirYcmV5nSwIRW

Functional programming helps us write better Ruby code

Jenny Shih / 施靜樺

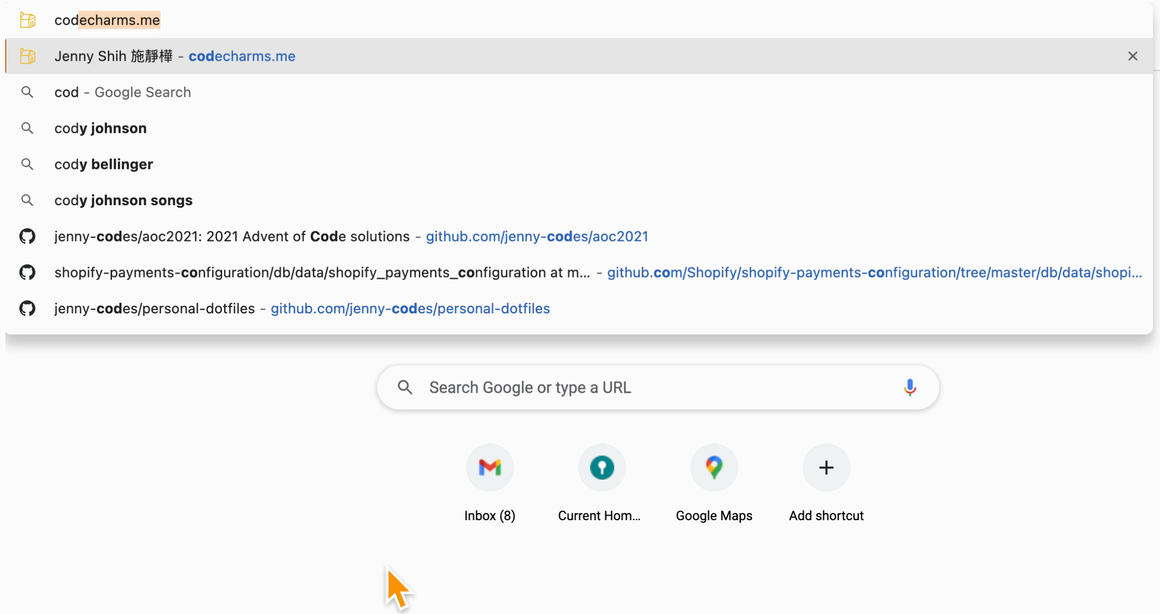
@jenny_codes

Jenny Shih / 施靜樺

🇹🇼 Taiwan -> 🇨🇦 Canada

💎 💧

https://codecharms.me



@jenny_codes

Programming paradigms

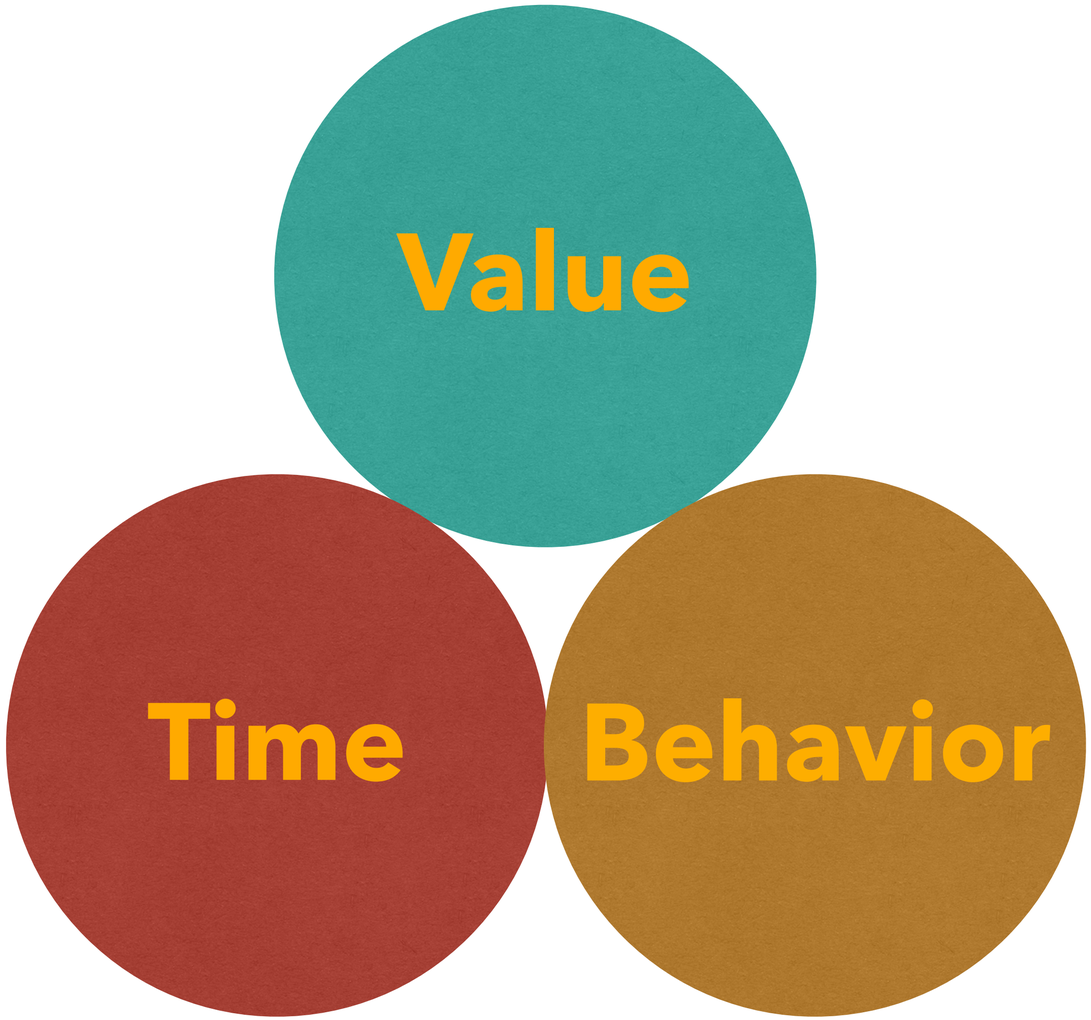
A way to classify programming languages based on their features

Programming paradigms

- Object Oriented Programming (OO)
- Functional Programming (FP)

Mental model

A program

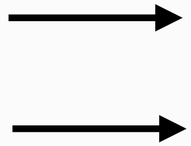


```
puts "Hello world"
```

```
puts "Hello world"
```

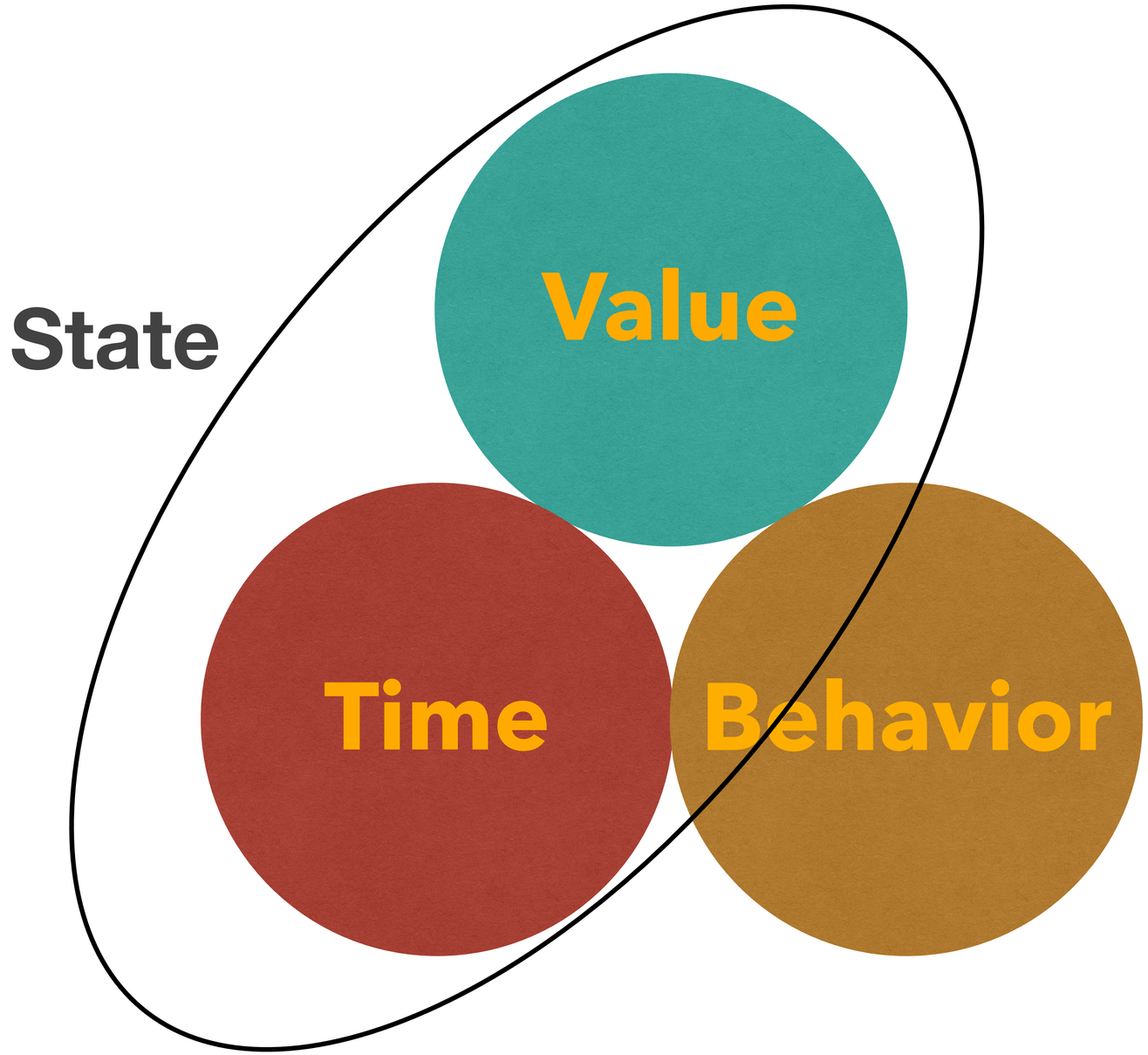
```ruby
conf = "RubyConf Mini"
conf.upcase!
puts conf
# => RUBYCONF MINI
```

Time

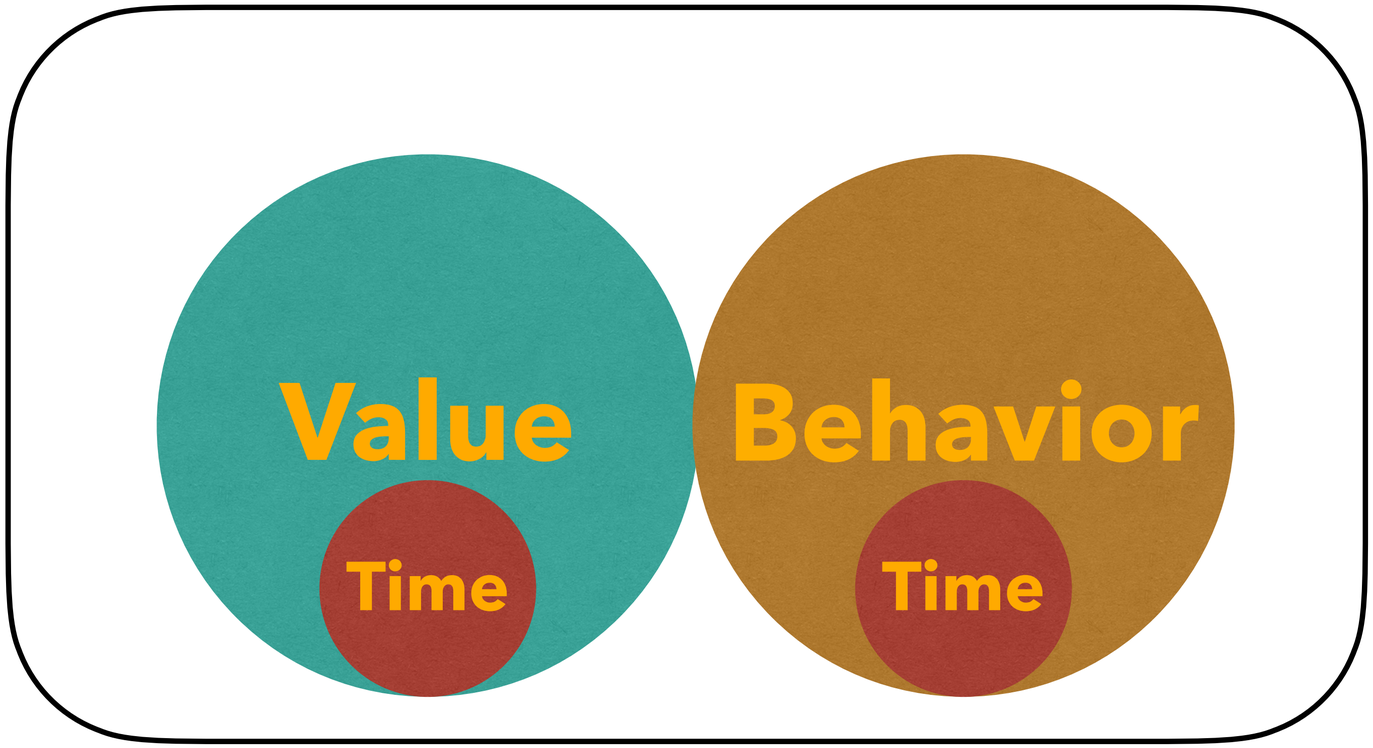


```ruby
conf = "RubyConf Mini"
conf.upcase!
puts conf
# => RUBYCONF MINI
```

A program

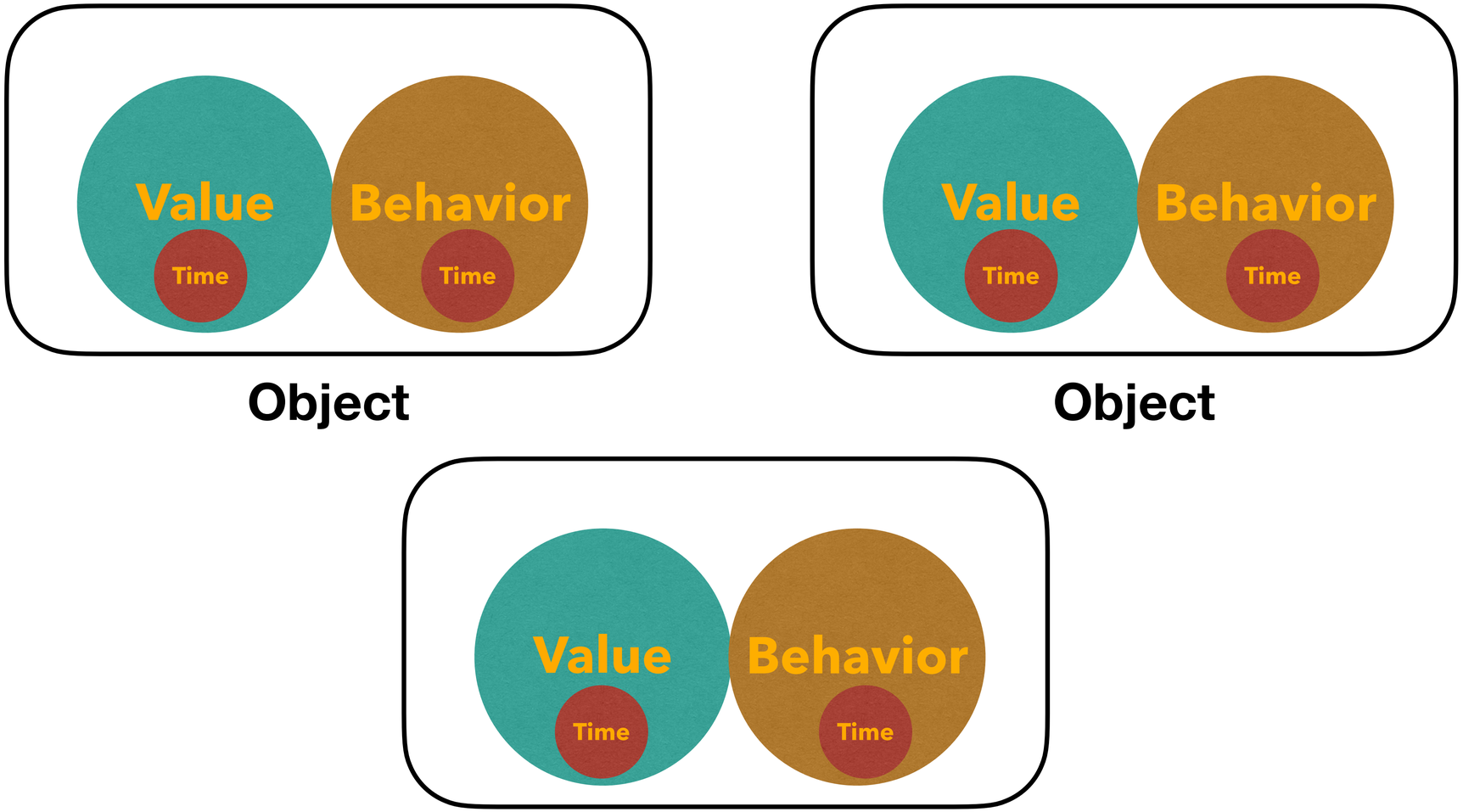


An OO program



Object

An OO program



A FP program

A FP program

Immutability

Immutability

Don’t change a value once you initialize it.

Immutability

```
>> immutable_arr = [1, 2].freeze
=> [1, 2]
>> immutable_arr << 3
(irb):30:in `<main>': can't modify frozen Array: [1, 2] (FrozenError)
```

Immutability

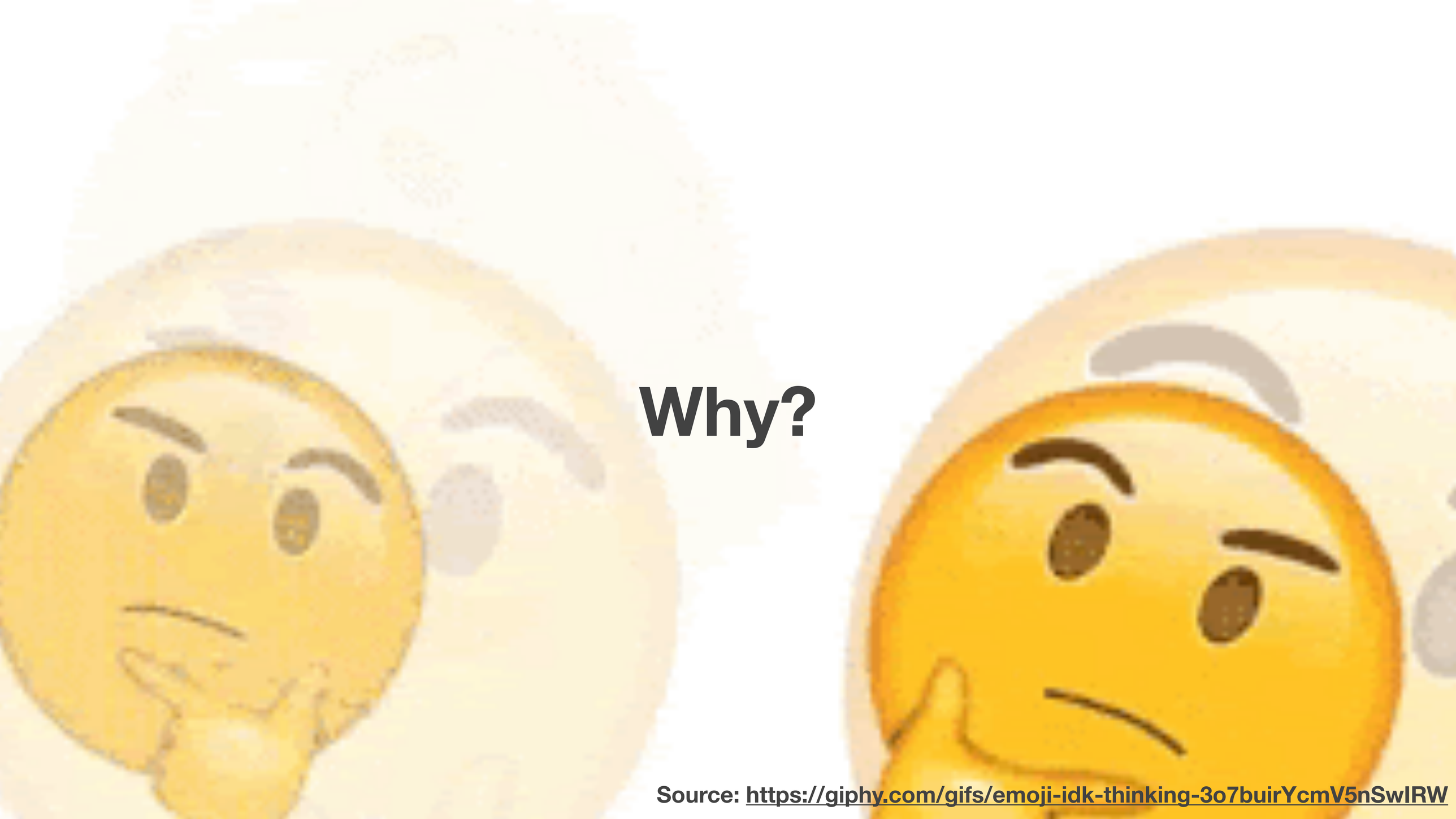
```
class MyObject
  attr_writer
  attr_accessor
  attr_reader
end
```

Immutability

```
class MyObject
  attr_writer
  attr_accessor
  attr_reader
end
```

Immutability

```
class MyObject
  attr_writer
  attr_accessor
  attr_reader
end
```



Why?

Source: https://giphy.com/gifs/emoji-idk-thinking-3o7buirYcmV5nSwIRW

Immutability: Why??

- Peace of mind

Immutability: Why??

```
Flight = Struct.new(:date)
my_flight = Flight.new(Date.parse('2022-11-14'))
friends_flight = my_flight
friends_flight.date = Date.parse('2022-11-13')
puts my_flight.date
#<Date: 2022-11-13 ...>
```

Immutability: Why??

- Peace of mind

Immutability: Why??

- Peace of mind
- Eliminate shared state

Immutability: Why??

- Peace of mind
- Eliminate shared state

Immutability: How???

Immutability: How???

- Provide a good reason for every value mutation.

Immutability: How???

- Provide a good reason for every value mutation.
- Use tools

Introduction

`dry-struct` is a gem built on top of `dry-types` which provides virtus-like DSL for defining typed struct classes.

Basic Usage

You can define struct objects which will have readers for specified attributes using a simple dsl:

```
require 'dry-struct'

module Types
  include Dry.Types()
end

class User < Dry::Struct
  attribute :name, Types::String.optional
  attribute :age, Types::Coercible::Integer
end

user = User.new(name: nil, age: '21')

user.name # nil
user.age # 21
```

Introduction

`dry-struct` is a gem built on top of `dry-types` which provides virtus-like DSL for defining typed struct classes.

Basic Usage

You can define struct objects which will have readers for specified attributes using a simple dsl:

```ruby
require 'dry-struct'

module Types
  include Dry.Types()
end

class User < Dry::Struct
  attribute :name, Types::String.optional
  attribute :age, Types::Coercible::Integer
end

user = User.new(name: nil, age: '21')

user.name # nil
user.age # 21
```

Immutability: How???

- Provide a good reason for every value mutation.
- Use tools
  - `dry-struct`: https://dry-rb.org/gems/dry-struct/main/

Immutability: How???

- Provide a good reason for every value mutation.
- Use tools
  - `dry-struct`: https://dry-rb.org/gems/dry-struct/main/
  - `Data.define` in Ruby `3.2.0-preview3`

Add Data class implementation: Simple immutable value object #6353

Code

Merged **nobu** merged 2 commits into `ruby:master` from `zverok:data-class` on Sep 30

Conversation 28 | Commits 2 | Checks 92 | Files changed 6 | +716 −6

zverok commented on Sep 10 • edited by nobu — Contributor

Source: #16122
Example docs rendering: Data

Design and implementation decisions made:

1. The "define data object method is called `Data.define`. As per Matz:

> `define` might be a candidate. But I still prefer shorter one (e.g. `def`), but you can try to persuade me.

There were a few quite reasonable arguments towards `define` in that ticket. To add to them, my PoV:

- I believe that nowadays (adding new APIs to the mature language), it is better to use full English words to remove confusion and increase readability;
- `def` is strongly associated in Ruby with "defining a method" and became a separate word in Rubyist's dictionary, not a "generic shortcut for 'define'"
- I believe that the "definition of the new type" (unlike the definition of a new method) is a situation where clarity is more important than saving 3 chars; and somewhat rarer.

2. `define` accepts keyword and positional args; they are converted to keyword args there, and checked in `initialize`

Reviewers
- tycooon
- eregon
- nobu

Assignees
No one assigned

Labels
Feature

Milestone
No milestone

Notifications Customize

Add Data class implementation: Simple immutable value object #6353

Code

Merged **nobu** merged 2 commits into `ruby:master` from `zverok:data-class` on Sep 30

Conversation 28 | Commits 2 | Checks 92 | Files changed 6 +716 −6

zverok commented on Sep 10 • edited by nobu Contributor

Source: #16122
Example docs rendering: Data

Design and implementation decisions made:

1. The "define data object method is called `Data.define`. As per Matz:

> `define` might be a candidate. But I still prefer shorter one (e.g. `def`), but you can try to persuade me.

There were a few quite reasonable arguments towards `define` in that ticket. To add to them, my PoV:

- I believe that nowadays (adding new APIs to the mature language), it is better to use full English words to remove confusion and increase readability;
- `def` is strongly associated in Ruby with "defining a method" and became a separate word in Rubyist's dictionary, not a "generic shortcut for 'define'"
- I believe that the "definition of the new type" (unlike the definition of a new method) is a situation where clarity is more important than saving 3 chars; and somewhat rarer.

Reviewers
tycooon
eregon
nobu

Assignees
No one assigned

Labels
Feature

Milestone
No milestone

Notifications Customize

Brandon Weaver

Posted on Oct 6 • Updated on Oct 13

New in Ruby 3.2 - Data.define

Victor Shepelev (Zverok) has just landed an extremely useful feature in Ruby, `Data.define`. You can find the merge here:

https://github.com/ruby/ruby/pull/6353

...and the Ruby discussion here:

https://bugs.ruby-lang.org/issues/16122

So what is it and what does it do? Well that's what we're going to take a look into.

What it Does

Succinctly put `Data.define` creates an immutable `Struct`-like type which can be initialized with either positional or keyword arguments:

Immutability: How???

- Provide a good reason for every value mutation.
- Use tools
  - `dry-struct`: https://dry-rb.org/gems/dry-struct/main/
  - `Data.define` in Ruby `3.2.0-preview3`

A FP program

A FP program

A FP program

Immutable data

Pure functions

A pure function

1. Only depends on its input argument
2. Does not mutate states
   (no side effects)

A pure function

1. **Only depends on its input argument**
2. Does not mutate states
   (no side effects)

A pure function

1. **Only depends on its input argument**
2. Does not mutate states
   (no side effects)

```
# Not pure
def two_days_from_now
  Time.now + 2.days
end
```

A pure function

1. **Only depends on its input argument**
2. Does not mutate states
   (no side effects)

```ruby
# Pure
def two_days_from(timestamp)
  timestamp + 2.days
end
```

A pure function

1. Only depends on its input argument
2. **Does not mutate states (no side effects)**

A pure function

1. Only depends on its input argument
2. **Does not mutate states (no side effects)**

```
# Not pure
def excite!(str)
  str << '!!'
end
```

A pure function

1. Only depends on its input argument
2. **Does not mutate states (no side effects)**

```ruby
# Not pure
def excite!(str)
  str << '!!'
end
```

```ruby
word = 'Functional Programming'
excite!(word)
puts word
# => 'Functional Programming!!'
```

A pure function

1. Only depends on its input argument
2. **Does not mutate states (no side effects)**

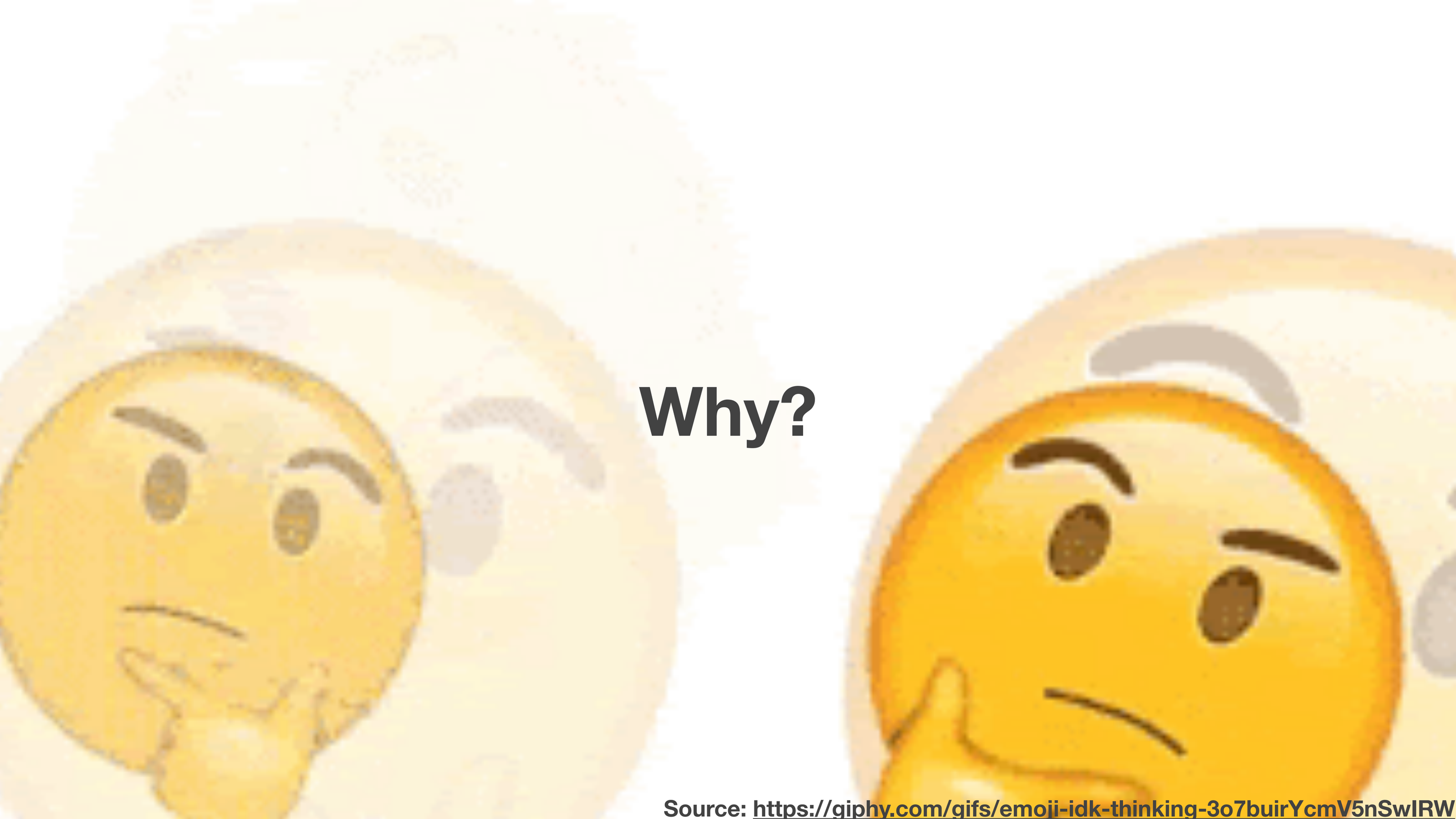
```
# Pure
def excite!(str)
  "#{str}!!"
end
```

A pure function

1. Only depends on its input argument
2. **Does not mutate states (no side effects)**

```ruby
# Pure
def excite!(str)
  "#{str}!!"
end
```

```ruby
word = 'Functional Programming'
excite!(word)
puts word
# => 'Functional Programming'
```



Source: https://giphy.com/gifs/emoji-idk-thinking-3o7buirYcmV5nSwIRW

Pure functions: Why??

“The problem with OO languages is they’ve got all this implicit environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle.”

– Joe Armstrong, Creator of Erlang

Pure functions: Why??

"The problem with OO languages is they've got all this **implicit environment** that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle."

– Joe Armstrong, Creator of Erlang

Pure functions: Why??

No implicit environment!!

Source: https://thenounproject.com/icon/forest-5299391/
Source: https://thenounproject.com/icon/monkey-298282/

Pure functions: Why??

No implicit environment!!

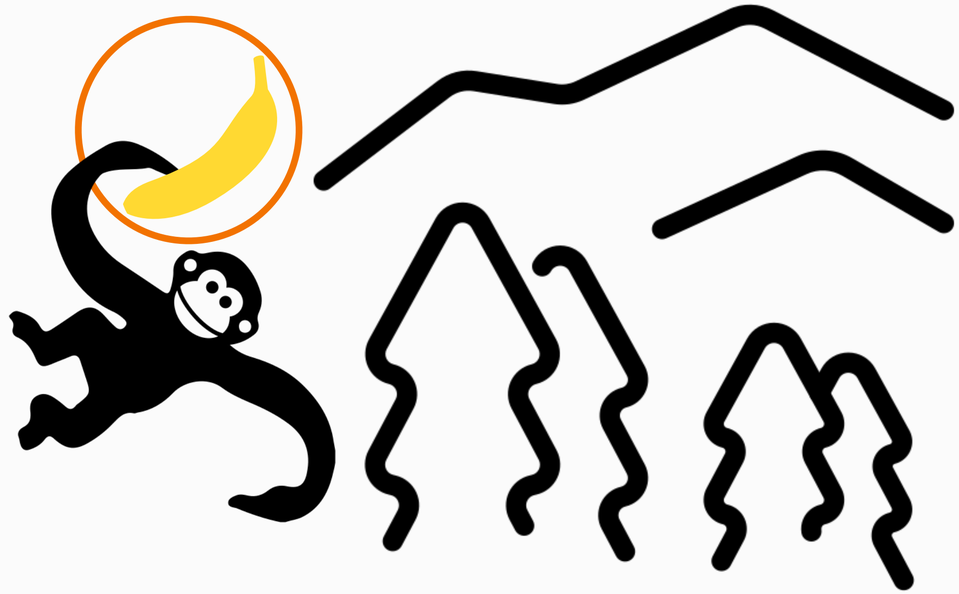
Source: https://thenounproject.com/icon/forest-5299391/
Source: https://thenounproject.com/icon/monkey-298282/

Pure functions: Why??

No implicit environment!!

Pure functions: Why??

No implicit environment!!

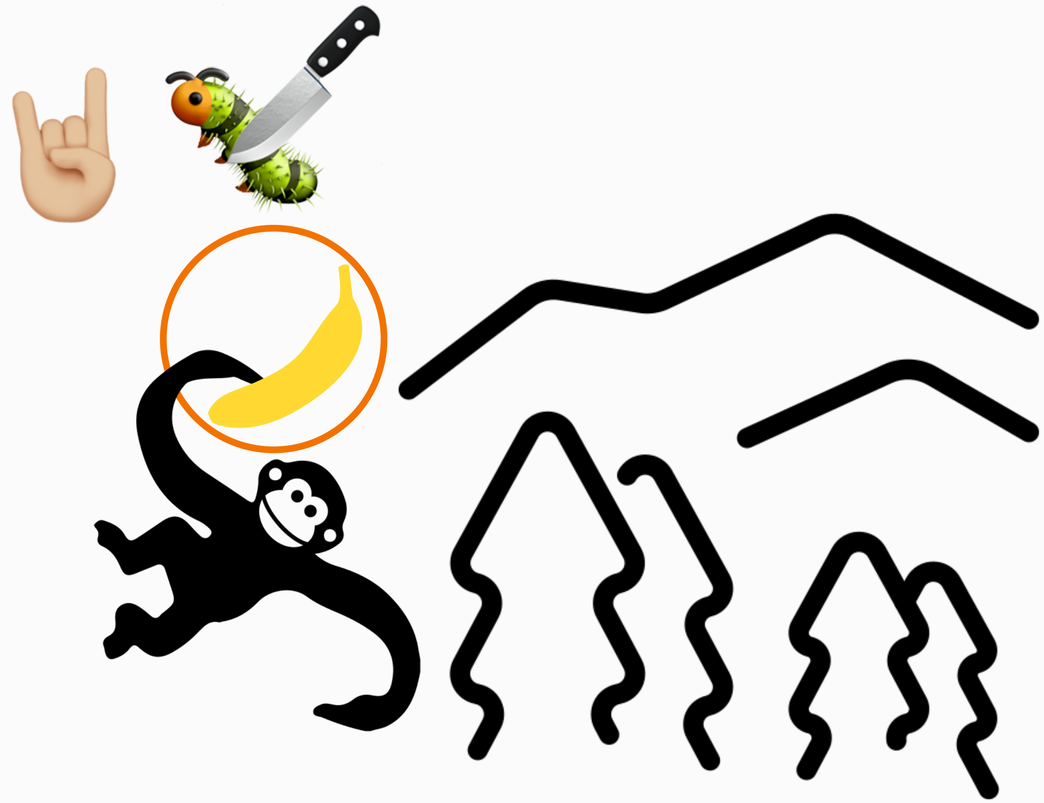


Pure functions: Why??

No implicit environment!!

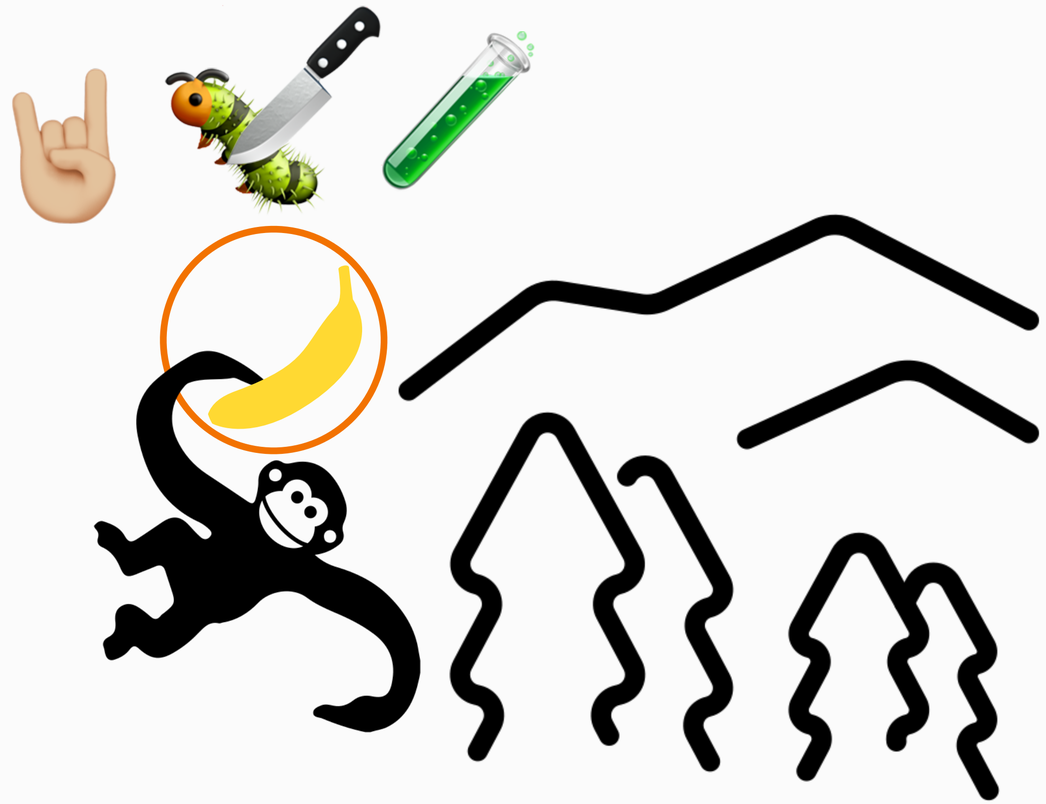
Pure functions: Why??

No implicit environment!!



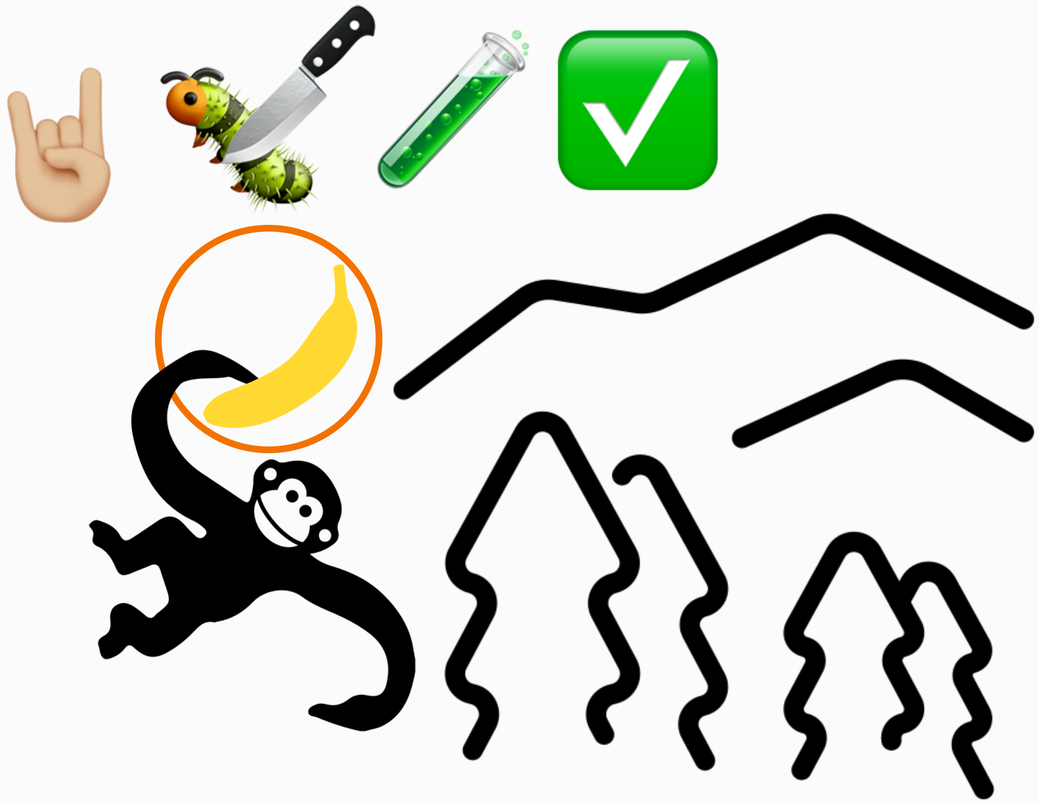
Pure functions: Why??

No implicit environment!!



Pure functions: Why??

No implicit environment!!

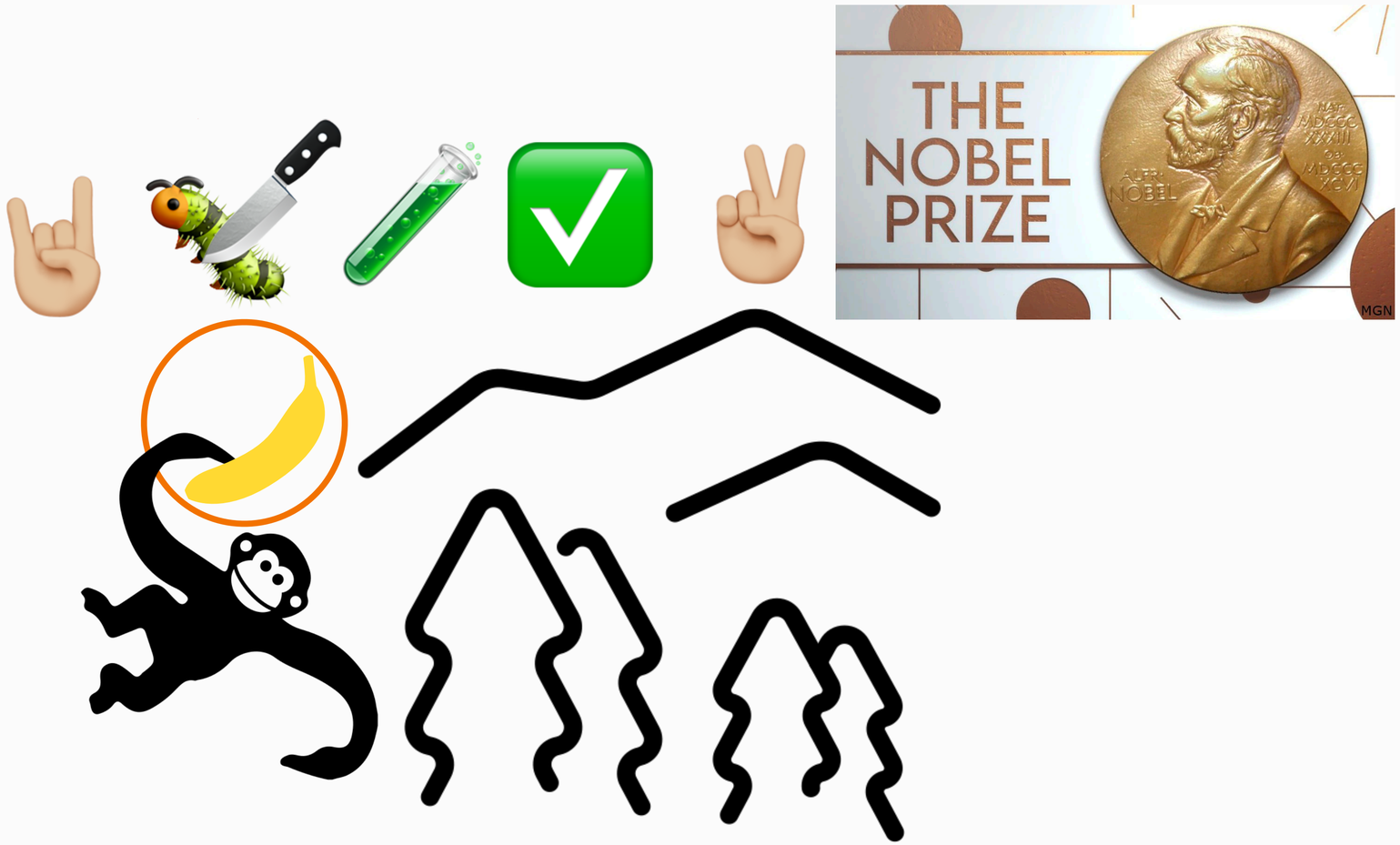


Pure functions: Why??

No implicit environment!!

Pure functions: Why??

No implicit environment!!



Source: https://www.ktiv.com/2022/10/07/nobel-peace-prize-awarded-human-rights-advocates/

Pure functions: How???

Pure functions: How???

- Parameterization/Dependency Injections

Pure functions: Parameterization

```ruby
# Needs to improve
class Newsletter
  def subscribe(email)
    is_valid = EmailFormValidator.new.validate(email)
    is_already_subscribed = @mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    @mailing_list << (email)
    true
  end
end
```

Pure functions: Parameterization

```ruby
# Needs to improve
class Newsletter
  def subscribe(email)
    is_valid = EmailFormValidator.new.validate(email)
    is_already_subscribed = @mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    @mailing_list << (email)
    true
  end
end
```

Pure functions: Parameterization

```ruby
# Needs to improve
class Newsletter
  def subscribe(email)
    is_valid = EmailFormValidator.new.validate(email)
    is_already_subscribed = @mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    @mailing_list << (email)
    true
  end
end
```

Pure functions: Parameterization

```ruby
# Needs to improve
class Newsletter
  def subscribe(email)
    is_valid = EmailFormValidator.new.validate(email)
    is_already_subscribed = @mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    @mailing_list << (email)
    true
  end
end
```

Pure functions: Parameterization

```ruby
# Needs to improve
class Newsletter
  def subscribe(email)
    is_valid = EmailFormValidator.new.validate(email)
    is_already_subscribed = @mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    @mailing_list << (email)
    true
  end
end
```

Pure functions: Parameterization

```ruby
# Needs to improve
class Newsletter
  def subscribe(email)
    is_valid = EmailFormValidator.new.validate(email)
    is_already_subscribed = @mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    @mailing_list << (email)
    true
  end
end
```

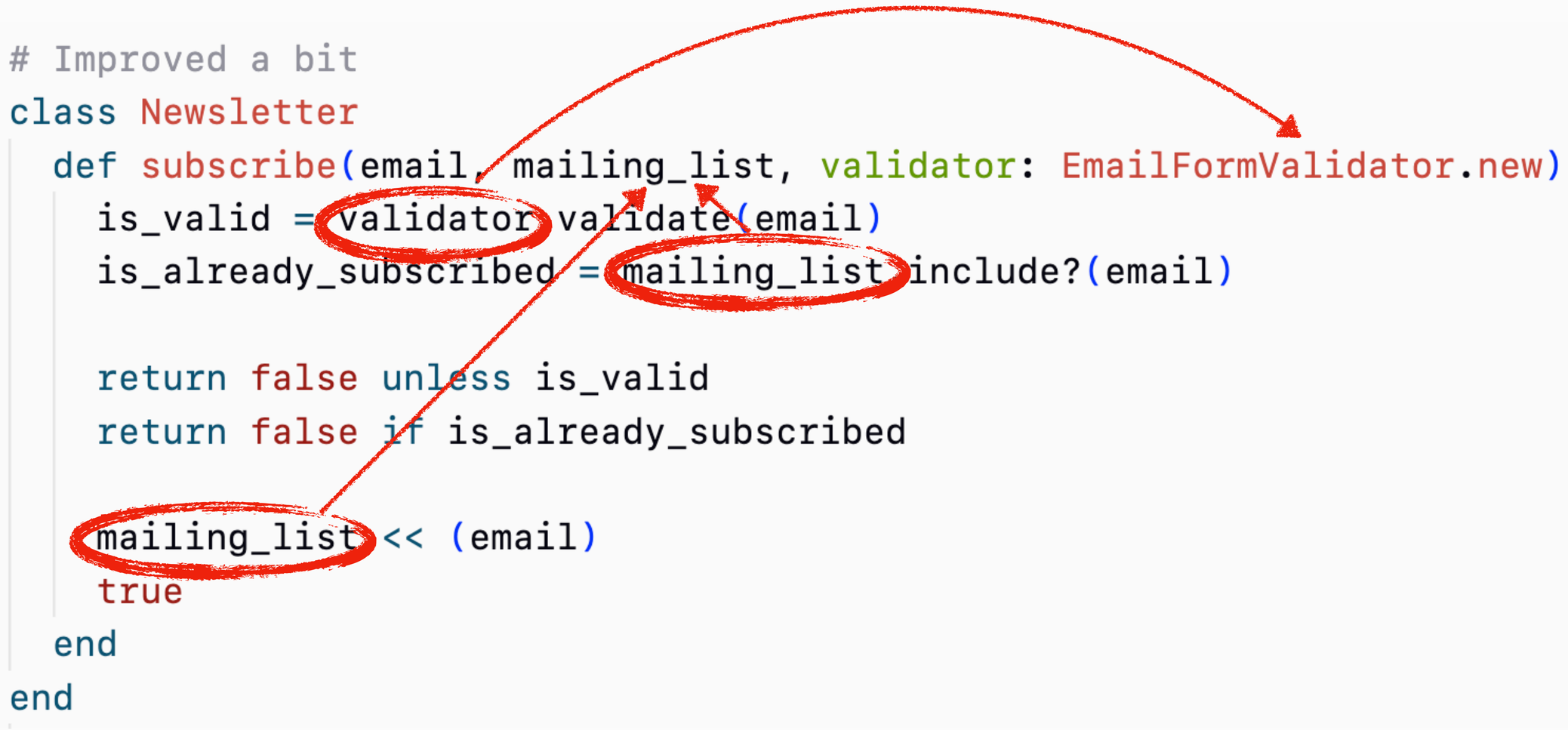
Pure functions: Parameterization

```ruby
# Needs to improve
class Newsletter
  def subscribe(email)
    is_valid = EmailFormValidator.new.validate(email)
    is_already_subscribed = @mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    @mailing_list << (email)
    true
  end
end
```

Pure functions: Parameterization



```ruby
# Improved a bit
class Newsletter
  def subscribe(email, mailing_list, validator: EmailFormValidator.new)
    is_valid = validator.validate(email)
    is_already_subscribed = mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    mailing_list << (email)
    true
  end
end
```

Pure functions: Parameterization

```ruby
# Improved a bit
class Newsletter
  def subscribe(email, mailing_list, validator: EmailFormValidator.new)
    is_valid = validator.validate(email)
    is_already_subscribed = mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    mailing_list << (email)
    true
  end
end
```

Pure functions: Parameterization

```ruby
# Improved a bit
class Newsletter
  def subscribe(email, mailing_list, validator: EmailFormValidator.new)
    is_valid = validator.validate(email)
    is_already_subscribed = mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    mailing_list << (email)
    true
  end
end
```

Pure functions: How???

- Parameterization/Dependency Injections
- **Transformation instead of mutation**

Pure functions: Transformation over Mutation

```ruby
# Improved a bit
class Newsletter
  def subscribe(email, mailing_list, validator: EmailFormValidator.new)
    is_valid = validator.validate(email)
    is_already_subscribed = mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    mailing_list << (email)
    true
  end
end
```

Pure functions: Transformation over Mutation

```ruby
# Improved a bit
class Newsletter
  def subscribe(email, mailing_list, validator: EmailFormValidator.new)
    is_valid = validator.validate(email)
    is_already_subscribed = mailing_list.include?(email)

    return false unless is_valid
    return false if is_already_subscribed

    mailing_list << (email) [*mailing_list, email]
    true
  end
end
```

@jenny_codes

Pure functions: Transformation over Mutation

```ruby
# Improved more
class Newsletter
  def subscribe(email, mailing_list, validator: EmailFormValidator.new)
    is_valid = validator.validate(email)
    is_already_subscribed = mailing_list.include?(email)

    return mailing_list unless is_valid
    return mailing_list if is_already_subscribed

    [*mailing_list, email]
  end
end
```

A pure function

1. Only depends on its input argument
2. Does not mutate states
   (no side effects)

```ruby
# Improved more
class Newsletter
  def subscribe(email, mailing_list, validator: EmailFormValidator.new)
    is_valid = validator.validate(email)
    is_already_subscribed = mailing_list.include?(email)

    return mailing_list unless is_valid
    return mailing_list if is_already_subscribed

    [*mailing_list, email]
  end
end
```

A pure function

1. **Only depends on its input argument ✅**
2. Does not mutate states
   (no side effects)

```ruby
# Improved more
class Newsletter
  def subscribe(email, mailing_list, validator: EmailFormValidator.new)
    is_valid = validator.validate(email)
    is_already_subscribed = mailing_list.include?(email)

    return mailing_list unless is_valid
    return mailing_list if is_already_subscribed

    [*mailing_list, email]
  end
end
```

A pure function

1. Only depends on its input argument ✅
2. **Does not mutate states ✅**
   (no side effects)

```ruby
# Improved more
class Newsletter
  def subscribe(email, mailing_list, validator: EmailFormValidator.new)
    is_valid = validator.validate(email)
    is_already_subscribed = mailing_list.include?(email)

    return mailing_list unless is_valid
    return mailing_list if is_already_subscribed

    [*mailing_list, email]
  end
end
```

A FP program

Immutable data

A FP program

Immutable data

Pure functions

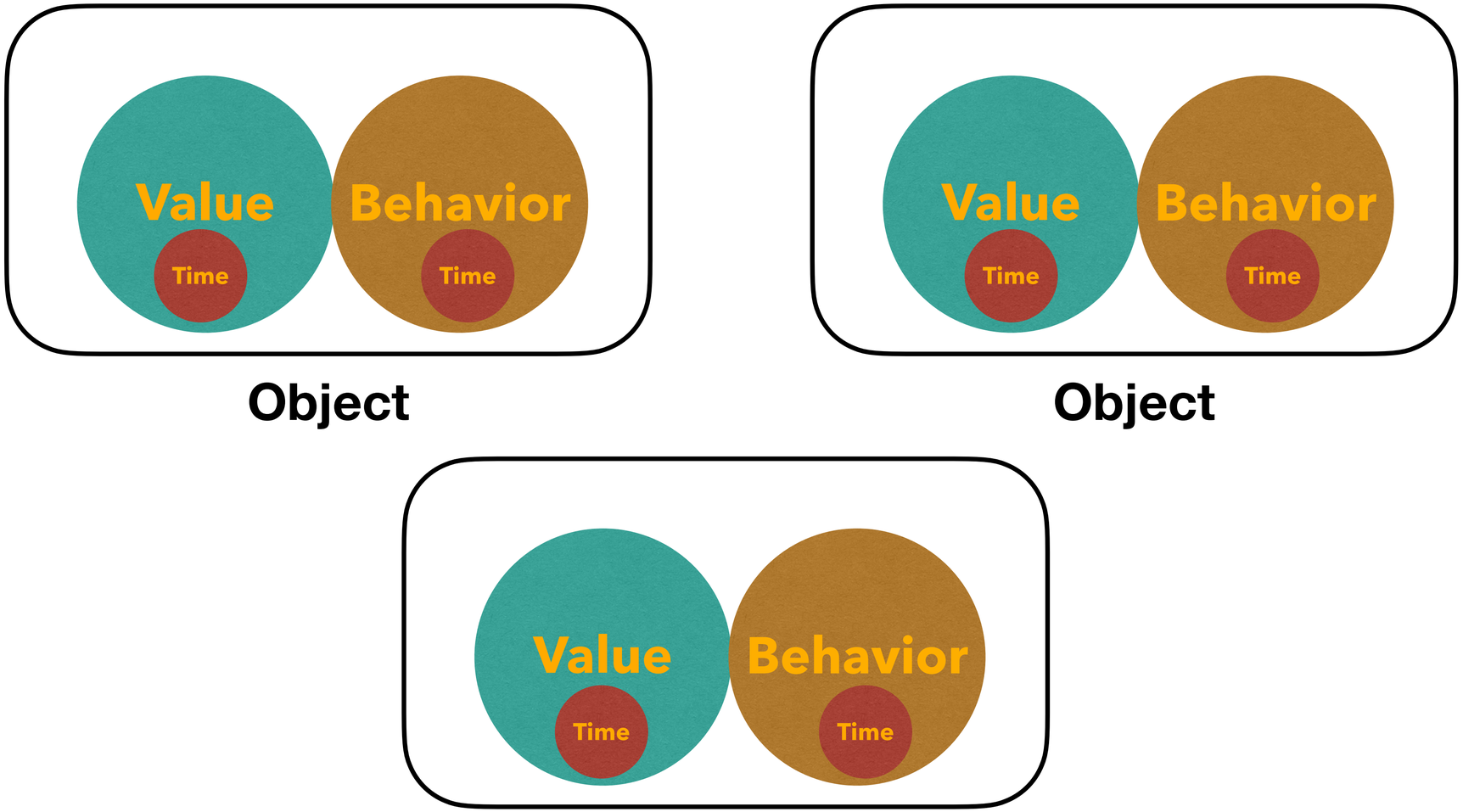
A FP program

Immutable data

Pure functions

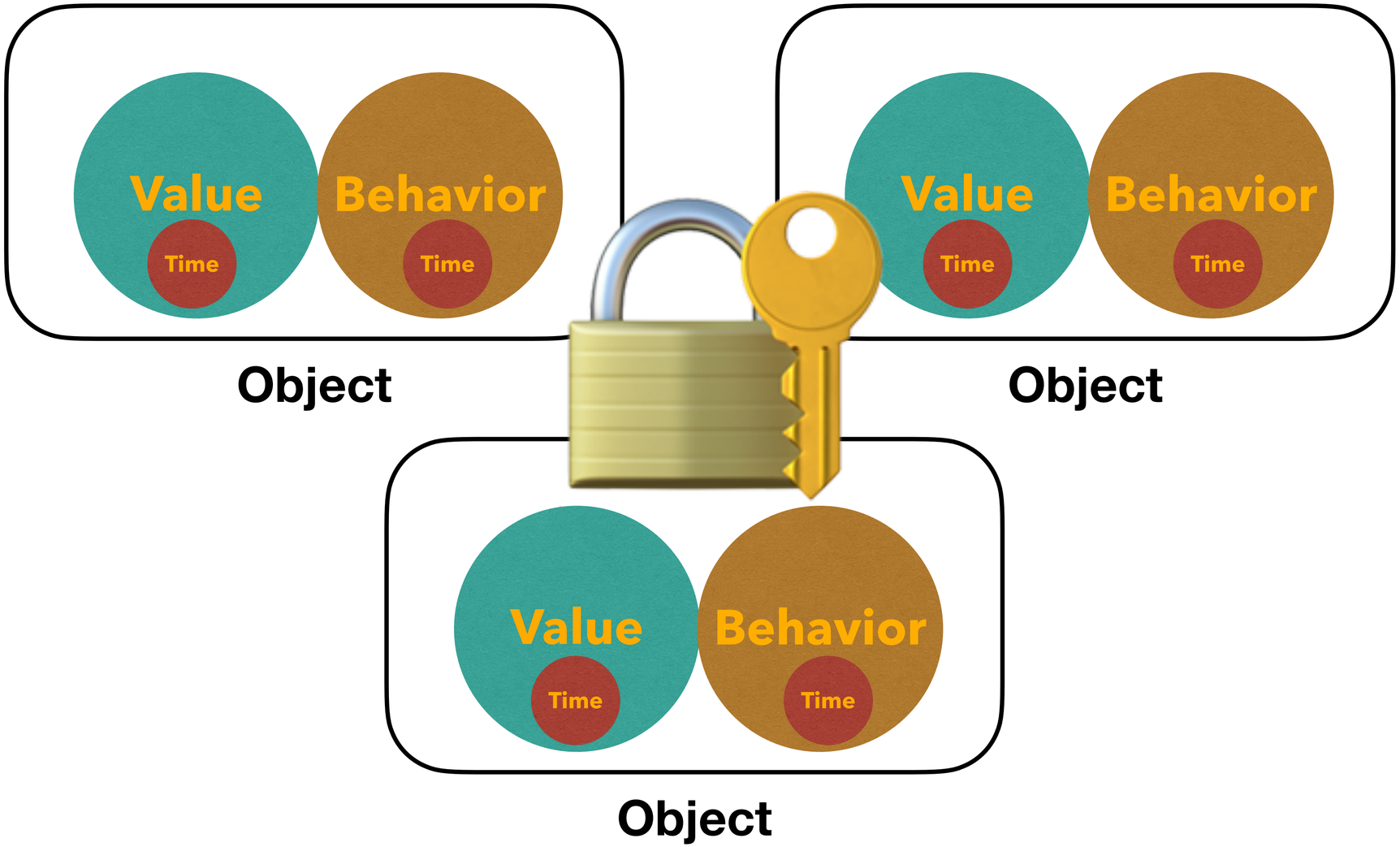
Concurrency

An OO program



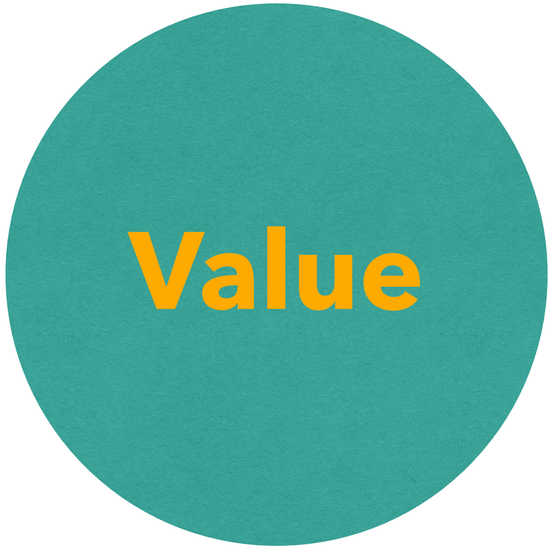
An OO program

Global Interpreter Lock



A FP program

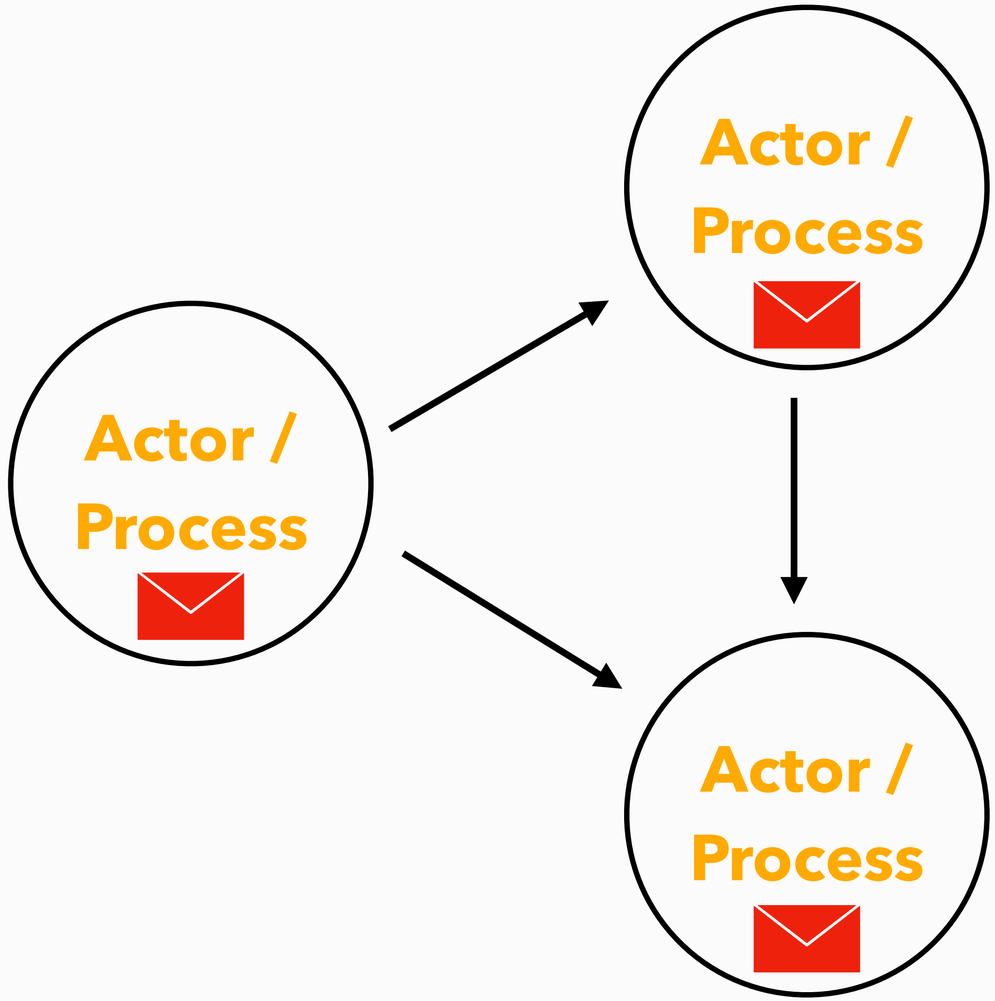
Immutable data



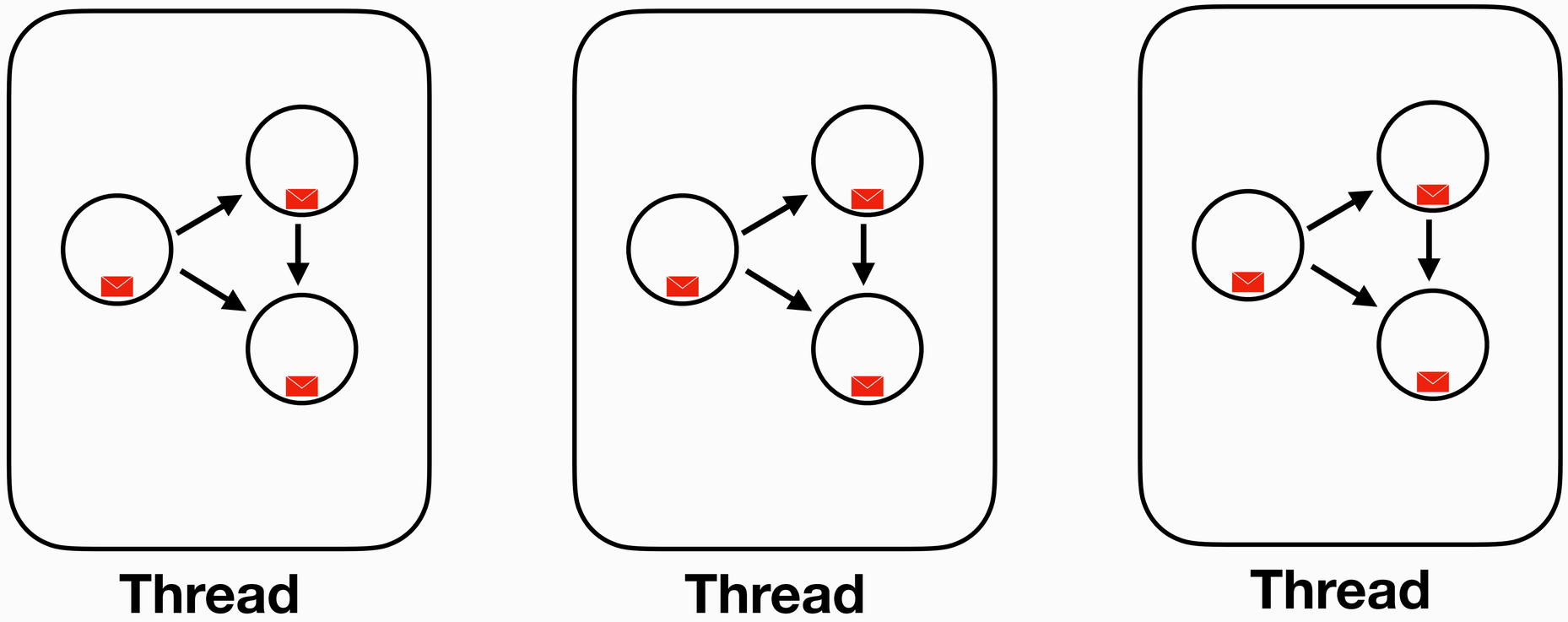
Pure functions

Concurrency 💯

Elixir's concurrency model: Actors



Elixir's concurrency model: Actors



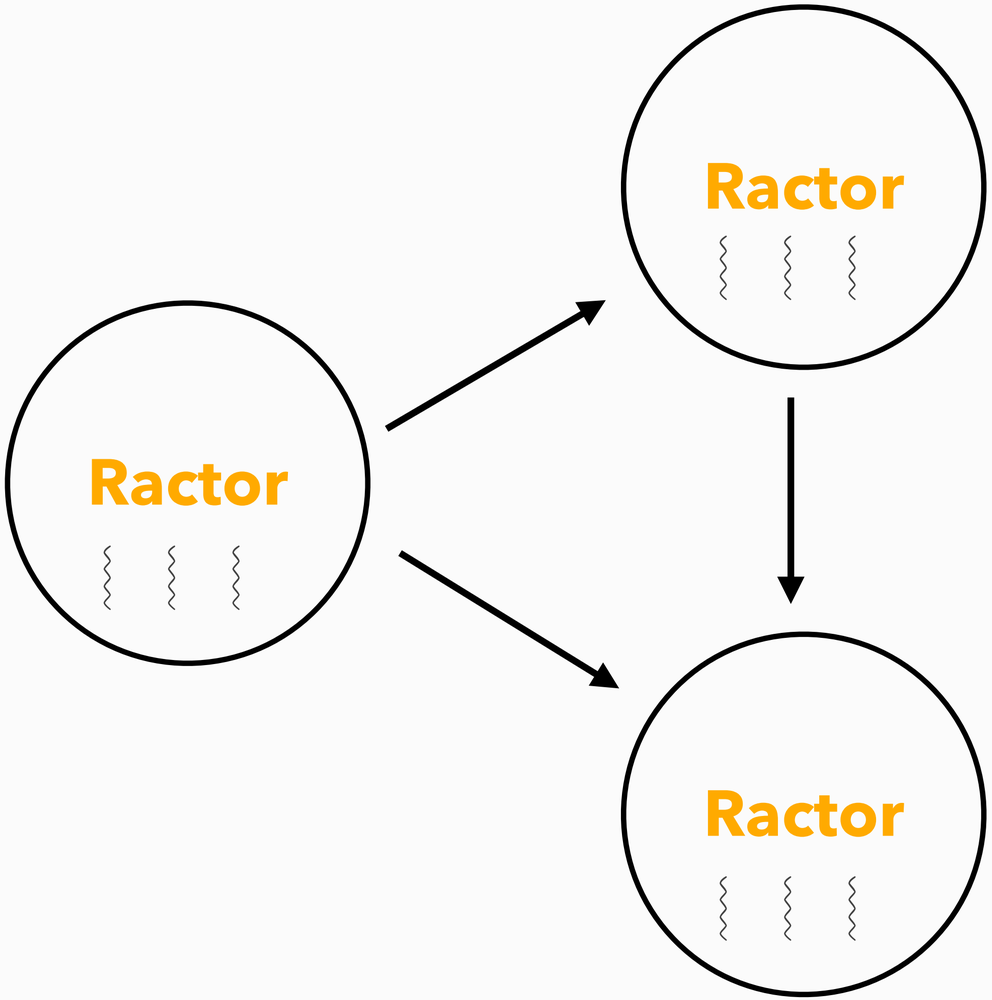
Ruby's concurrency abstraction: Ractor

"It's multi-core age today. Concurrency is very important. With Ractor, along with Async Fiber, Ruby will be a real concurrent language."

– Matz

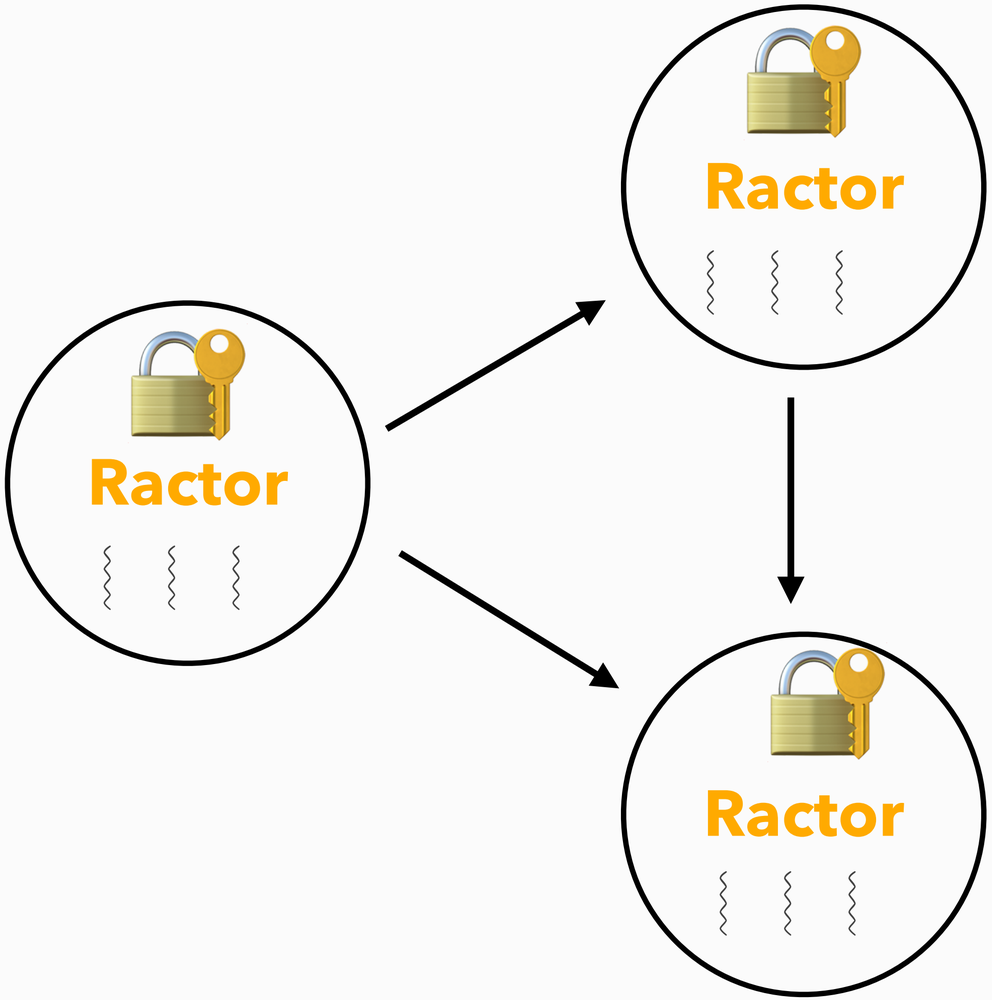
https://www.ruby-lang.org/en/news/2020/12/25/ruby-3-0-0-released/

Ruby's concurrency abstraction: Ractor



= threads

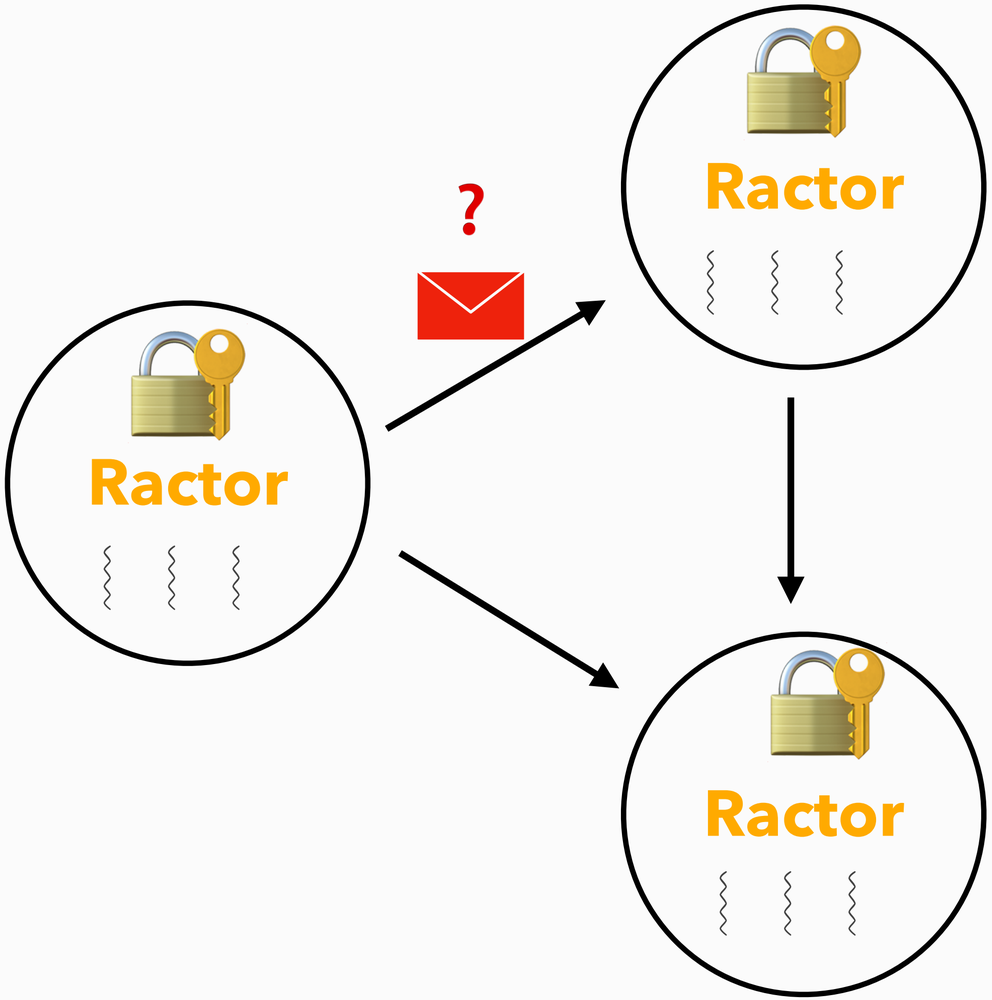
Ruby's concurrency abstraction: Ractor



= threads

= Global Interpreter Lock

Ruby's concurrency abstraction: Ractor



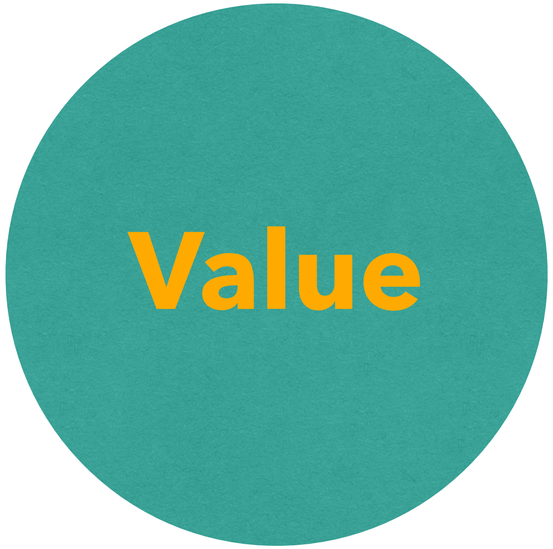
= threads

= Global Interpretation Lock

```
>> RUBY_VERSION
=> "3.2.0"
>> Ractor.new { puts "Hello Ractor" }
<internal:ractor>:267: warning: Ractor is experimental,
 and the behavior may change in future versions of Ruby
! Also there are many implementation issues.
Hello Ractor
```

A FP program

Immutable data



Pure functions

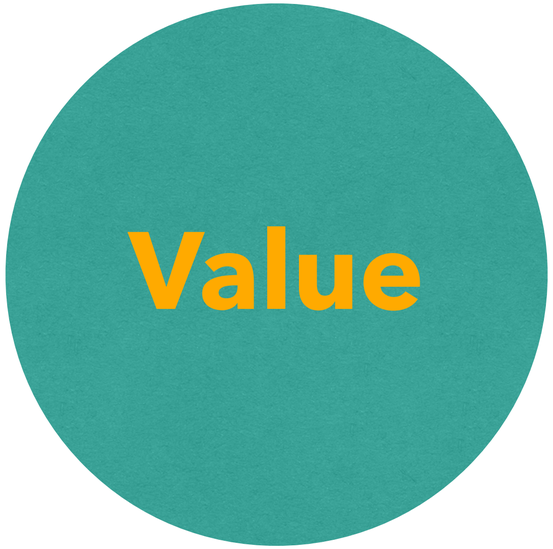
Concurrency

Recap

We are defined not so much by what we can do, but what we don’t do.

A FP program

Immutable data

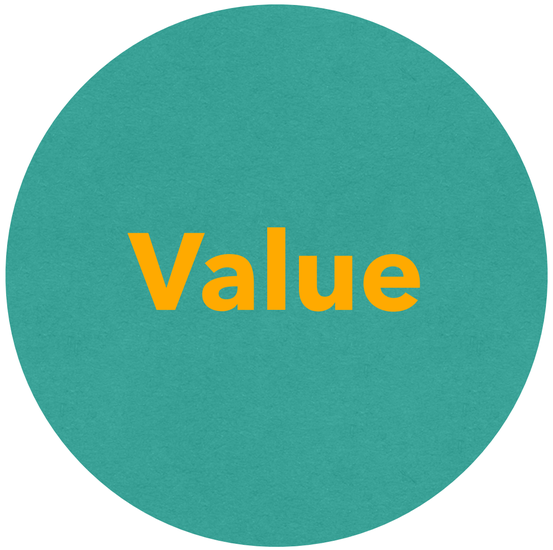


Pure functions

Concurrency

A FP program

Immutable data



Pure functions

Concurrency

A FP program

Immutable data

Pure functions

Concurrency

A FP program

Immutable data

Pure functions

Concurrency

Functional programming helps us write better Ruby code

Resources: Functional programming mental model

- Jose Valim's ElixirConf 2017 Keynote by Jose Valim
- Are we there yet by Rich Hickey
- The language of programming by Anjana Vakil
- Value Object by Martin Fowler

Resources: Functional programming architecture

(That I didn't get to talk about because I'd get too excited and run overtime)

- Functional Architecture for the Practical Rubyist by Tim Riley
- Functional web with Ruby by Luca Luigi
- Boundaries by Gary Bernhardt
- The Hanami Ruby framework
- (Book) Domain Modelling Made Functional by Scott Wlaschin

Resources: Ruby docs

- https://ruby-doc.org/core-3.2.0_preview2/Ractor.html
- https://scoutapm.com/blog/ruby-ractor
- https://dev.to/baweaver/new-in-ruby-32-datadefine-2819
- https://github.com/ruby/ruby/pull/6353