ARCHITECTURE
THE RECLAIMED YEARS

PIOTR SOLNICA

- ROM-RB CREATOR
- DRY-RB CO-FOUNDER
- GITHUB.COM/SOLNIC
  - @_SOLNIC_
    - SOLNIC.EU

ARCHITECTURE
THE LOST YEARS

"THE WEB IS A DELIVERY MECHANISM.
THE WEB IS A DETAIL"

- UNCLE BOB

"THE TOP LEVEL ARCHITECTURE OF MY RAILS APPLICATION, DID NOT SCREAM ITS INTENT AT YOU, IT SCREAMED THE FRAMEWORK AT YOU, IT SCREAMED RAILS AT YOU"

– UNCLE BOB

"IT'S GOOD FOR DHH, NOT SO GOOD FOR YOU"

- UNCLE BOB

"DATABASE IS A DETAIL"

- UNCLE BOB

RAILS IS YOUR ARCHITECTURE
EMBRACE IT OR LEAVE IT

FAST TESTS ⚡

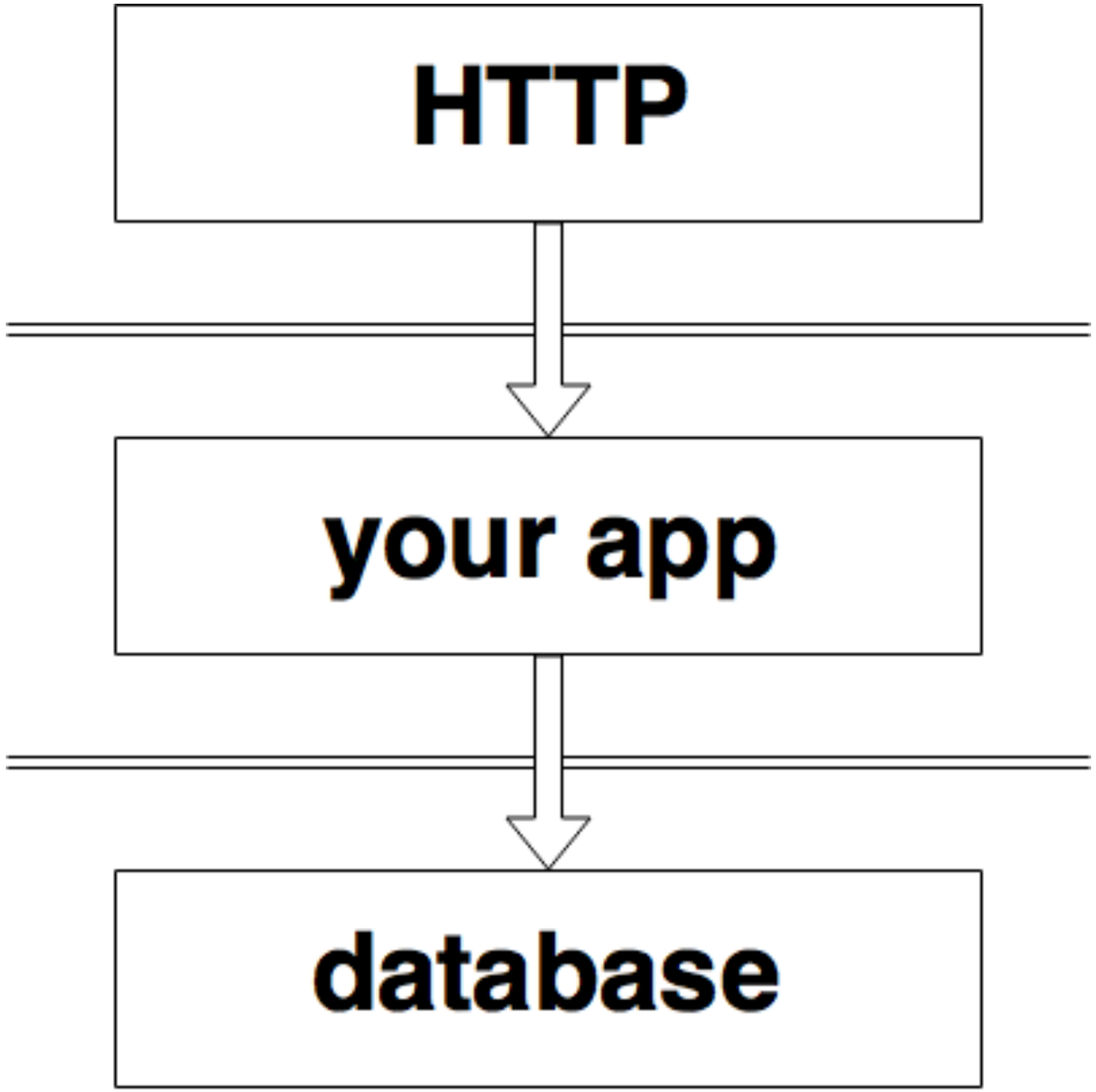
- BOUNDARIES
- DATA STRUCTURES
- DEPENDENCIES

BOUNDARIES

HTTP

your app

database

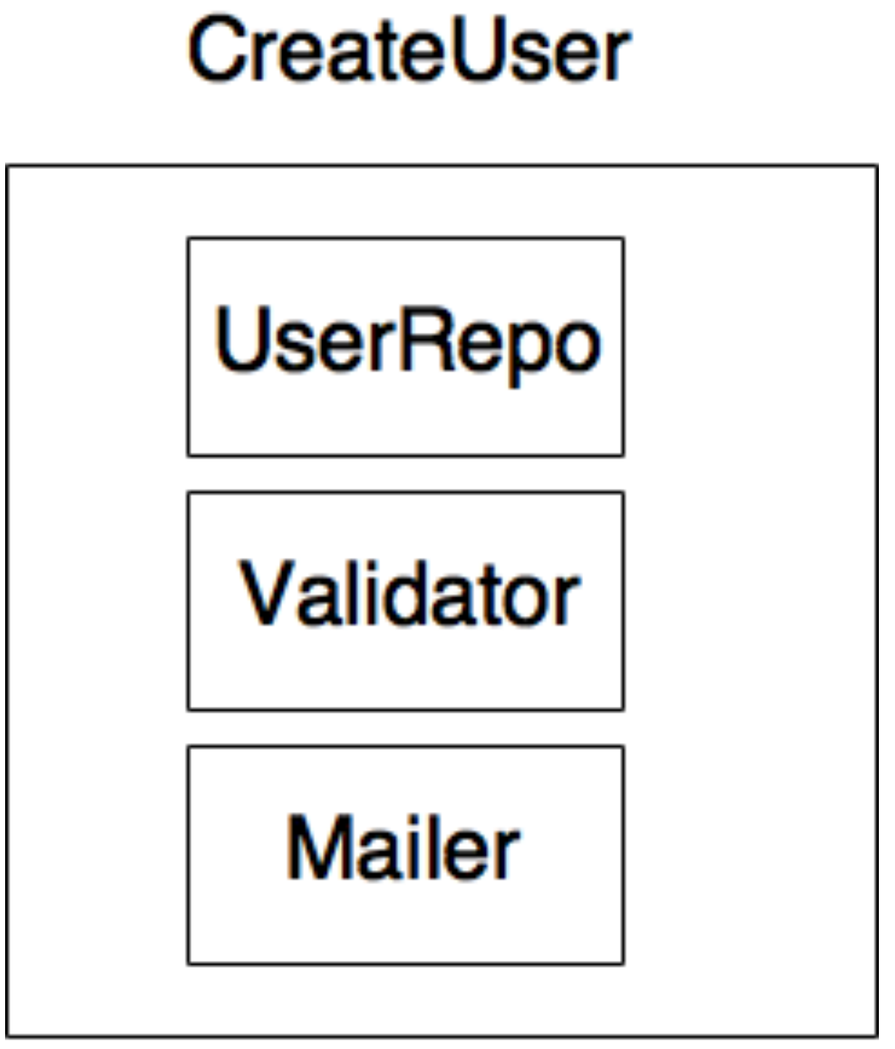

HTTP
your app
database

DATA STRUCTURES

- HTTP REQUEST
- APPLICATION RESPONSE
- VIEW-SPECIFIC DATA
- DOMAIN-SPECIFIC DATA

DEPENDENCIES

CreateUser


CreateUser
UserRepo
Validator
Mailer

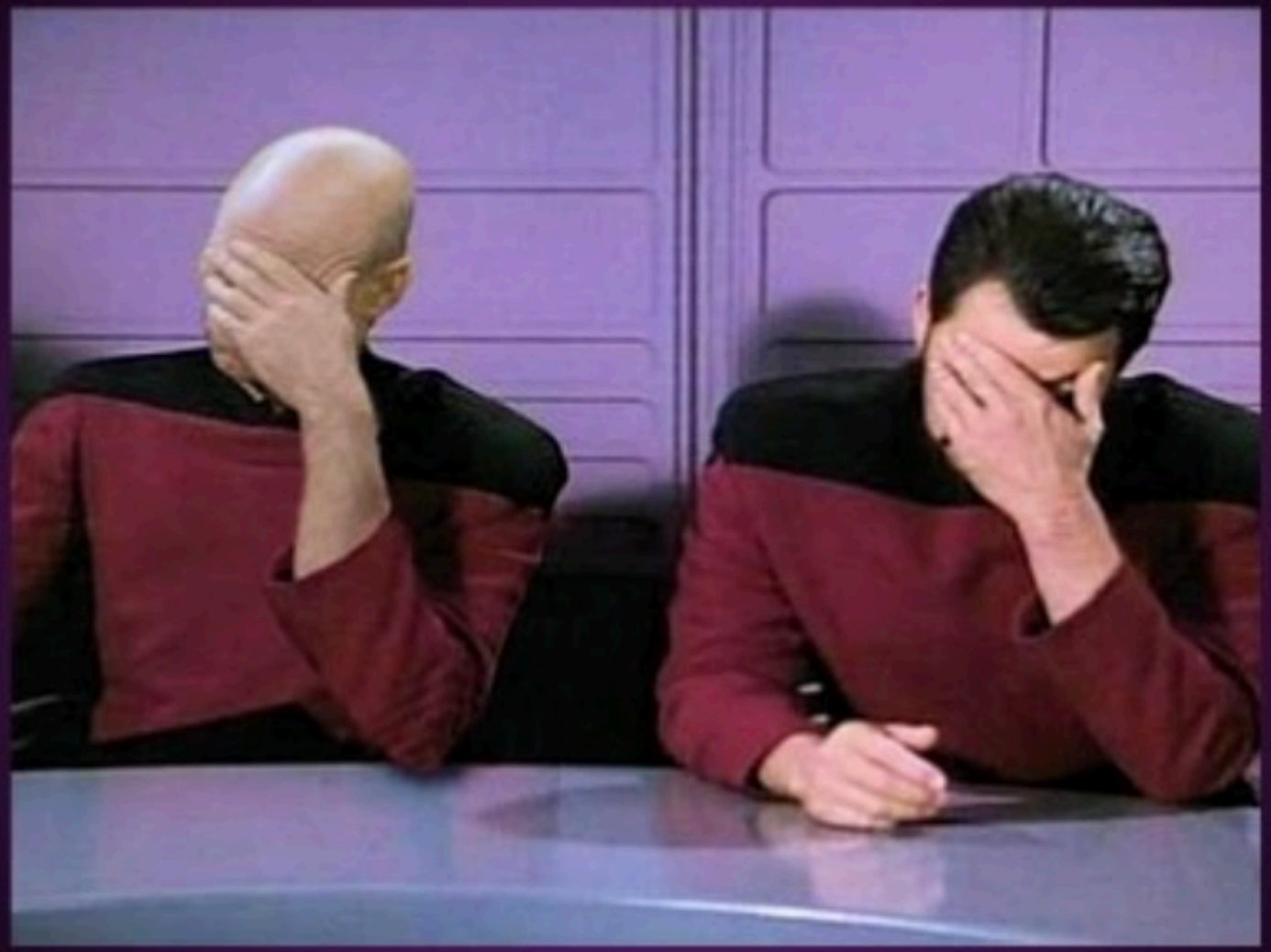
```
class CreateUser

end
```

```ruby
class CreateUser
  attr_reader :user_repo, :validator, :mailer

  def initialize(user_repo:, validator:, mailer:)
    @user_repo = user_repo
    @validator = validator
    @mailer = mailer
  end
end
```

- CLASSES
- MODULES
- SINGLETON METHODS

PROBLEM WITH CLASSES

CLASSES IN RUBY ARE GLOBAL, STATEFUL, MUTABLE VARIABLES



- MINIMIZE STATE IN CLASSES
- DON'T RELY ON CLASS STATE AT RUNTIME
- DON'T RELY ON MONKEY-PATCHING

PROBLEM WITH MODULES

MODULES IN RUBY IS A FORM OF MULTIPLE INHERITANCE

IT'S HARD TO ACHIEVE A COHERENT SYSTEM WHEN MODULES ARE USED EXTENSIVELY

FAVOR COMPOSITION OVER INHERITANCE

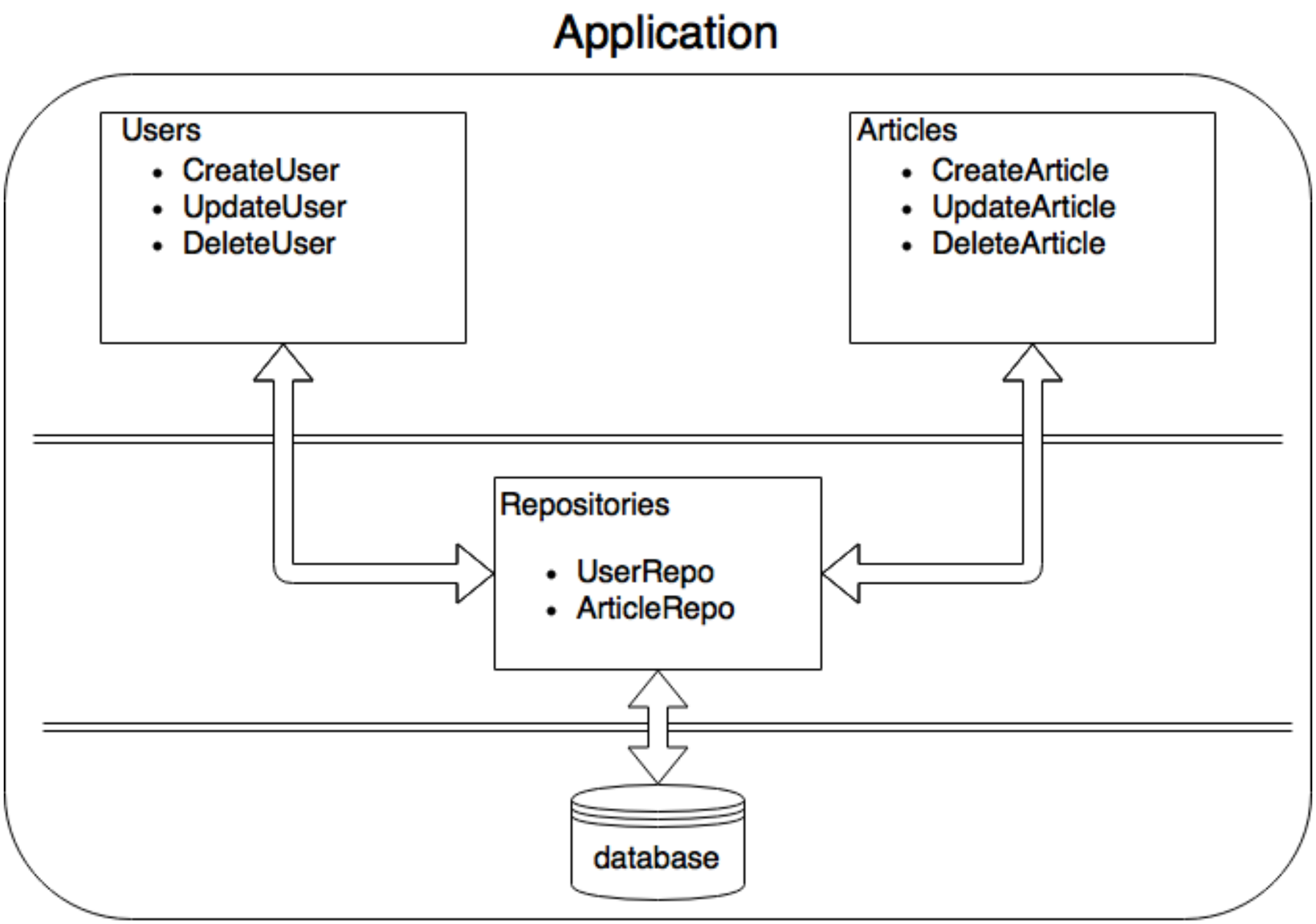
SINGLETON METHODS

- USING SINGLETON METHODS COUPLE YOUR CODE TO CLASS/ MODULE CONSTANTS
- SINGLETON METHODS EASILY LEAD TO AWKWARD, PROCEDURAL CODE

- MINIMIZE USAGE OF SINGLETON METHODS. THEY ARE ONLY GOOD AS "BUILDER" METHODS, OR TOP-LEVEL CONFIGURATION APIS
- DON'T USE THEM AT RUNTIME. OBJECTS ARE 10 X BETTER AND MORE FLEXIBLE

DRY-SYSTEM

- AN ARCHITECTURE FOR RUBY APPLICATIONS
- BASED HEAVILY ON LIGHTWEIGHT DEPENDENCY INJECTION
- ALLOWS YOU TO COMPOSE AN APPLICATION FROM ISOLATED COMPONENTS


Application
Users
CreateUser
UpdateUser
DeleteUser
Articles
CreateArticle
UpdateArticle
DeleteArticle
Repositories
UserRepo
ArticleRepo
database

RUBY APPLICATION COMES FIRST

```
app
  |-lib
  |-system
  |-spec
```

```
app
  |-lib
  |-system
    |- app.rb <= your app
    |- import.rb <= DI extension
  |-spec
```

```
app
  |-lib
    |- users/create_user.rb
    |- repos/user_repo.rb
  |-system
  |-spec
```

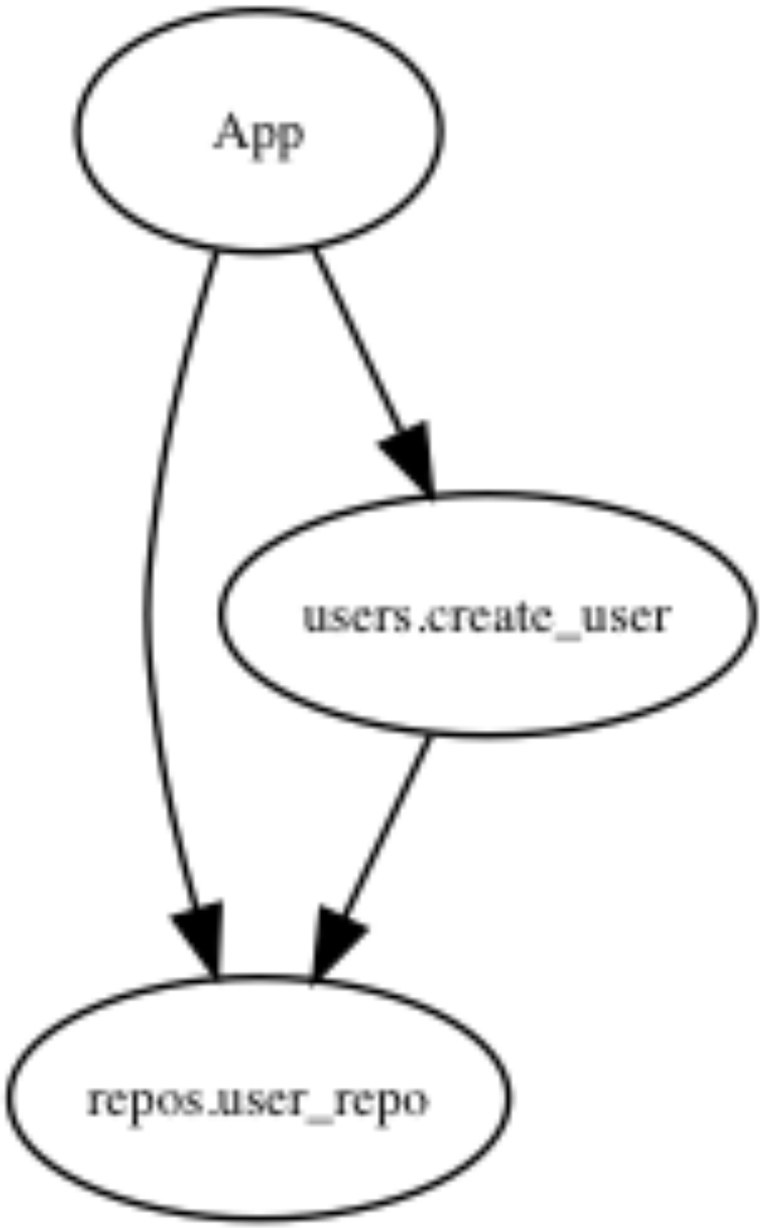
```ruby
# app/system/app.rb

require 'dry/system/container'

class App < Dry::System::Container
  configure do |config|
    config.auto_register = %w(lib)
  end

  load_paths! 'lib', 'system'
end
```

SIMPLE OBJECT COMPOSITION

```
require 'import'

module Users
  class CreateUser
    include Import['repos.user_repo']

    def call(params)
      user_repo.create(params)
    end
  end
end
```

YOUR APP IS THE ENTRY POINT TO YOUR SYSTEM

```
∞ pry -r ./system/app

[1] pry(main)> App['users.create_user']
=> #<Users::CreateUser:0x00007ff3a1b2e520..>

[2] pry(main)> App['repos.user_repo']
=> #<Repos::UserRepo:0x00007ff3a11b0890..>
```


App
users.create_user
repos.user_repo

TESTING IN ISOLATION

```ruby
require 'users/create_user'

RSpec.describe Users::CreateUser do
  subject(:create_user) do
    Users::CreateUser.new
  end

  describe '#call' do
    it 'returns created user' do
      user = create_user.call(id: 1, name: 'Jane')

      expect(user).to eql(id: 1, name: 'Jane')
    end
  end
end
```

USER INTERFACE AS AN EXTENSION

- WEB UI BASED ON HTML/CSS/JS
- JSON API
- CLI INTERFACE
- ...

LET'S ADD A WEB INTERFACE ON TOP USING RODA

```ruby
# system/web.rb

require_relative 'app'
require 'roda'

class Web < Roda
  opts[:api] = App

  plugin :json

  route do |r|
    r.post 'users' do
      api['users.create_user'].call(r[:user])
    end
  end

  def api
    self.class.opts[:api]
  end
end
```

```
∞ curl -X POST http://localhost:9292/users -d "user[id]=1&user[name]=Jane"
{"id":"1","name":"Jane"}
```

LET'S ADD A CLI ON TOP USING HANAMI-CLI

```ruby
#!/usr/bin/env ruby

require "bundler/setup"
require "hanami/cli"
require "json"
require_relative '../system/boot'

module Commands
  extend Hanami::CLI::Registry

  class CreateUser < Command
    desc "Creates a user"

    argument :user, desc: "User data"

    def call(user: nil, **)
      params = JSON.parse(user)
      output = App['users.create_user'].call(params)
      puts "Created #{output.inspect}"
    end
  end

  register "create_user", CreateUser
end

Hanami::CLI.new(Commands).call
```

```
∞ bin/app create_user '{"id":1,"name":"Jane"}'
Created {"id"=>1, "name"=>"Jane"}
```

DID YOU NOTICE THE BOUNDARIES HERE?

WEB INPUT AS PRE-PROCESSED RACK PARAMS

```
r[:user] # { "id" => 1, "name" => "Jane" }
```

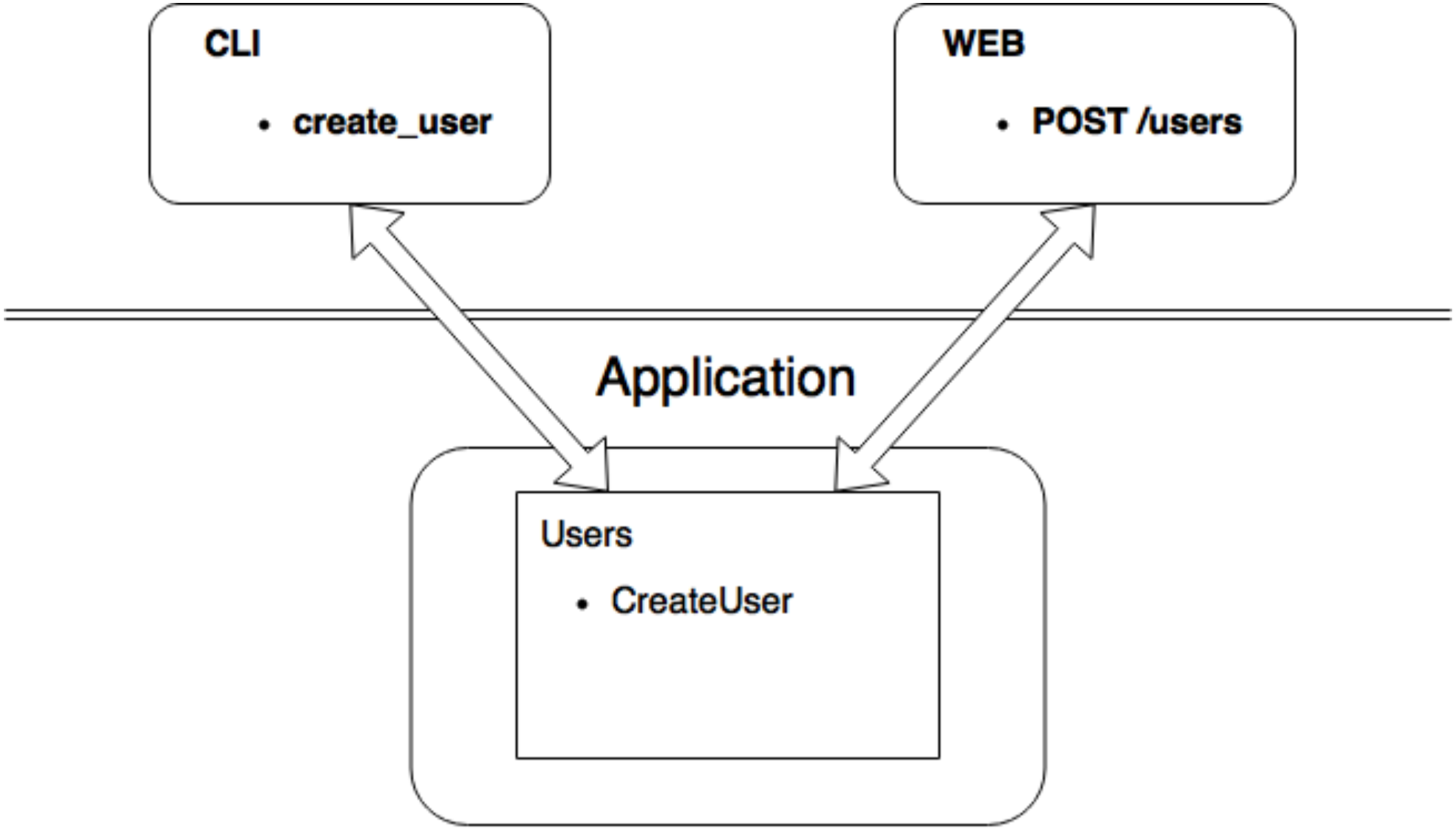
CLI INPUT AS A PLAIN JSON STRING

```
'{"id":1,"name":"Jane"}'
```

THIS IS NOT A CONCERN **OF YOUR APPLICATION**

YOUR APPLICATION IS AN API WITH OBJECTS ACCEPTING SPECIFIC DATA STRUCTURES AS INPUT

```
# user creation end-point
App['users.create_user']

# expected data structure schema
{ id: Integer, name: String }
```


CLI
• create_user
WEB
• POST /users
Application
Users
• CreateUser

CLEAN ARCHITECTURE

- RUBY APP COMES FIRST
- RESPECTING BOUNDARIES
- OBJECT COMPOSITION
- USER INTERFACE AS AN EXTENSION OF YOUR RUBY APP

THANK YOU

MORE THINGS TO CHECK OUT

- BOUNDARIES TALK BY GARY BERNHARDT
  - DRY-SYSTEM ON GITHUB
- SAMPLE APP FROM SLIDES ON GITHUB