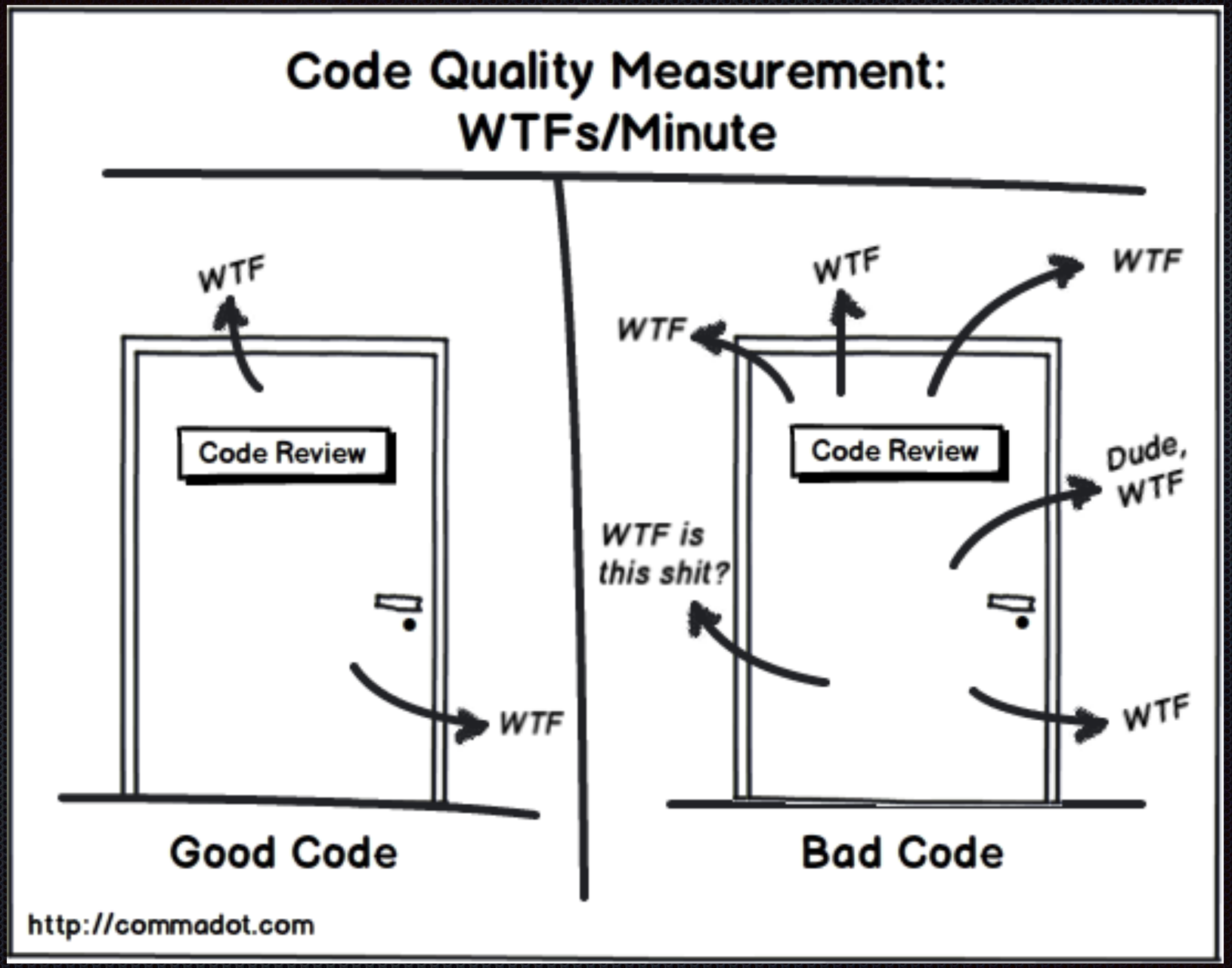
Dependency management in Ruby and Rails

RubyC

Pavel Forkert

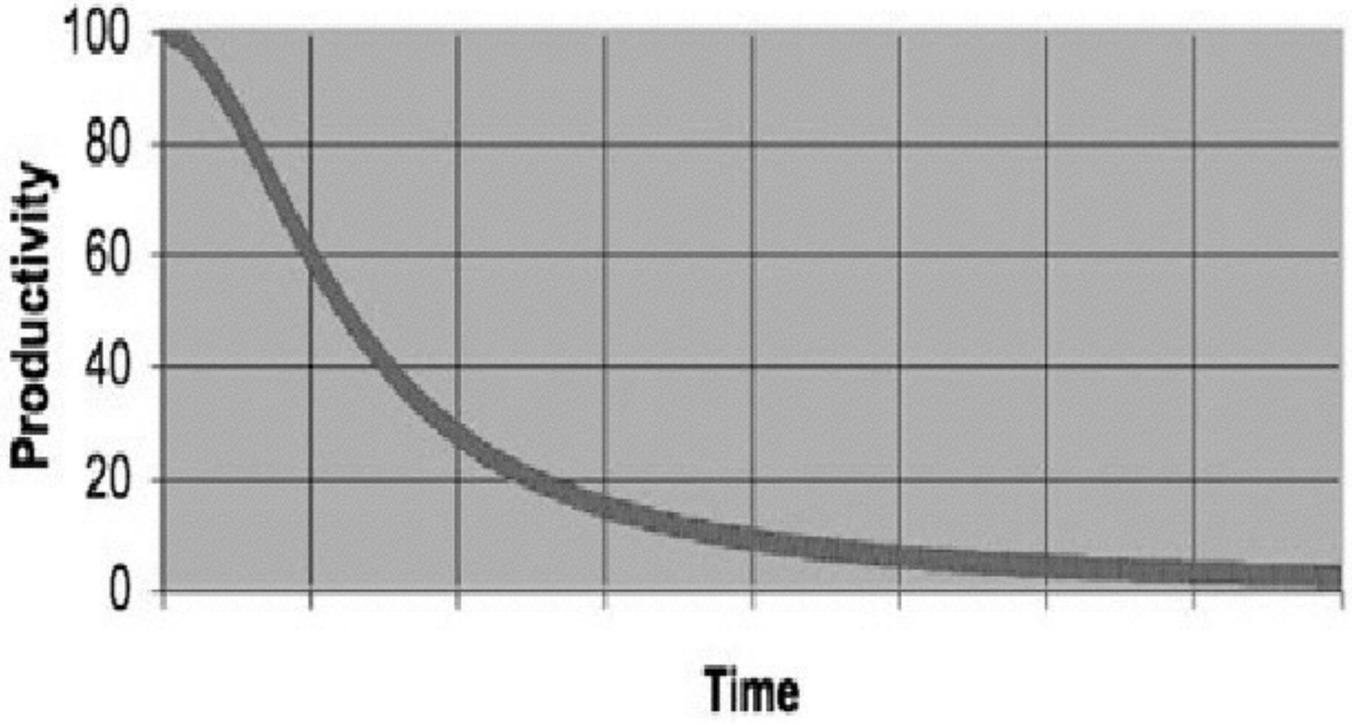
~~Bundler~~ ~~RubyGems~~


@fxposter



http://commadot.com/wtf-per-minute/

Developers productivity vs. time



Encapsulation

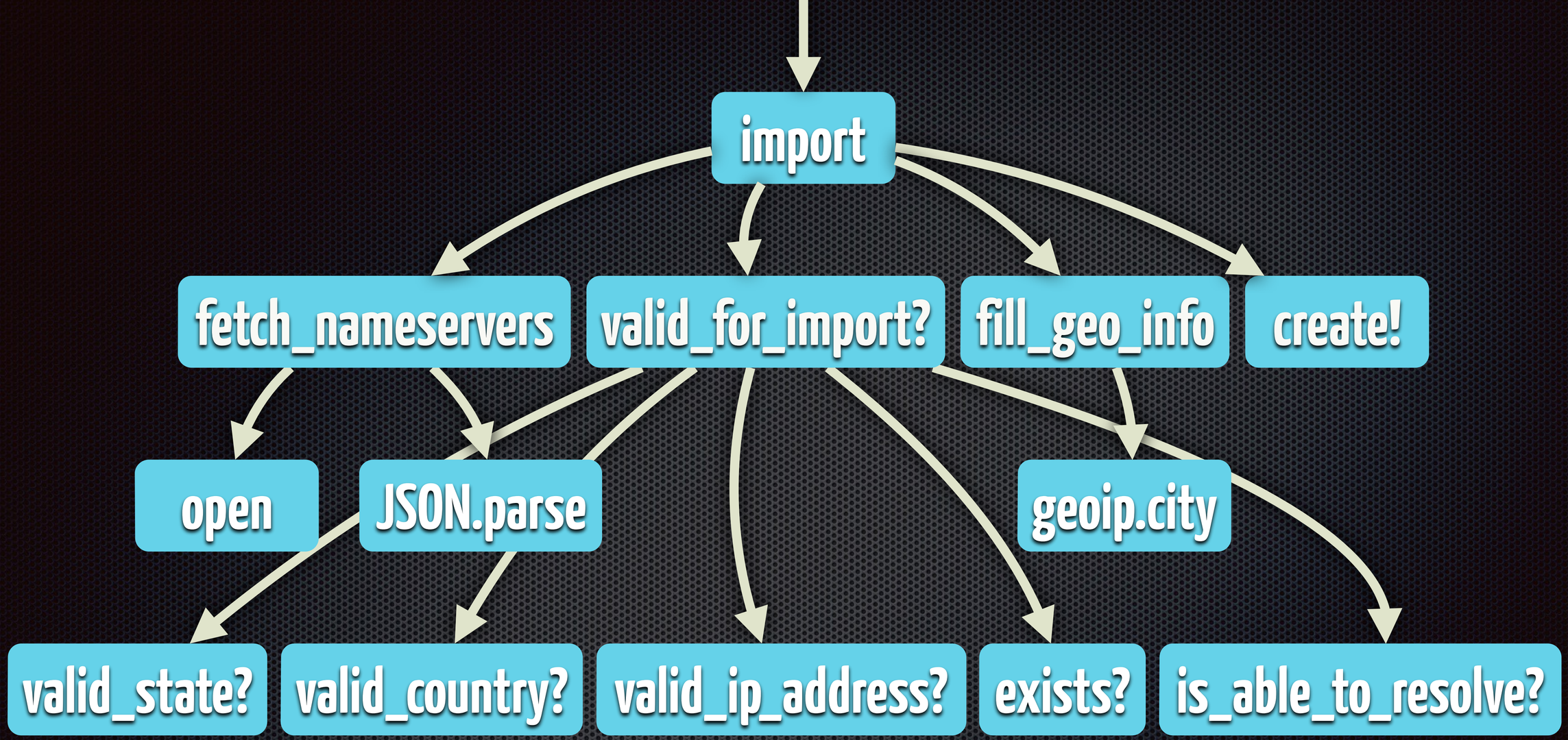
Inheritance

Polymorphism

```ruby
class Nameserver < ActiveRecord::Base
  def self.import
    nameservers = fetch_nameservers()

    geoip = GeoIP.new(Rails.root.join('db', 'GeoLiteCity.dat'))
    nameservers.each do |nameserver|
      Rails.logger.info "Processing nameserver: #{nameserver[:ip]}"
      nameserver = nameserver.slice(:country_id, :city, :state, :ip)
      if valid_for_import?(nameserver)
        fill_geo_info(geoip, nameserver)
        create!(nameserver)
      end
    end
  end
  # ...
```

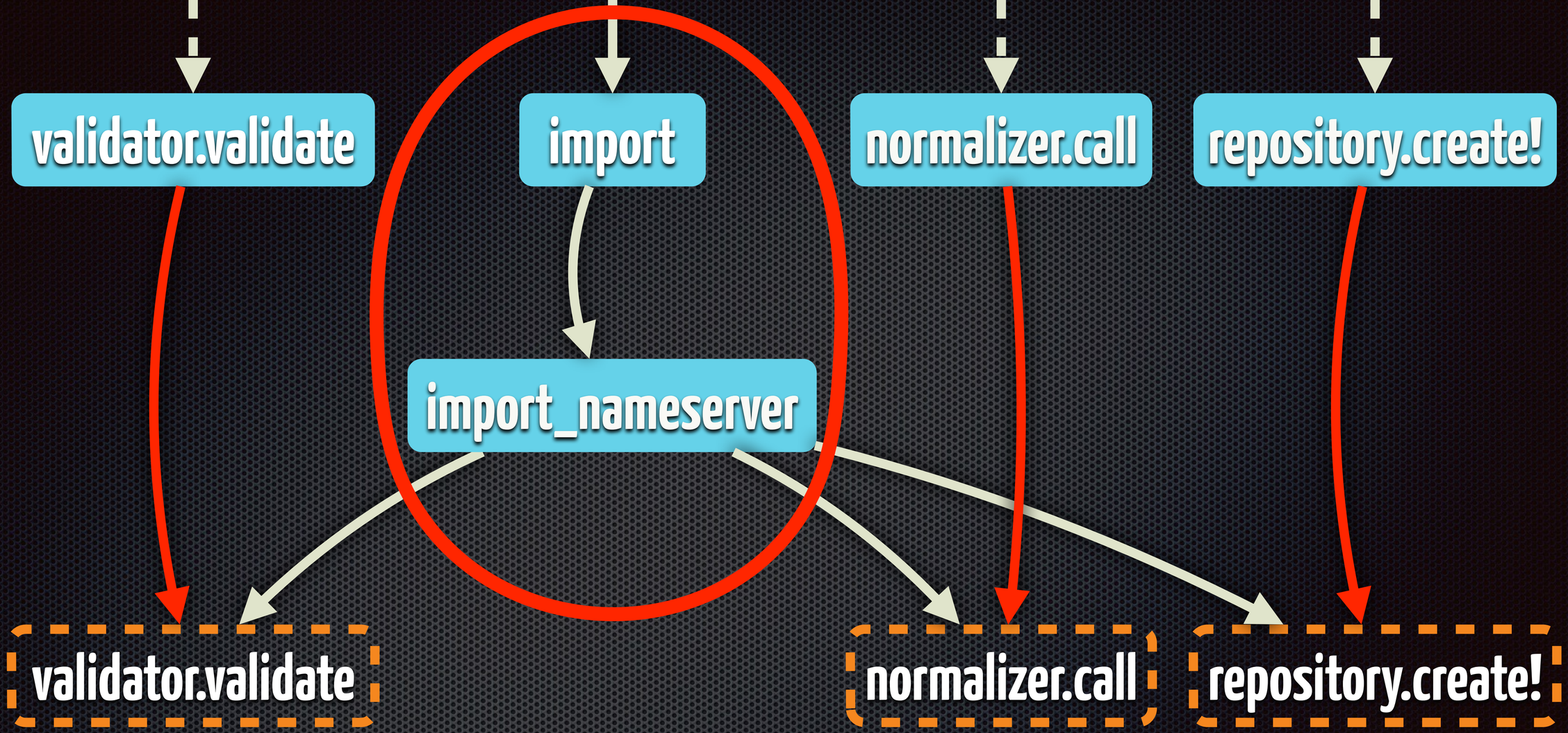
```ruby
# ...
def self.fill_geo_info(geoip, nameserver)
  nameserver[:city] = nameserver[:city].presence

  city = geoip.city(nameserver[:ip])
  nameserver[:latitude] = city.latitude
  nameserver[:longitude] = city.longitude
end

def self.fetch_nameservers
  open('http://public-dns.tk/nameservers.json') { |f|
    JSON.parse(f.read).map(&:symbolize_keys)
  }
end
# ...
```

```ruby
  # ...
  def self.valid_for_import?(nameserver)
    valid_country?(nameserver) &&
      valid_state?(nameserver) &&
      valid_ip_address?(nameserver) &&
      !exists?(:ip => nameserver[:ip]) &&
      is_able_to_resolve?(nameserver)
  end

  private
    # def self.valid_country?(nameserver)
    # def self.valid_state?(nameserver)
    # def self.valid_ip_address?(nameserver)
    # def self.is_able_to_resolve?(nameserver)
    # ...
    # Еще 10 страниц кода в том же классе
    # И этот код уже не связан с импортом
  end
end
```


import
fetch_nameservers
valid_for_import?
fill_geo_info
create!
open
JSON.parse
geoip.city
valid_state?
valid_country?
valid_ip_address?
exists?
is_able_to_resolve?

```ruby
module Nameservers
  class Importer
    def initialize(logger: Rails.logger,
                   repository: Nameserver,
                   validator: Validator.new(repository: repository),
                   data_normalizer: Normalizer.new(
                      geoip: GeoIP.new(Rails.root.join('db', 'GeoLiteCity.dat'))))
      # ...
    end

    def import(nameservers)
      nameservers.each do |nameserver|
        @logger.info "Processing nameserver: #{nameserver[:ip]}"
        import_nameserver(nameserver)
      end
    end
```

```ruby
    # ...
    private
    def import_nameserver(nameserver)
      if @validator.validate(nameserver)
        nameserver = @data_normalizer.call(nameserver)
        @repository.create!(nameserver)
      end
    end
  end
end
```


validator.validate
import
normalizer.call
repository.create!
import_nameserver
validator.validate
normalizer.call
repository.create!


Seams

1

```ruby
class MediaService::PictureStorage
  def delete(path)
    # delete ...
    cdn = Akamai.new(
              :username => 'A',
              :password => 'B',
              :hostname => 'C')
    cdn.delete(path)
  end
end
```

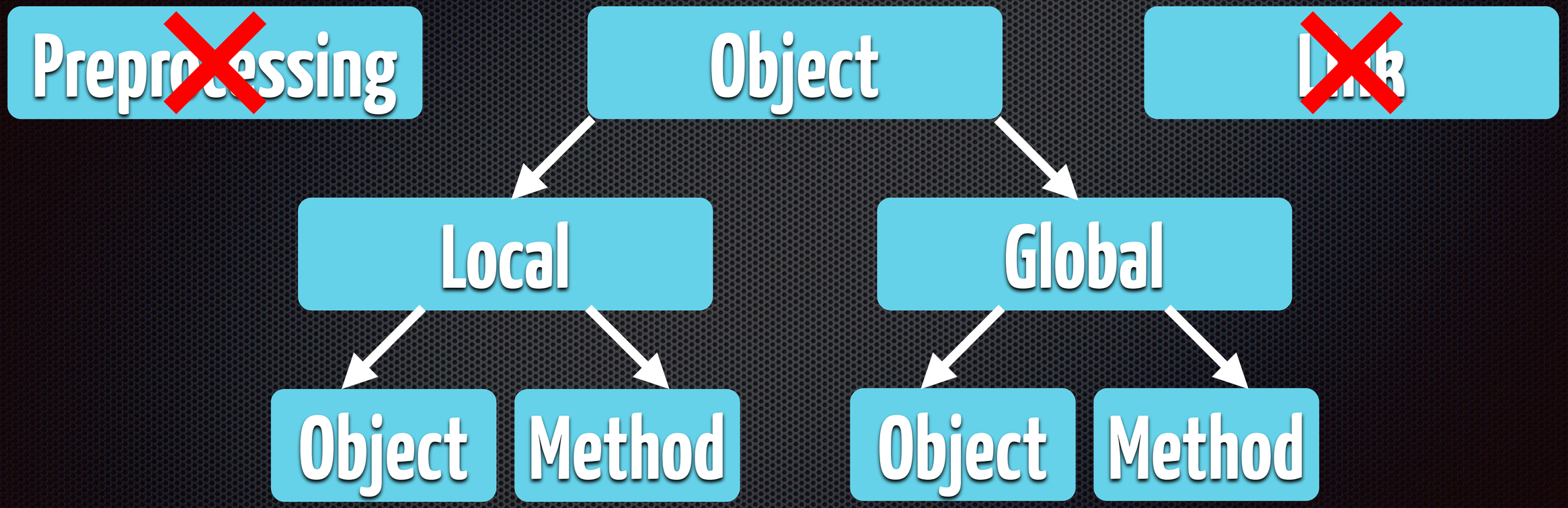
2

```ruby
class MediaService::PictureStorage
  def initialize(cdn)
    @cdn = cdn
  end

  def delete(path)
    # delete ...
    @cdn.delete(path)
  end
end
```

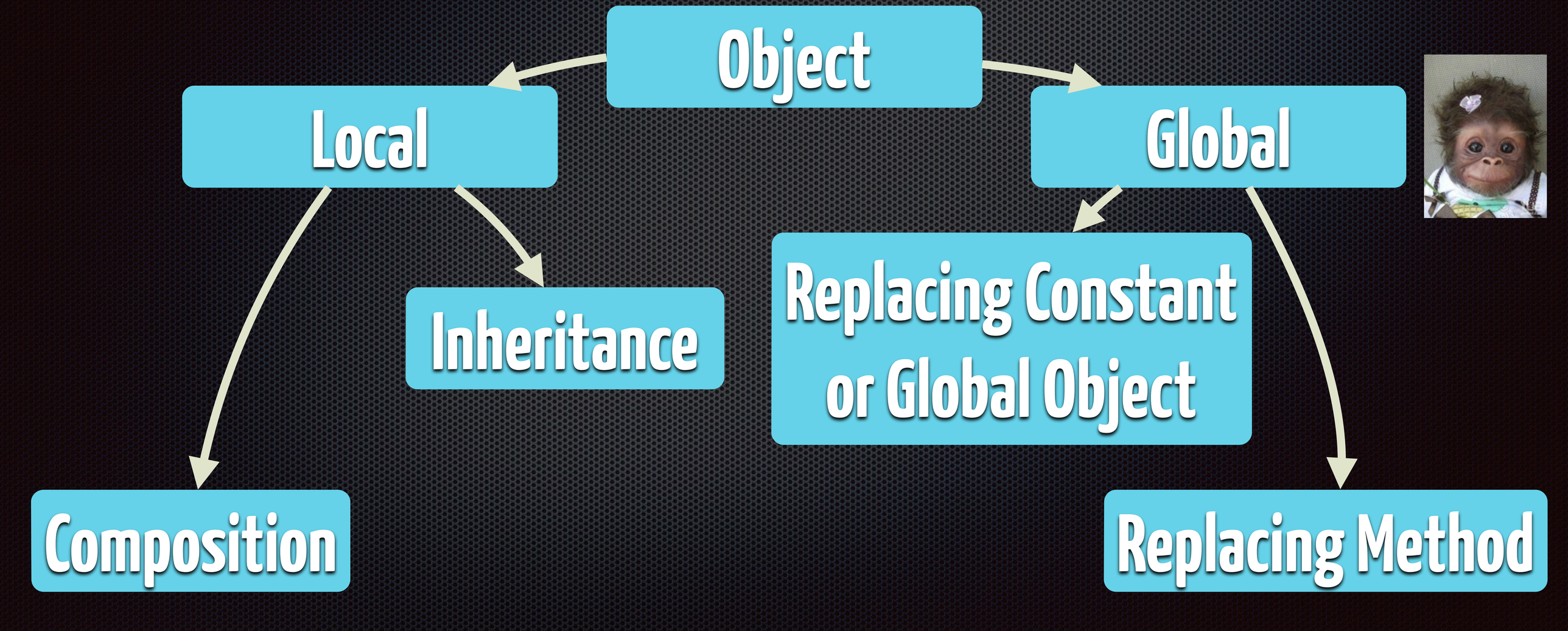
3

```ruby
class MediaService::PictureStorage
  def delete(path)
    # delete ...
    Application.cdn.delete(path)
  end
end
```

Seams


Preprocessing
Object
Link
Local
Global
Object
Method
Object
Method

Seams


Object
Local
Global
Inheritance
Replacing Constant or Global Object
Composition
Replacing Method

Global / Object / Method

1

```
class MediaServicePictureStorage
  def delete(path)
    # delete ...
    cdn = Akamai.new(
            :username => 'A',
            :password => 'B',
            :hostname => 'C')
    cdn.delete(path)
  end
end
```

Local / Object

2

```
class MediaServicePictureStorage
  def initialize(cdn)
    @cdn = cdn
  end

  def delete(path)
    # delete ...
    @cdn.delete(path)
  end
end
```

Global / Object / Method

3

```
class MediaServicePictureStorage
  def delete(path)
    # delete ...
    Application.cdn.delete(path)
  end
end
```

```ruby
class MediaService::PictureStorage
  def delete(path)
    # delete ...
    Application.cdn.delete(path)
  end
end
```

```ruby
application = Application
Application = AnotherApplication
storage.delete(path)
Application = application

method = Akamai.instance_method(:delete)
Akamai.send(:define_method, :delete) do |path|
  # ...
end
storage.delete(path)
Akamai.send(:define_method, :delete, method)
```

```ruby
class MediaService::PictureStorage
  def delete(path)
    # delete ...
    Application.cdn.delete(path)
  end
end
```

```ruby
method = Application.cdn.method(:delete)
Application.cdn.define_singleton_method(:delete) do |path|
  # ...
end
storage.delete(path)
Application.cdn.define_singleton_method(:delete, method)

Application.with_cdn(akamai) do
  storage.delete(path)
end
```


Single Responsibility
Open-Closed
Liskov Substitution
Interface Segregation
Dependency Inversion

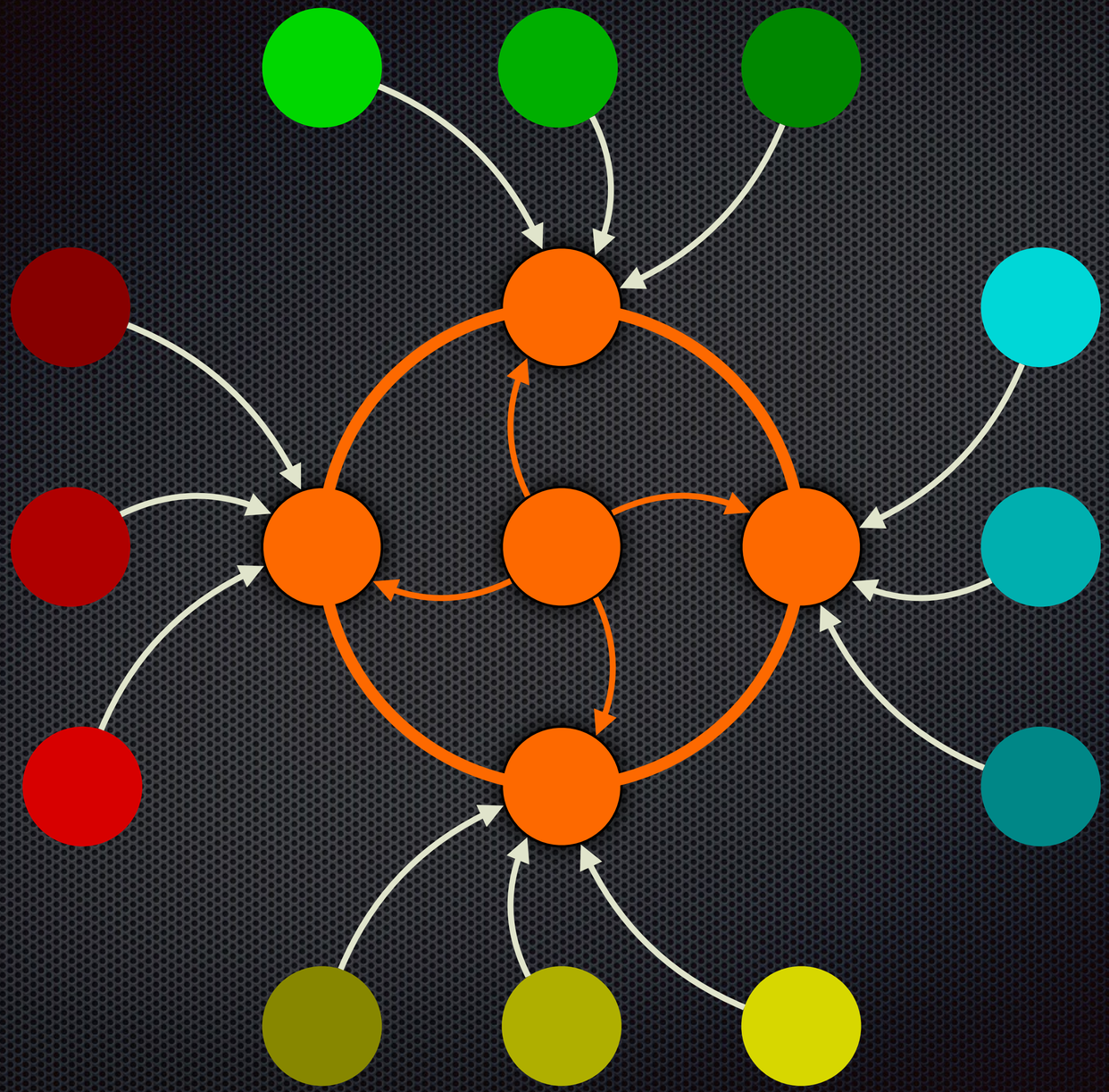
Interface Ownership

```ruby
class AddsProductToCart
  def initialize(user)
    @user = user
  end

  def call(product)
    if product.published? && @user.is_customer_of?(product.vendor)
      @user.add_to_cart(product)
    end
  end
end
```

```ruby
class User
  def add_to_cart(product)
    # ...
  end
end
```

```ruby
class User
  def profile
  end
end

class Profile
  def add_to_cart(product)
  end
end
```

```ruby
user.add_to_cart(product) -> user.profile.add_to_cart(product)
```



What if this class was in some gem?

```
class AddsProductToCart
  def initialize(user)
    @user = user
  end

  def call(product)
    if product.published? && @user.is_customer_of?(product.vendor)
      @user.add_to_cart(product)
    end
  end
end
```


I DON'T ALWAYS MOCK
BUT WHEN I DO
I MOCK EVERYTHING

Dependency management

is all about

specifying clear boundaries between objects

```ruby
class MediaService::PictureStorage
  def initialize(cdn)
    @cdn = cdn
  end

  def get(path)
    # ...
  end

  def update(path, file)
    # ...
    @cdn.delete(path)
  end

  def delete(path)
    # ...
    @cdn.delete(path)
  end
end
```

```ruby
class MediaService::PictureStorage
  def initialize(cdn)
    @cdn = cdn
  end

  def get(path)
    # ...
  end

  def update(path, file)
    # ...
    @cdn.updated(path, file)
  end

  def delete(path)
    # ...
    @cdn.deleted(path)
  end
end
```

```
class MediaService::PictureStorage
 def initialize(listener)
    @listener = listener
  end

  def update(path, file)
    @listener.updated(path, file)
  end

  def delete(path)
    @listener.deleted(path)
  end
end
```

Events

```ruby
class MediaService::PictureStorage
  attr_reader :events

  def initialize(events)
    @events = events
  end

  def update(path, file)
    @events.updated(path, file)
    # or
    @events.trigger(:updated, path, file)
  end

  def delete(path)
    @events.deleted(path)
    # or
    @events.trigger(:deleted, path)
  end
end
```

```ruby
class EventSet
  def on(event, &observer)
    @observers ||= {}
    @observers[event] ||= []
    @observers[event] << observer
  end

  def off(event, &observer)
    return unless @observers
    return unless @observers[event]
    @observers[event].delete(observer)
  end

  def trigger(event, *args)
    if @observers && @observers[event]
      @observers[event].each do |observer|
        observer.call(*args)
      end
    end
  end
end
```

```ruby
MyGem.configure do |config|
end

MyGem.call_method
```

```ruby
config = MyGem::Config.new(:a => 'b')
client = MyGem::Client.new(config)
client.call_method
```

ActiveValidations

- Separate validators, errors and objects being validated

```
address_validator = Validator.new {
  validates :street, PresenceValidator
  validates :city, PresenceValidator
  validates :country, InclusionValidator.new(in: COUNTRIES)
}

address_validator.validate(address)
=> ValidationResult(value: address, success: false, errors:
[{ key: :street, value: address.street, resolution: :invalid }])
```

- Separate validators, errors and objects being validated
- Easy-to-write validators, that have as less dependencies as possible

```ruby
class PresenceValidator
  def validate(value, errors)
    if value.blank?
      errors.add(:invalid)
    end
  end
end
```

- Separate validators, errors and objects being validated
- Easy-to-write validators, that have as less dependencies as possible
- Support nested objects and arrays

```
person_validator = Validator.new {
  validates :name, PresenceValidator
  validates :surname, PresenceValidator
  validates :email, FormatValidator.new(format: EMAIL_REGEXP)
  validates :address, address_validator
  validates_many :accounts, account_validator
}
```

- Separate validators, errors and objects being validated
- Easy-to-write validators, that have as less dependencies as possible
- Support nested objects and arrays
- Support both hash and objects (ie: pluggable "getters" model)

```ruby
address_validator.validate(street: 'Baker',
                           city: 'London',
                           country: 'UK')

class HashGetter
  def get(hash, key)
    hash[key]
  end
end

address_validator.validate(Address.new(street: 'Baker',
                                       city: 'London',
                                       country: 'UK'))

class ObjectGetter
  def get(object, key)
    object.public_send(key)
  end
end
```

```
data = {
  street_address: 'Baker',
  city: 'London',
  country_code: 'UK'
}
```

```
class MappingGetter
  def initialize(parent, hash)
    @parent = parent
    @hash = hash
  end

  def get(object, key)
    @parent.get(object, @hash.fetch(key) { key })
  end
end
```

```
data = {
  street_address: 'Baker',
  city: 'London',
  country_code: 'UK'
}
```

```
hash_getter = HashGetter.new
mapping_getter = MappingGetter.new(hash_getter,
                                   street: :street_address,
                                   country: :country_code)
address_validator.validate(data, getter: mapping_getter)
=> ValidationResult(value: address, success: true, errors: [])
```

It's just a concept

Hope for your feedback

Thanks!