Hexagonal Rails

Luiz Costa
gutomcosta@gmail.com
@gutomcosta

THE
DEVELOPER'S
CONFERENCE

beep!

nosso propósito é criar as melhores experiências em serviços médicos

hexágonos?

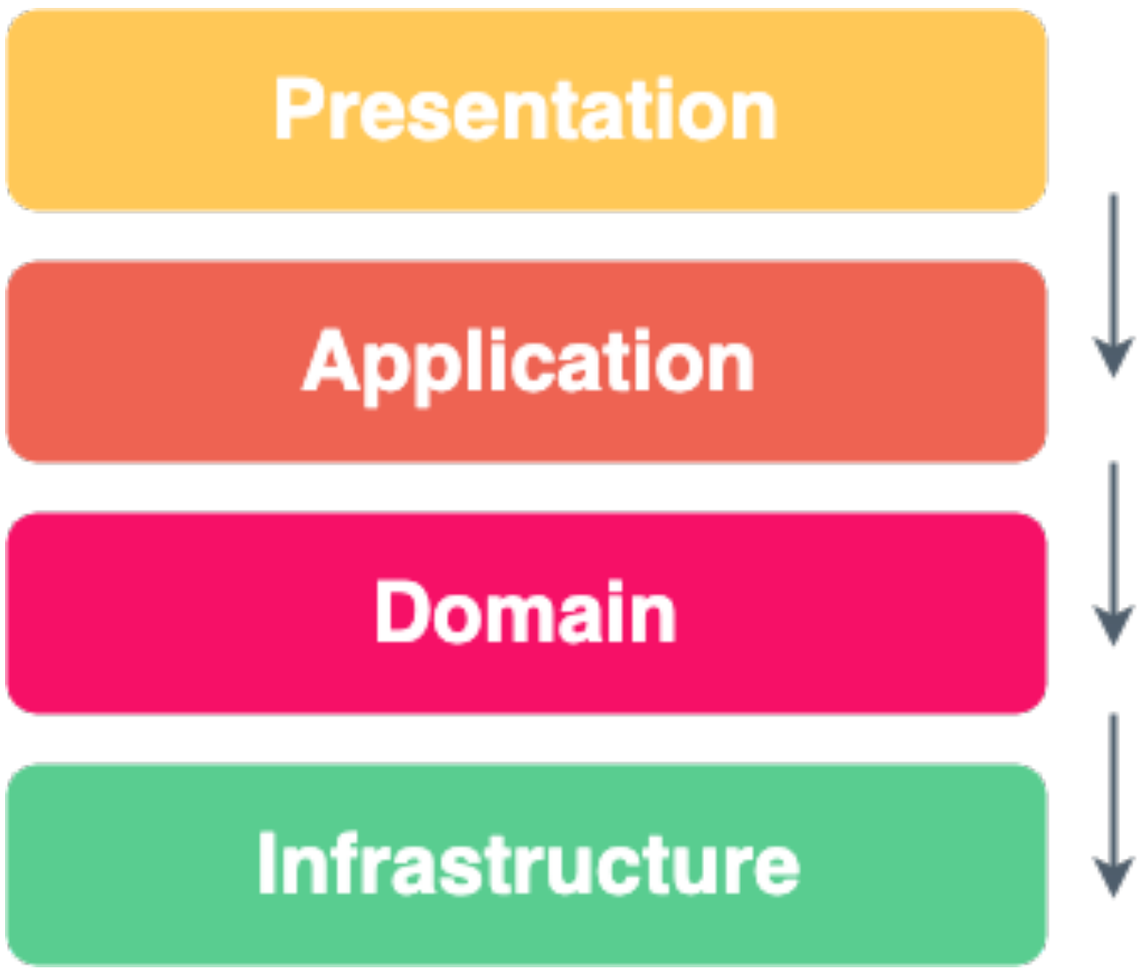
ou ports and adapters

mas antes, layers...

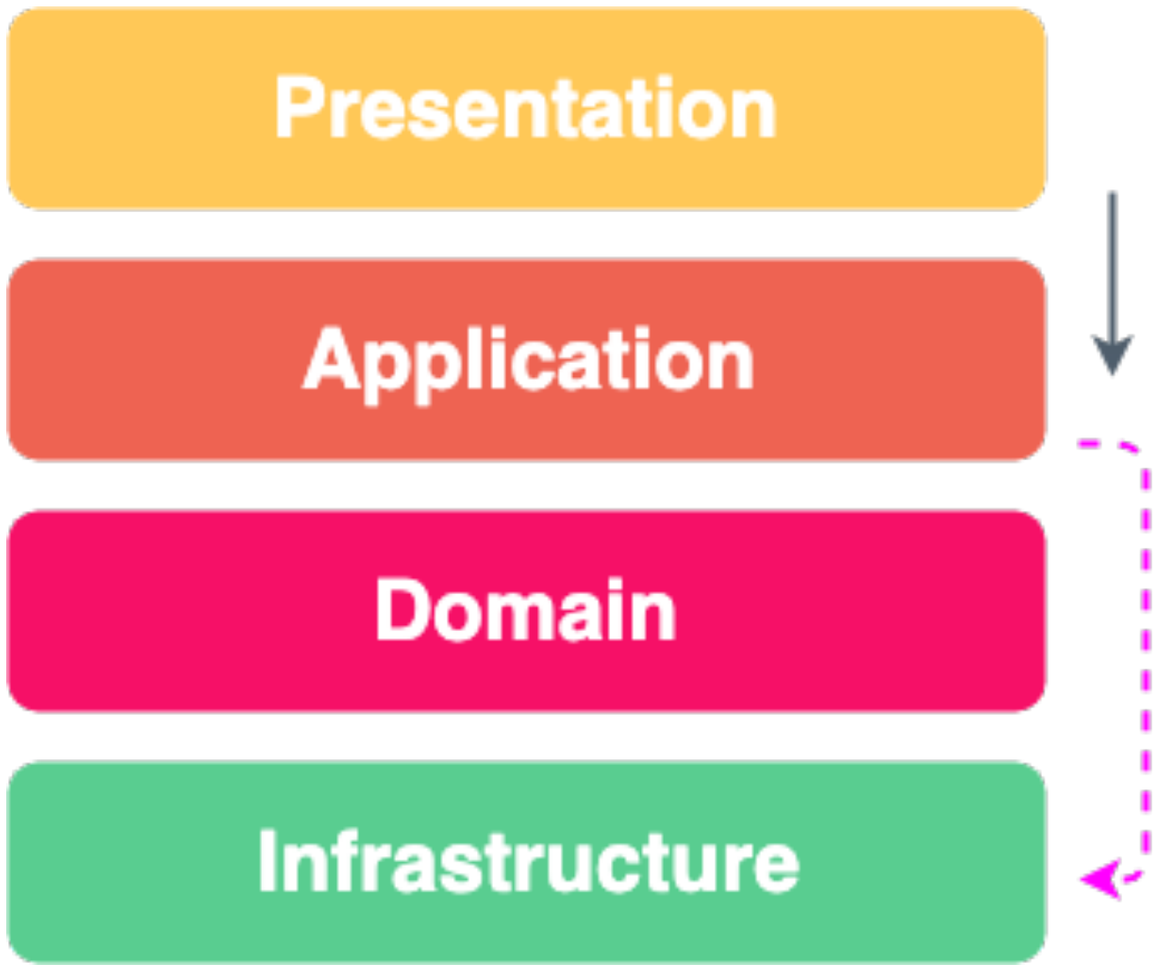
ou simplesmente camadas

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic).

Software Architecture Patterns, Mark Richards
https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/



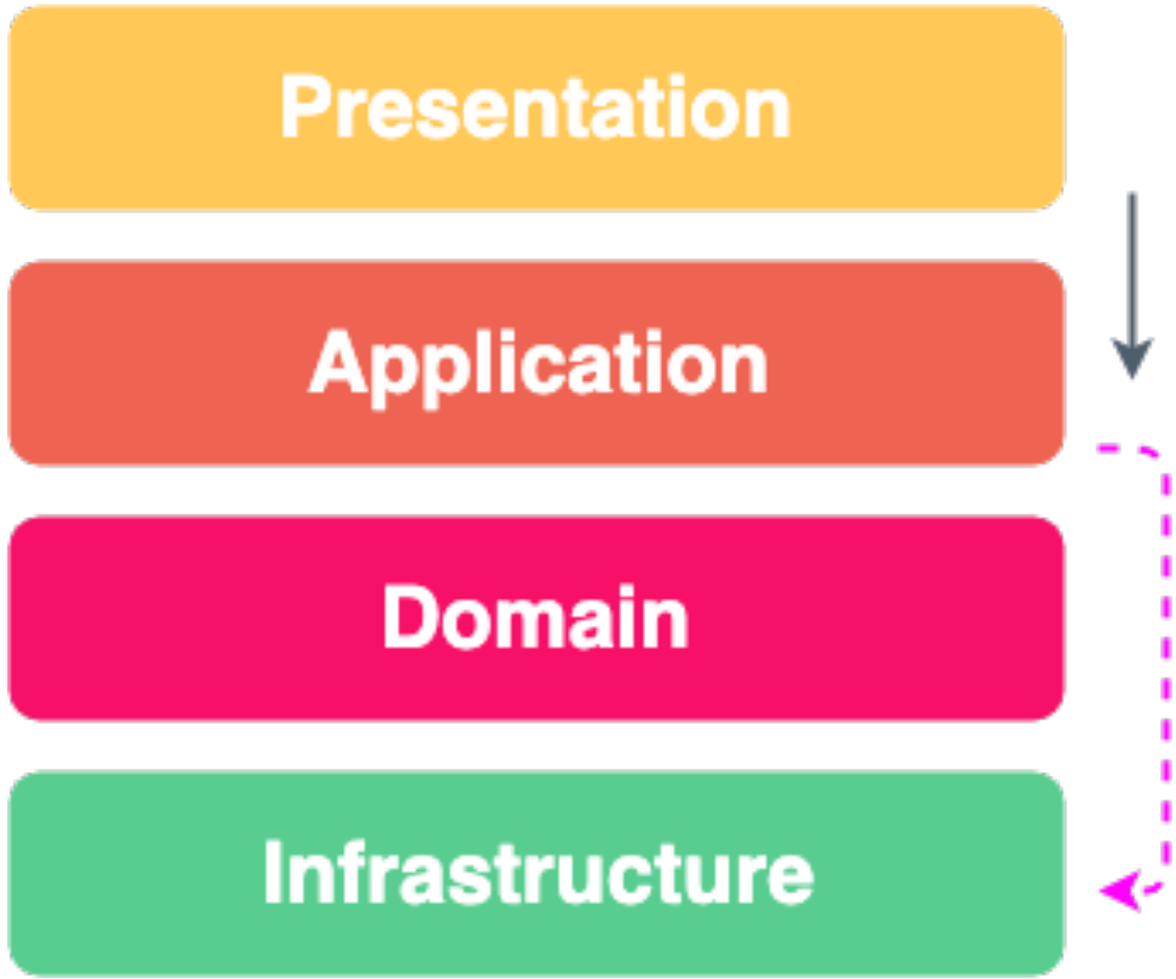
Na versão mais “restrita” deste pattern, as camadas superiores só acessam as inferiores passando pela camada seguinte.

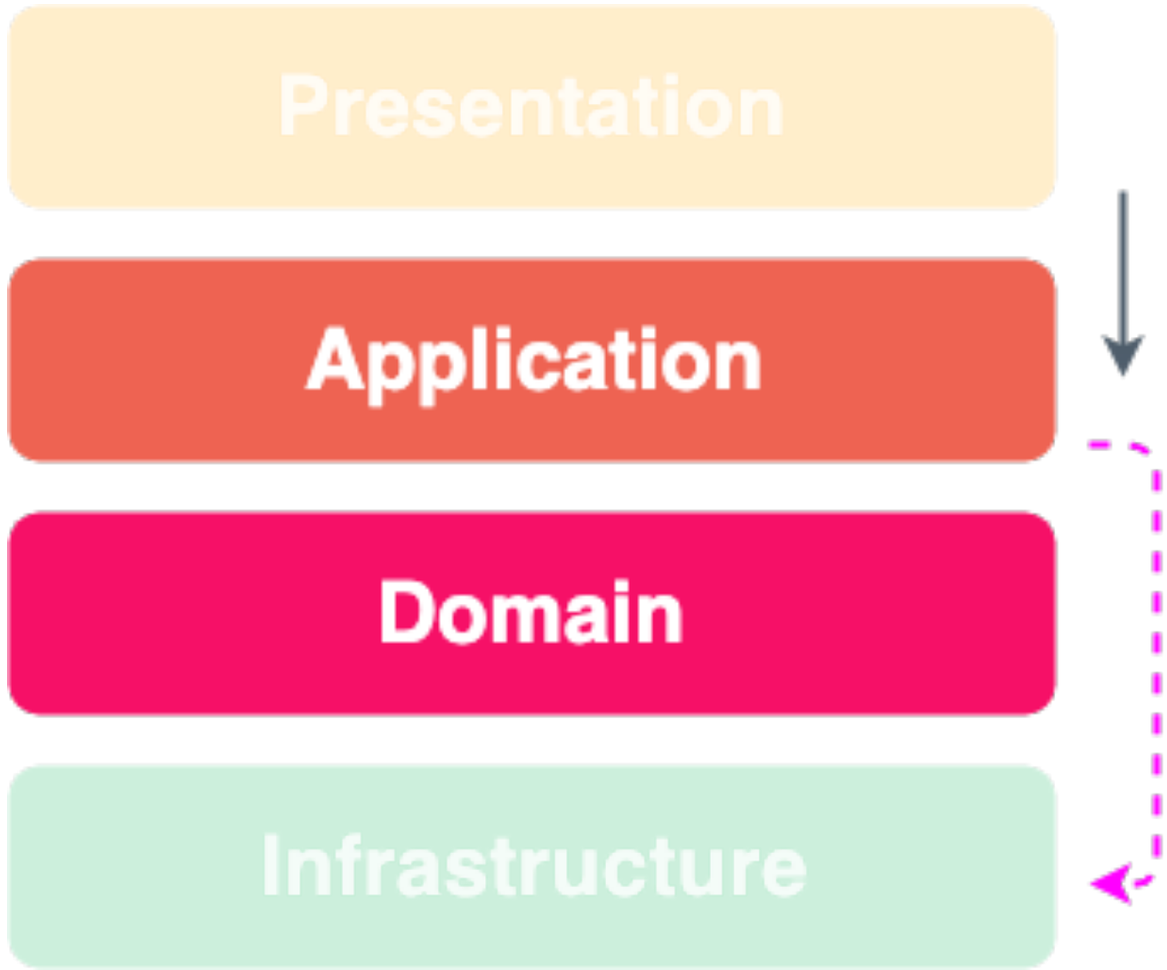


Existem algumas variações, por exemplo, quando algumas camadas podem acessar diretamente outras camadas sem passar pela seguinte. Neste caso diz-se que existem camadas “abertas”.

hexágonos?

ou ports and adapters


Presentation
Application
Domain
Infrastructure


Presentation
Application
Domain
Infrastructure

Presentation

Business

Infrastructure

Presentation

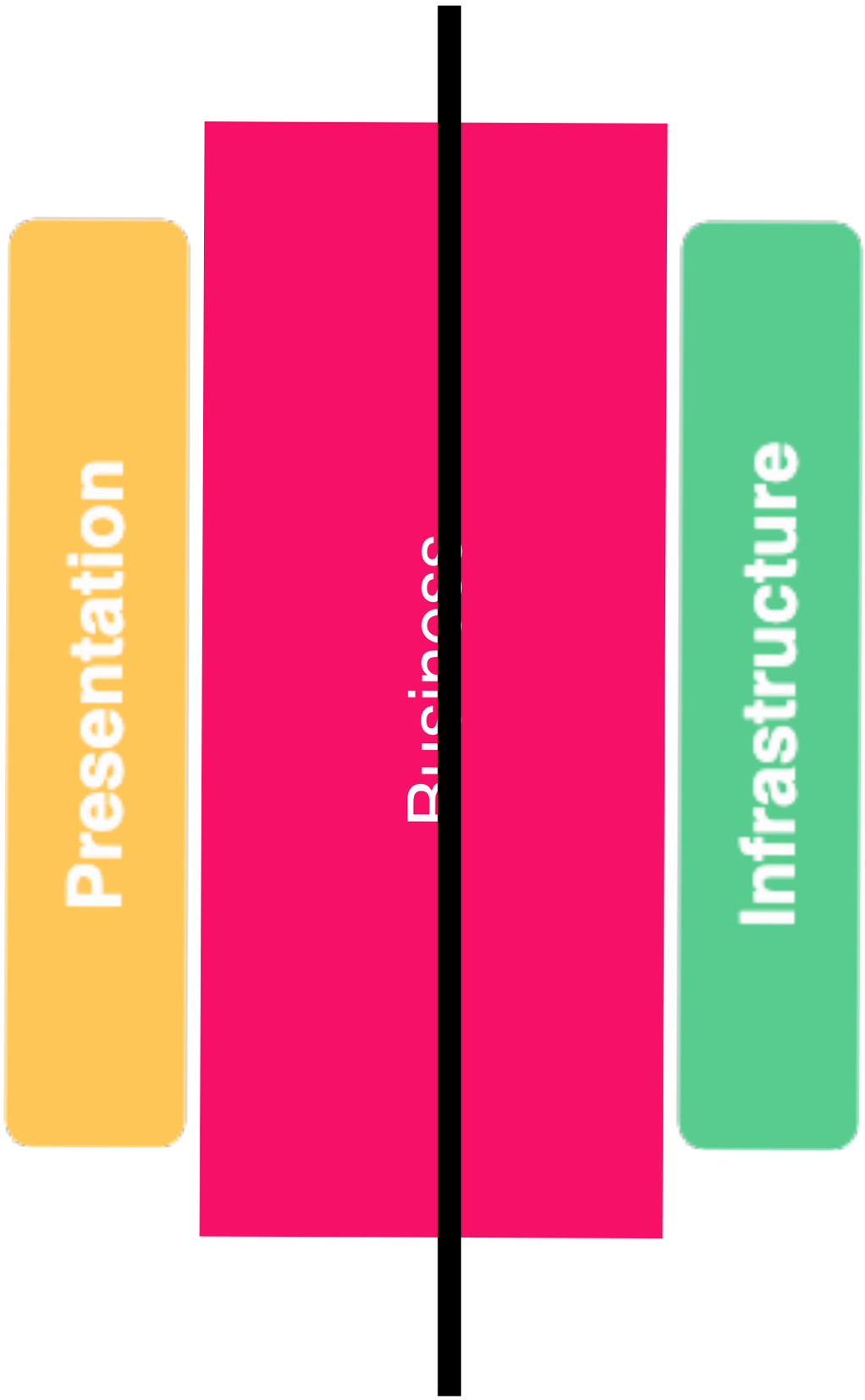
Business

Infrastructure

entry / exit points

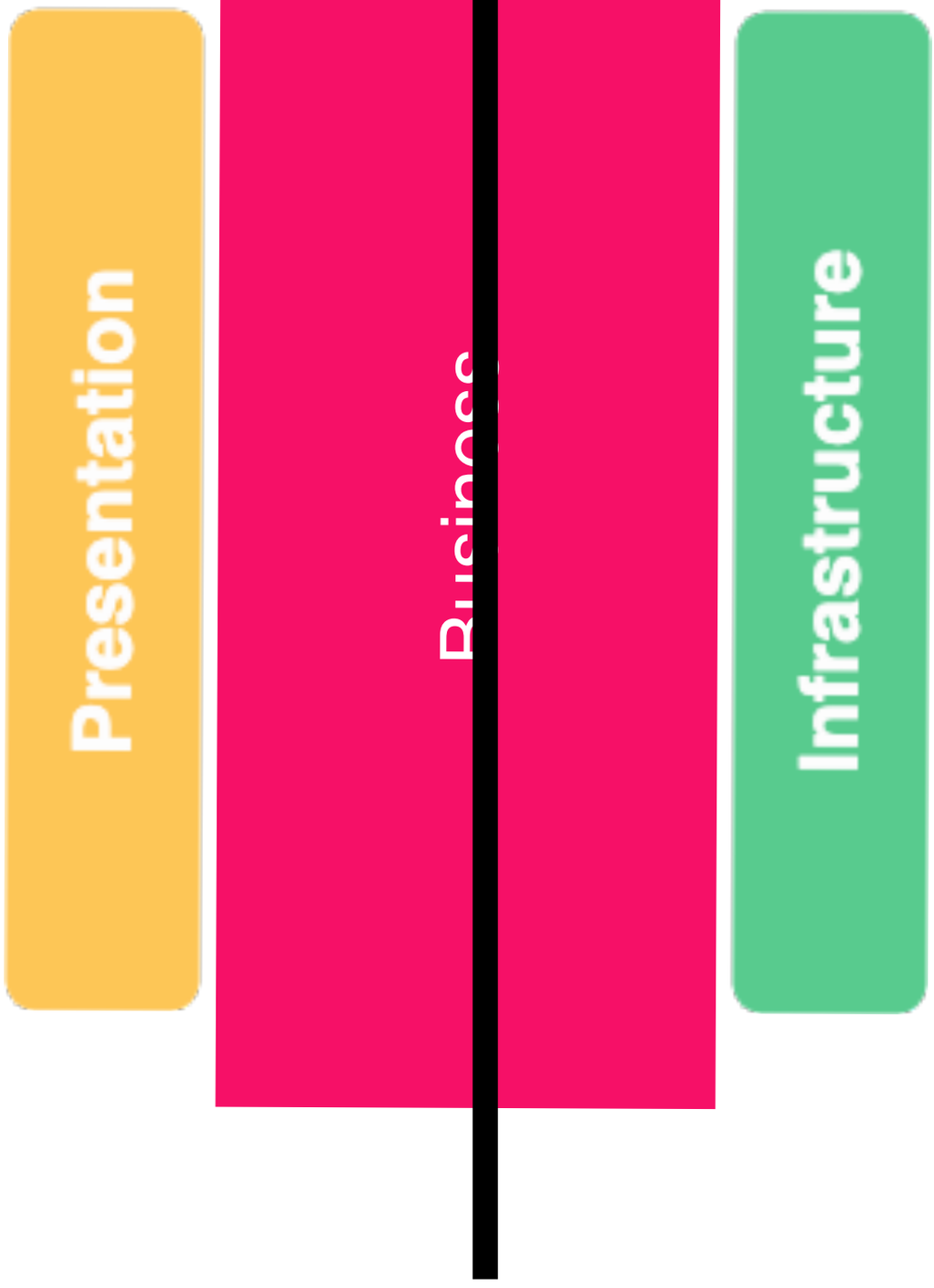
entry

exit



entry

exit



cada lado pode ter

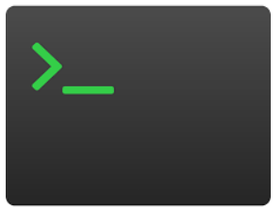
vários entry / exit points

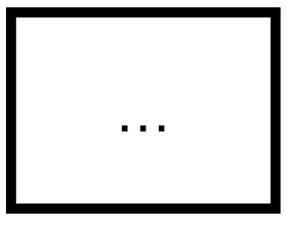
entry

exit

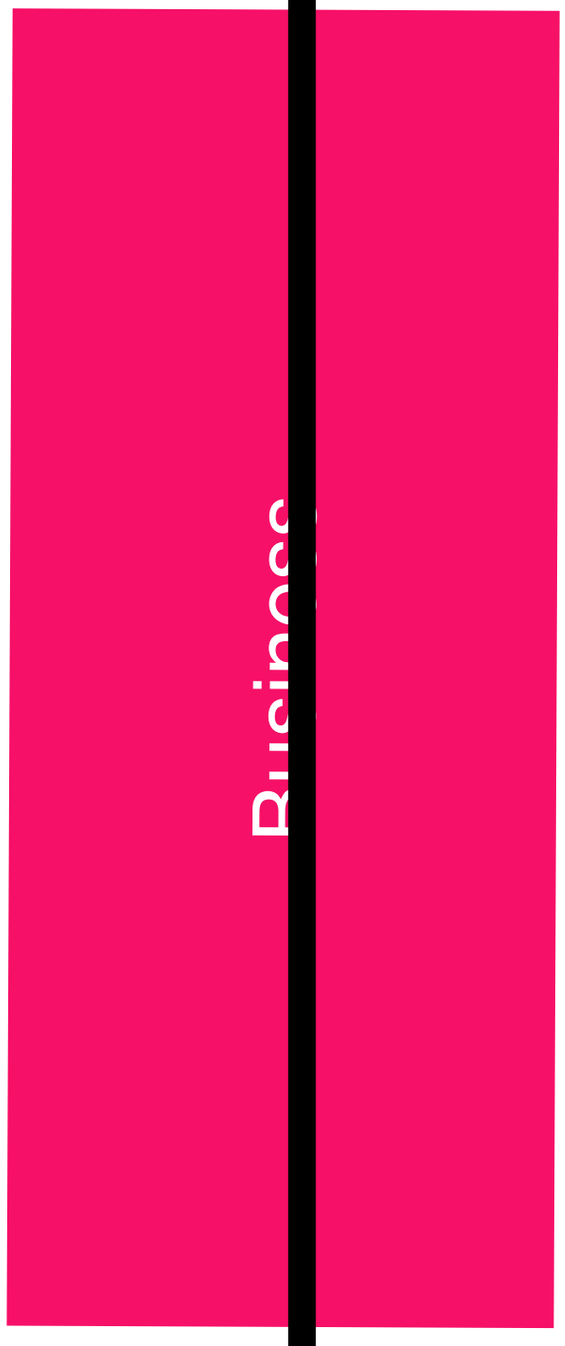

EDIT PROFILE


API

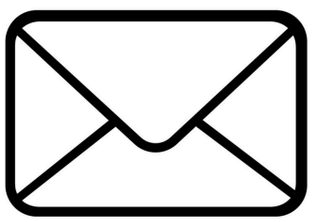



...

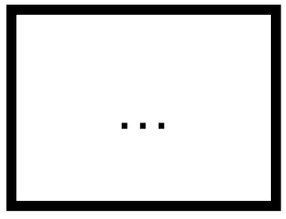

Presentation


Business


Infrastructure

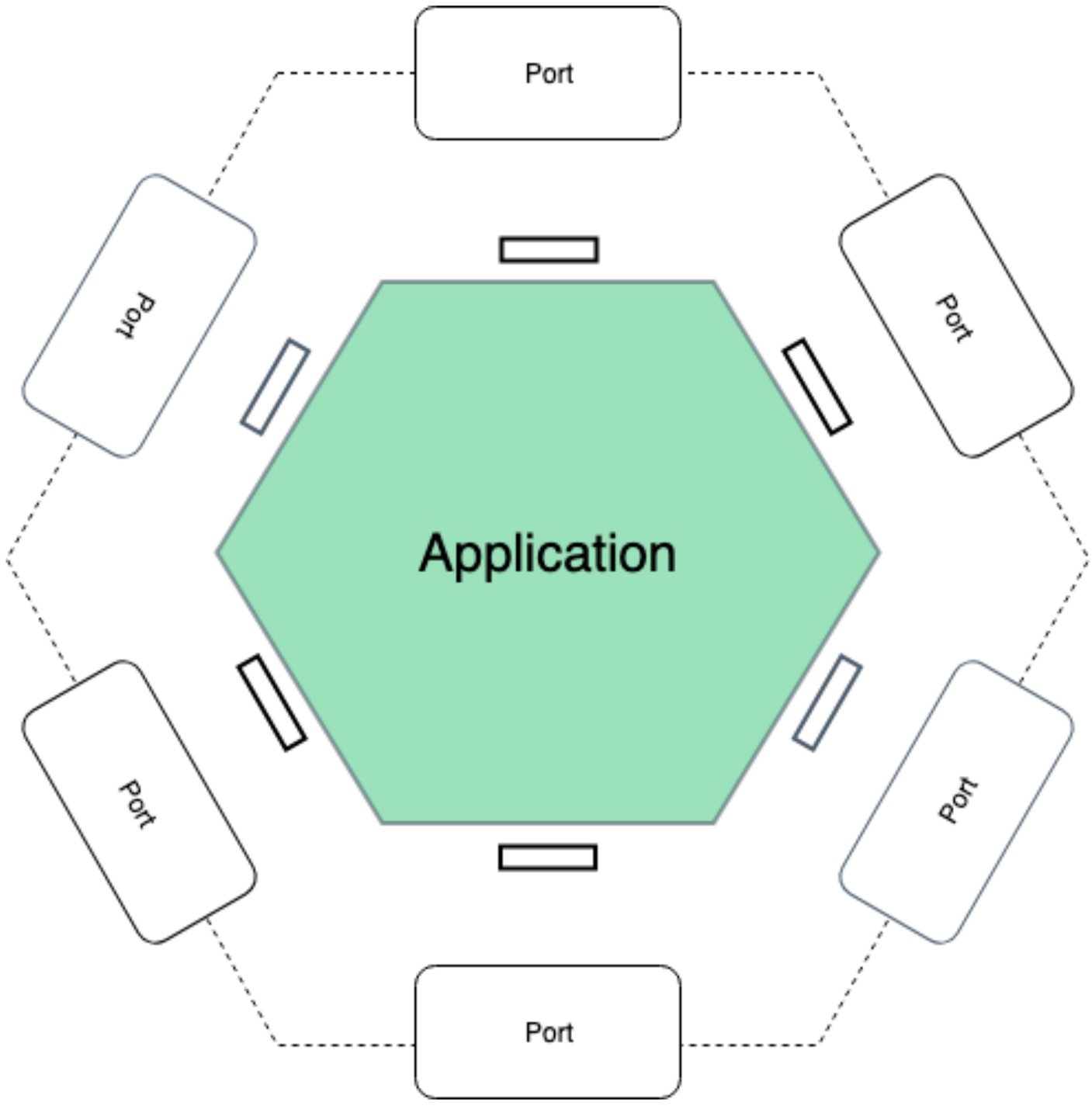



SMS


...

hexágonos

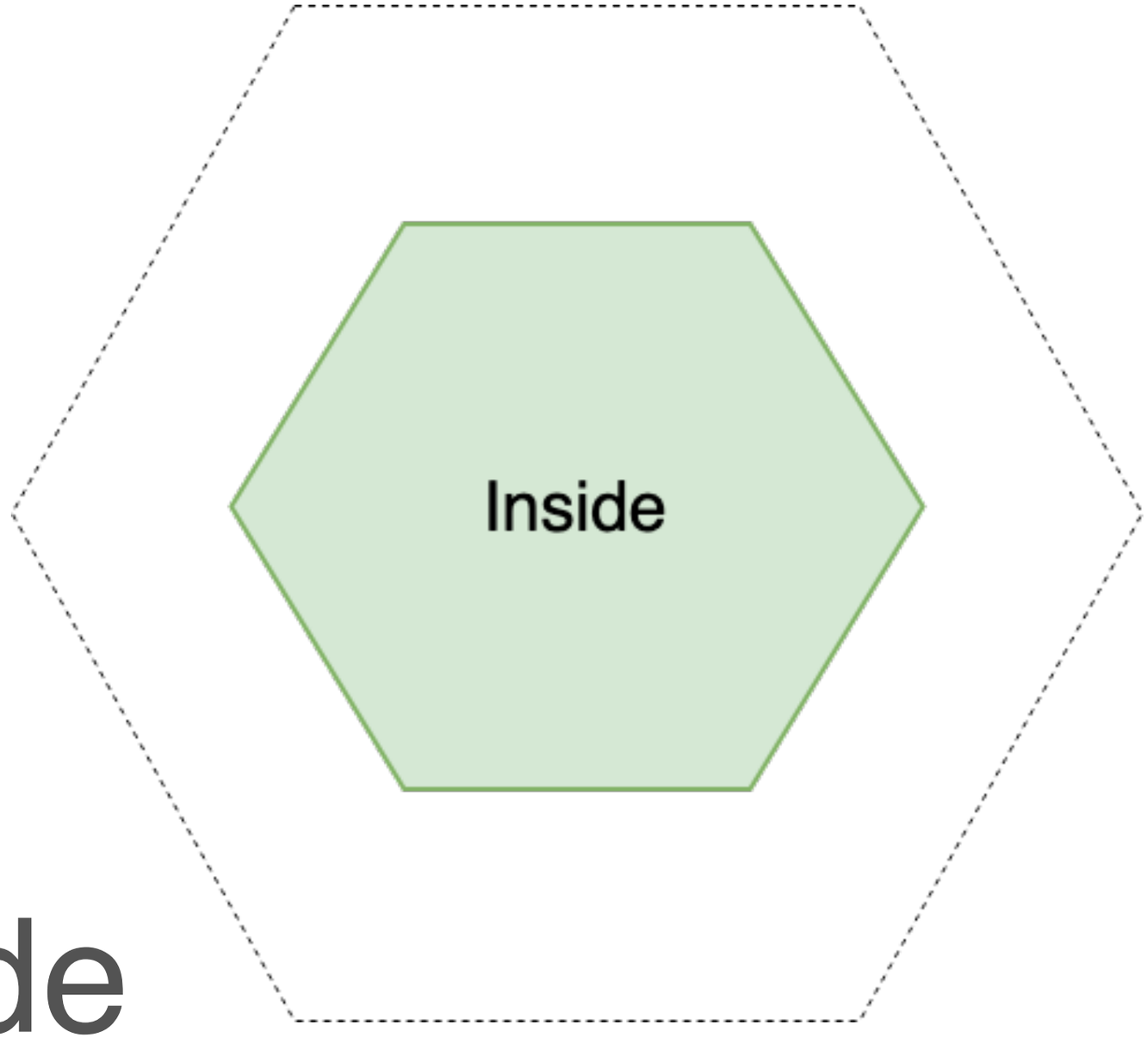
finalmente!


Port
Port
Port
Application
Port
Port
Port

Allow an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases.

Hexagonal Architecture, Alistair Cockburn
https://bit.ly/2XBQHNx

ao invés de entry/exit points, agora **inside/outside**



outside

ports and adapters

Each face of the hexagon (port) represents some "reason" the application is trying to talk with the outside world. Events arrive from the outside world at a port. The adapter converts it into a usable procedure call or message and passes it to the application. The application is blissfully ignorant of the nature of the input device…

Ports and Adapters Architecture, Alistair Cockburn

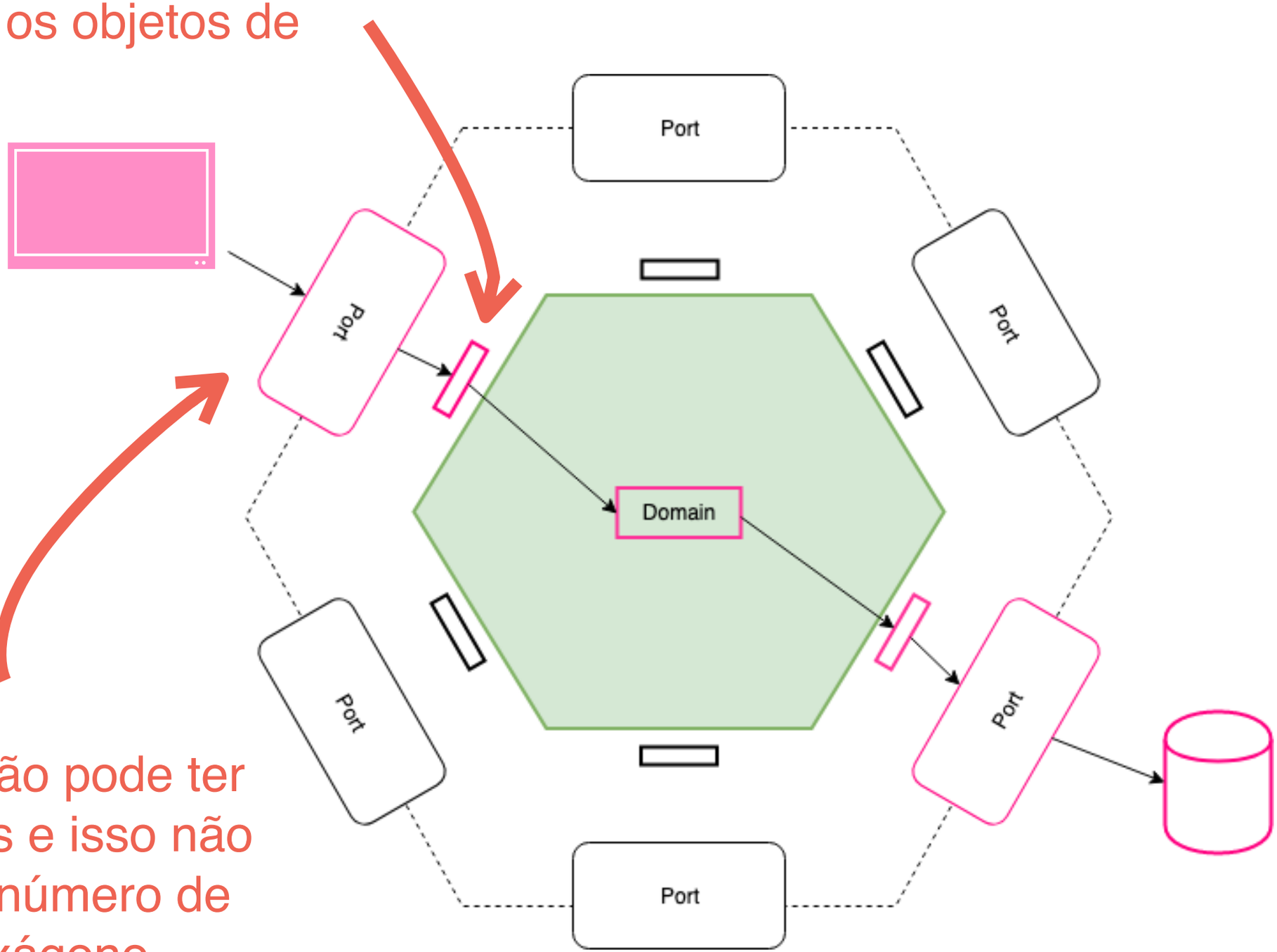
http://wiki.c2.com/?PortsAndAdaptersArchitecture

Each face of the hexagon (**port**) represents some "reason" the application is trying to talk with the outside world. Events arrive from the outside world at a port. The **adapter** converts it into a usable procedure call or message and passes it to the application. The application is blissfully ignorant of the nature of the input device…

Ports and Adapters Architecture, Alistair Cockburn

http://wiki.c2.com/?PortsAndAdaptersArchitecture

O adaptador converte a mensagem e manipula ou delega para os objetos de domínio

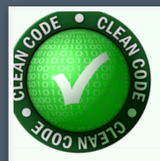


uma aplicação pode ter várias portas e isso não se limita ao número de lados do hexágono

algumas vantagens

- **diminuir acoplamento**
- **melhor para testar**
- **adaptabilidade**

clean architecture



The Clean Code Blog

by Robert C. Martin (Uncle Bob)

atom/rss feed

- Classes vs. Data Structures
  06-16-2019
- Types and Tests
  06-08-2019
- 737 Max 8
  05-18-2019
- FP vs. OO List Processing
  12-17-2018
- We, The Unoffended
  12-16-2018
- SJWJS
  12-14-2018
- The Tragedy of Craftsmanship.
  08-28-2018
- Too Clean?
  08-13-2018
- Integers and Estimates
  06-21-2018

Screaming Architecture

30 September 2011

Imagine that you are looking at the blueprints of a building. This document, prepared by an architect, tells you the plans for the building. What do these plans tell you?

If the plans you are looking at are for a single family residence, then you'll likely see a front entrance, a foyer leading to a living room and perhaps a dining room. There'll likely be a kitchen a short distance away, close to the dining room. Perhaps a dinette area next to the kitchen, and probably a family room close to that. As you looked at those plans, there'd be no question that you were looking at a *house*. The architecture would *scream*: **house**.

Or if you were looking at the architecture of a library, you'd likely see a grand entrance, an area for check-in-out clerks, reading areas, small conference rooms, and gallery after gallery capable of holding bookshelves for all the books in the library. That architecture would *scream*: **Library**.

So what does the architecture of your application scream? When you look at the top level directory structure, and the source files in the highest level package; do they

FEEDS

Atom
RSS

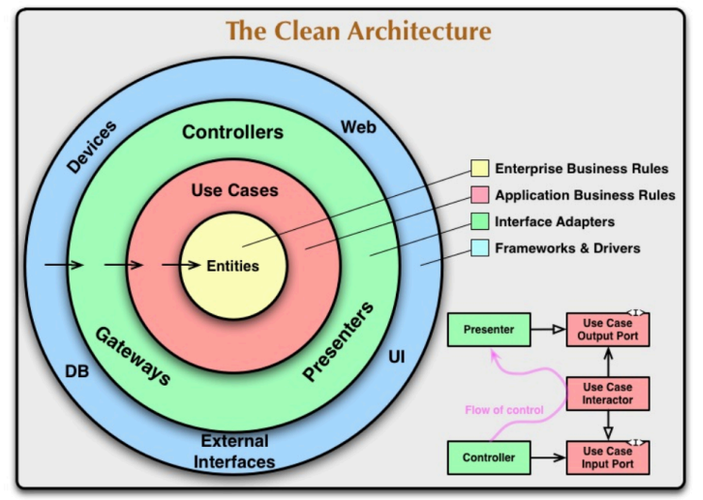
TOPICS

8th Light University
AWS
Android
Ansible
Apprentice Blog of the Week
Apprenticeship
Architecture
Bugs
Business
Clojure
ClojureScript
Coding
Communications
Community
Consulting
Craftsmanship
Design
DevOps
Elixir
Front-end
Hypermedia
Inspiration
Java
JavaScript
Learning
Microservices
Mobbing
Pairing
Principles

The Clean Architecture

Uncle Bob / 13 Aug 2012 Architecture Craftsmanship

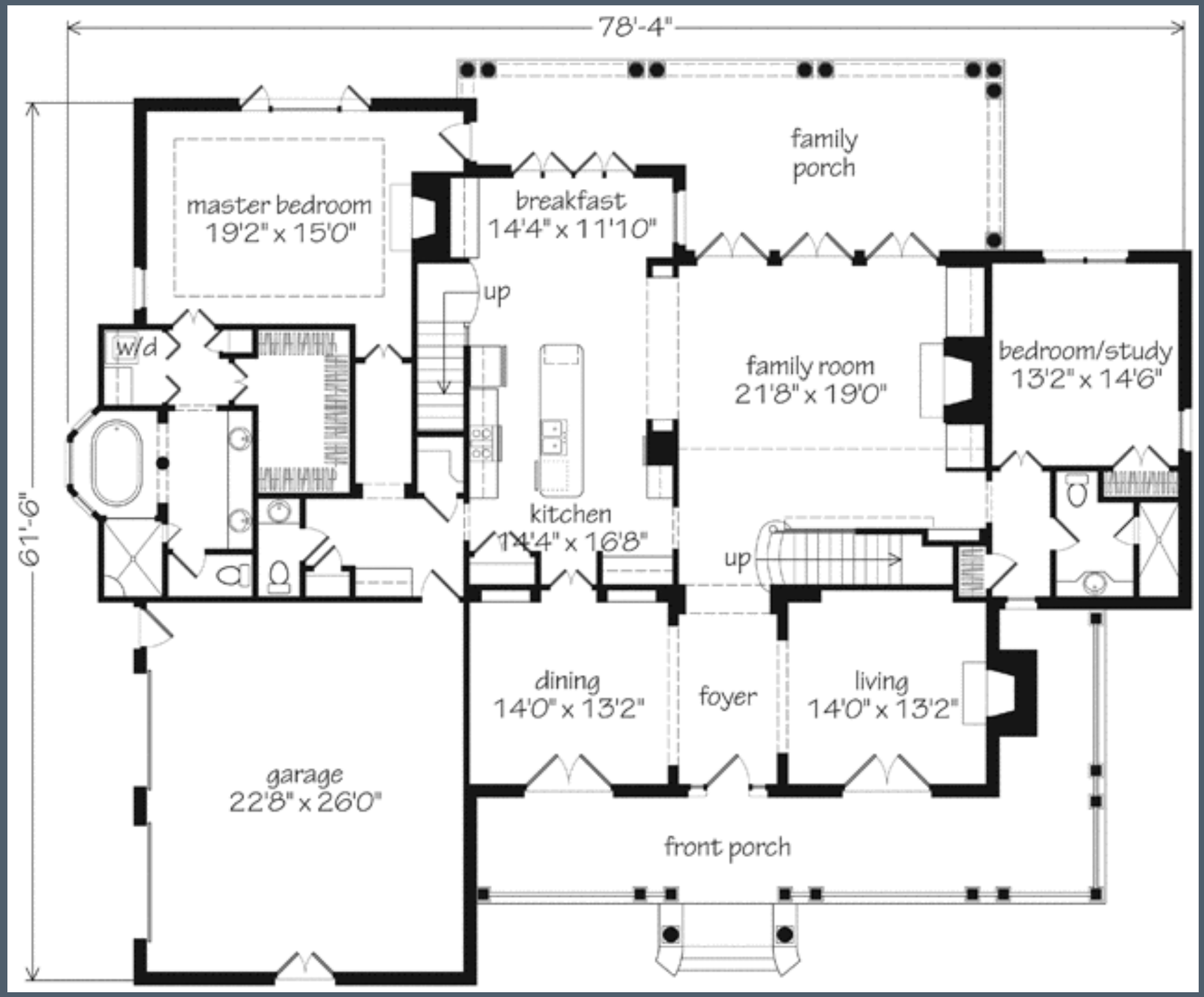
Share Tweet Share



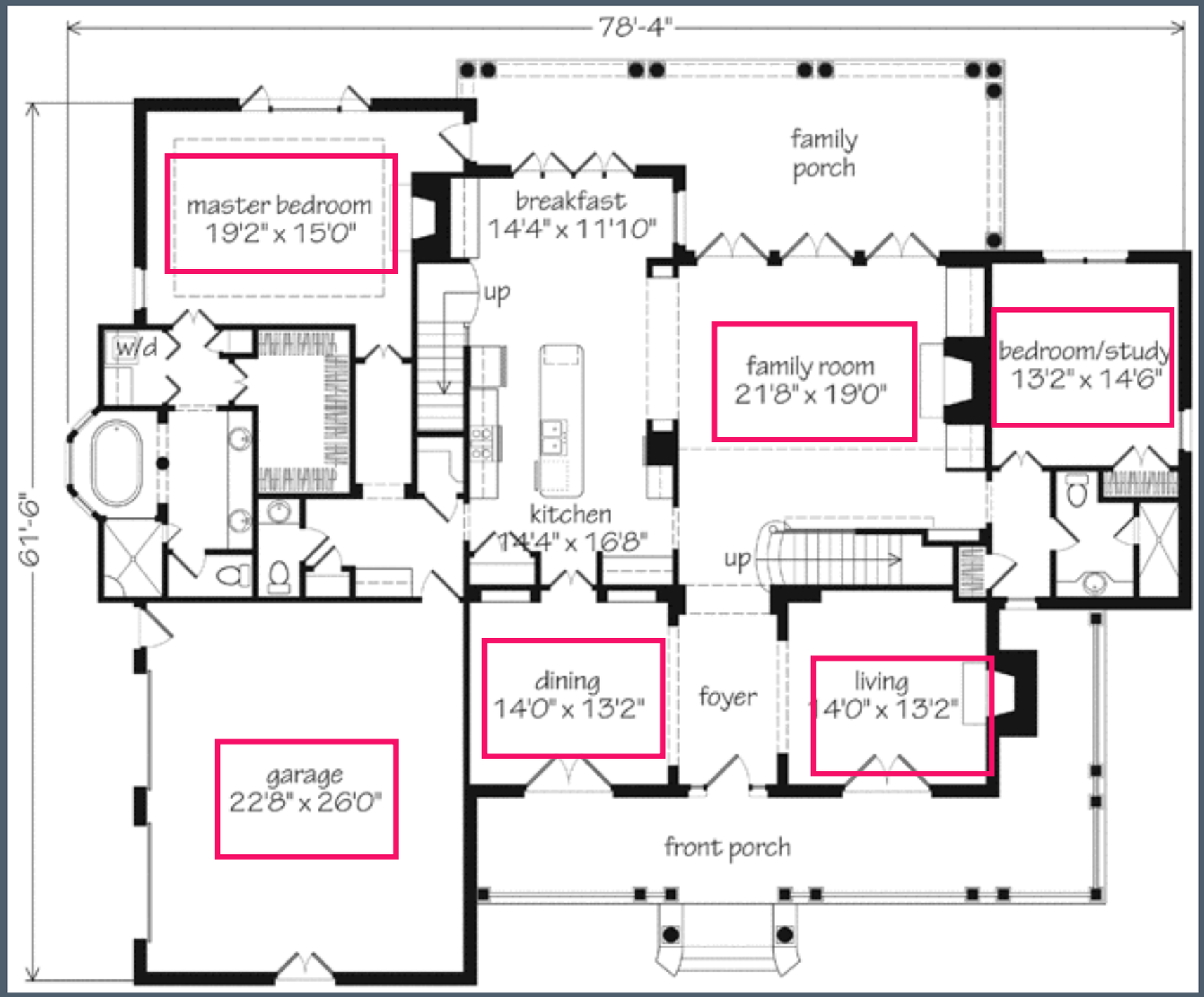
Over the last several years we've seen a whole range of ideas regarding the architecture of systems. These include:

- Hexagonal Architecture (a.k.a. Ports and Adapters) by Alistair Cockburn and adopted by Steve Freeman, and Nat Pryce in their wonderful book Growing Object Oriented Software
- Onion Architecture by Jeffrey Palermo
- Screaming Architecture from a blog of mine last year
- DCI from James Coplien, and Trygve Reenskaug.

sobre o quê?

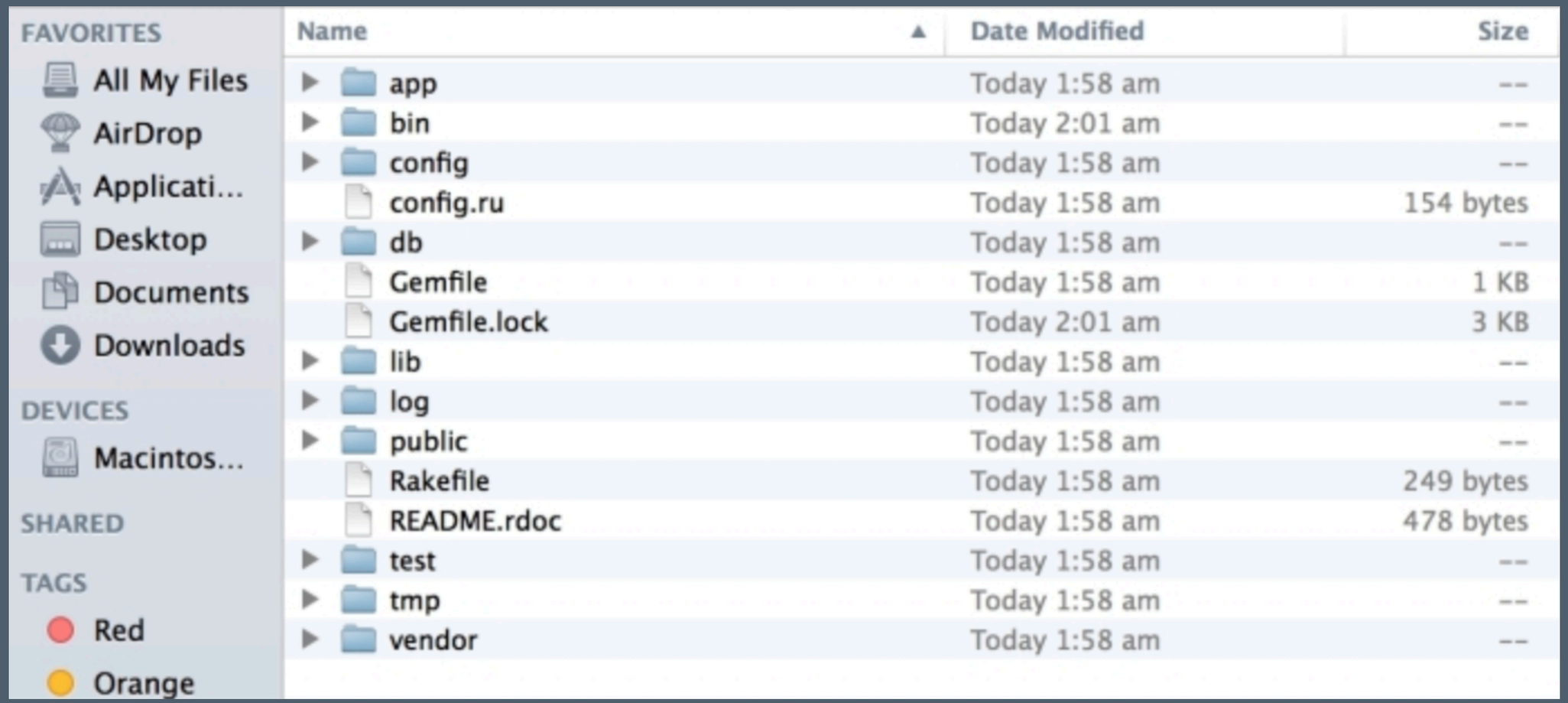

78'-4"
61'-6"
master bedroom
19'2" x 15'0"
breakfast
14'4" x 11'10"
family
porch
up
w/d
family room
21'8" x 19'0"
bedroom/study
13'2" x 14'6"
kitchen
14'4" x 16'8"
up
dining
14'0" x 13'2"
foyer
living
14'0" x 13'2"
garage
22'8" x 26'0"
front porch

sobre o quê?


78'-4"
61'-6"
master bedroom
19'2" x 15'0"
breakfast
14'4" x 11'10"
family
porch
up
w/d
family room
21'8" x 19'0"
bedroom/study
13'2" x 14'6"
kitchen
14'4" x 16'8"
up
dining
14'0" x 13'2"
foyer
living
14'0" x 13'2"
garage
22'8" x 26'0"
front porch

como uma rails app se parece?

sobre o quê?


FAVORITES
All My Files
AirDrop
Applicati...
Desktop
Documents
Downloads
DEVICES
Macintos...
SHARED
TAGS
Red
Orange
Name
Date Modified
Size
app
Today 1:58 am
--
bin
Today 2:01 am
--
config
Today 1:58 am
--
config.ru
Today 1:58 am
154 bytes
db
Today 1:58 am
--
Gemfile
Today 1:58 am
1 KB
Gemfile.lock
Today 2:01 am
3 KB
lib
Today 1:58 am
--
log
Today 1:58 am
--
public
Today 1:58 am
--
Rakefile
Today 1:58 am
249 bytes
README.rdoc
Today 1:58 am
478 bytes
test
Today 1:58 am
--
tmp
Today 1:58 am
--
vendor
Today 1:58 am
--

Architectures are not (or should not) be about frameworks. Architectures should not be supplied by frameworks. Frameworks are tools to be used, not architectures to be conformed to. If you architecture is based on frameworks, then it cannot be based on your use cases.

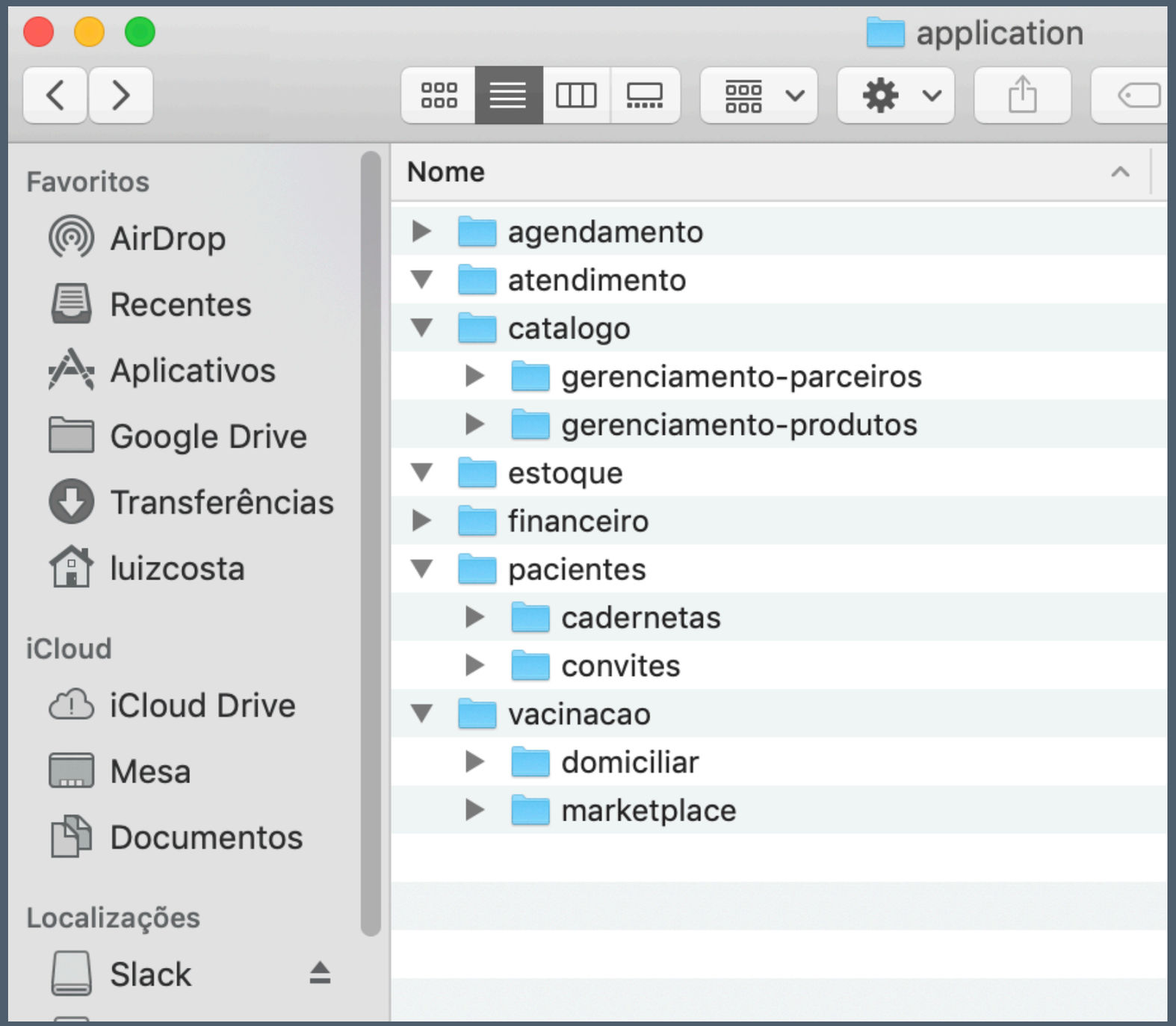
Screaming Architecture, Robert Martin
https://blog.cleancoder.com/uncle-bob/2011/09/30Screaming-Architecture.html

Architectures are not (or should not) be about frameworks. Architectures should not be supplied by frameworks.

Frameworks are tools to be used, not architectures to be conformed to. If you architecture is based on frameworks, then it cannot be based on your use cases.

Screaming Architecture, Robert Martin
https://blog.cleancoder.com/uncle-bob/2011/09/30Screaming-Architecture.html

sobre o quê?


application
Favoritos
AirDrop
Recentes
Aplicativos
Google Drive
Transferências
luizcosta
iCloud
iCloud Drive
Mesa
Documentos
Localizações
Slack
Nome
agendamento
atendimento
catalogo
gerenciamento-parceiros
gerenciamento-produtos
estoque
financeiro
pacientes
cadernetas
convites
vacinacao
domiciliar
marketplace

Your architectures should tell readers about the system, not about the frameworks you used in your system. If you are building a health-care system, then when new programmers look at the source repository, their first impression should be: “Oh, this is a health-care system”.

Screaming Architecture, Robert Martin
https://blog.cleancoder.com/uncle-bob/2011/09/30Screaming-Architecture.html

architecture is about **intent.**

Architecture the lost years, Robert Martin
https://youtu.be/WpkDN78P884?t=704

rails is not your application

it’s a web framework

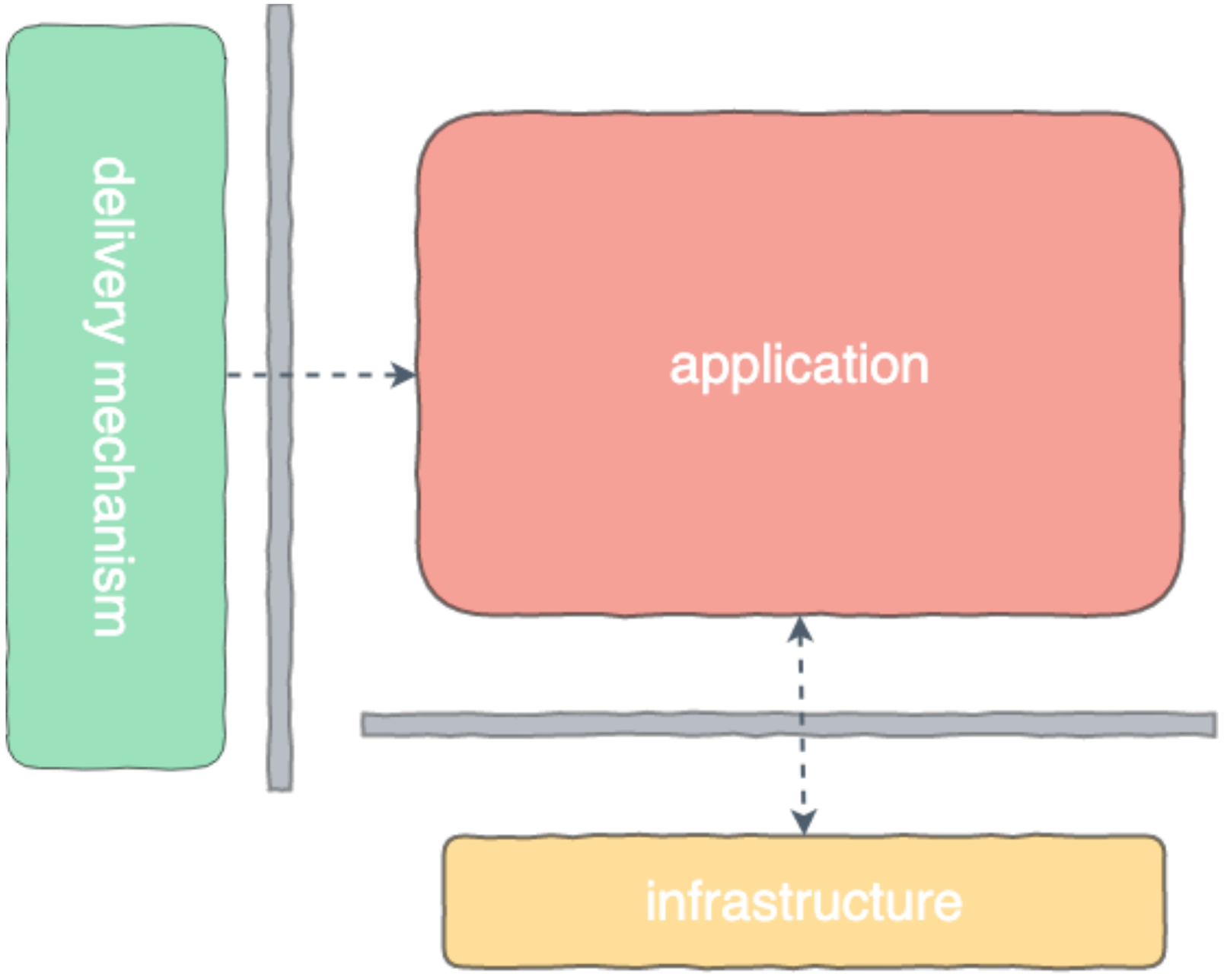
http://blog.firsthand.ca/2011/10/rails-is-not-your-application.html

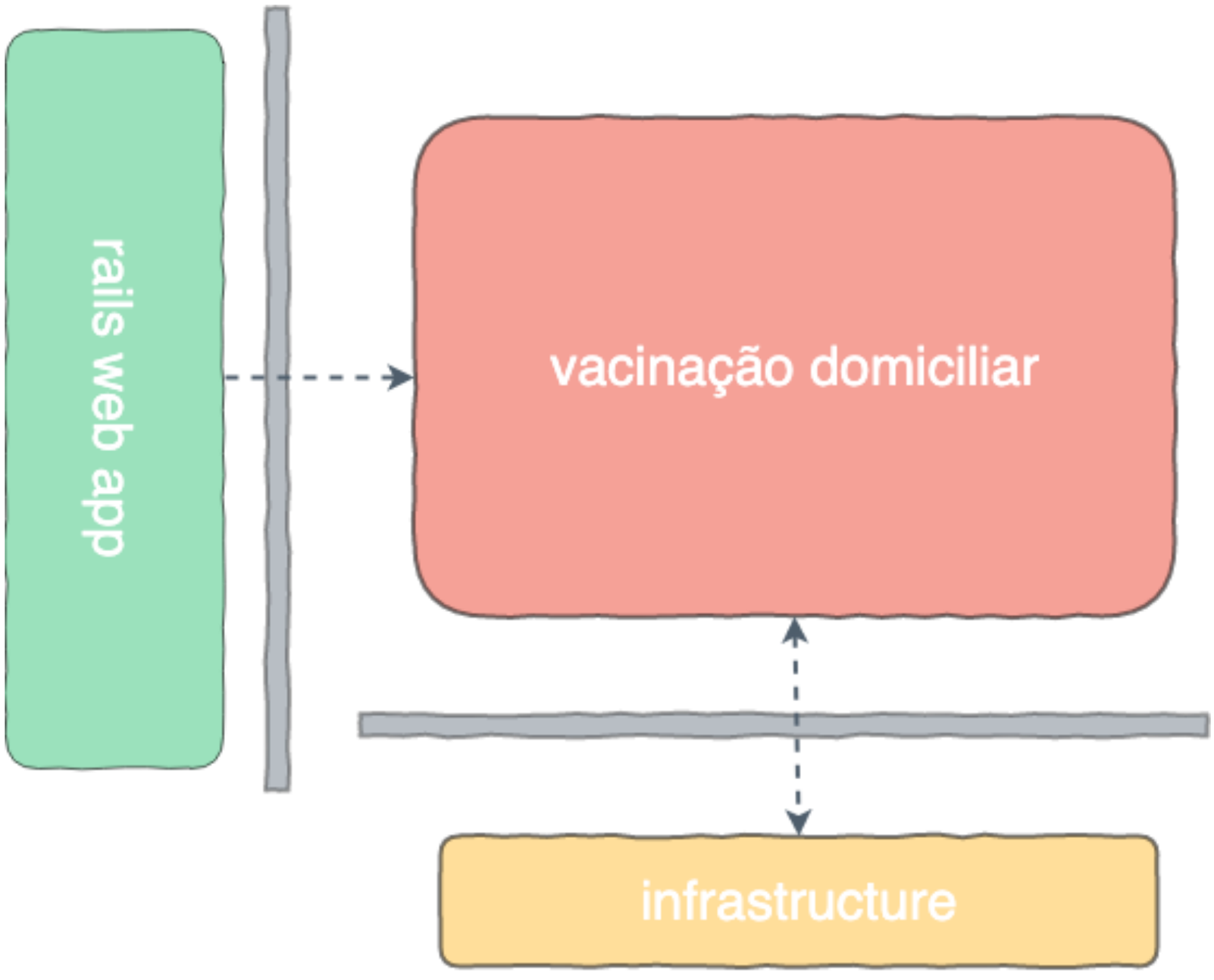
the web is a

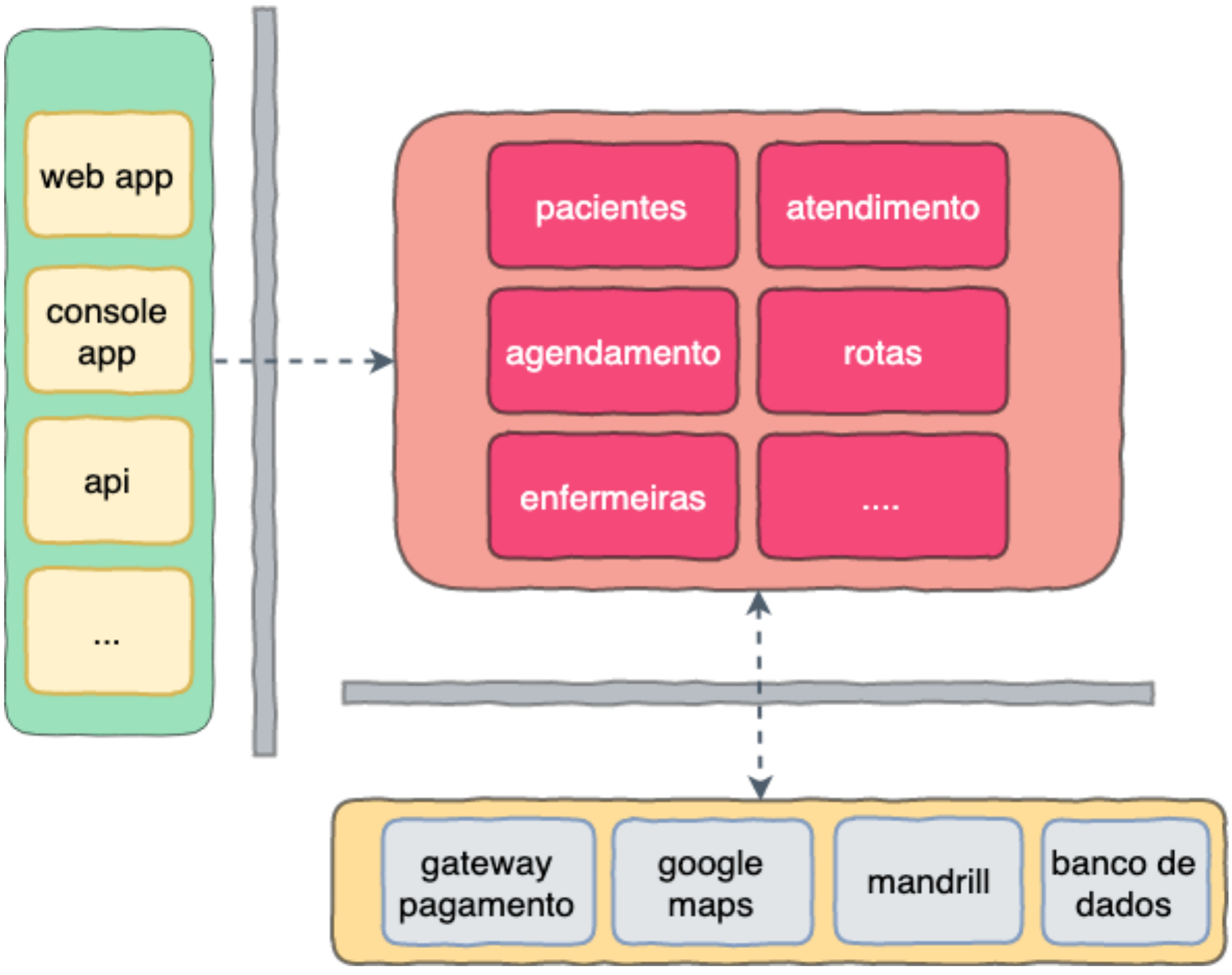
delivery mechanism

Architecture the lost years, Robert Martin

https://youtu.be/WpkDN78P884?t=581


delivery mechanism
application
infrastructure


rails web app
vacinação domiciliar
infrastructure

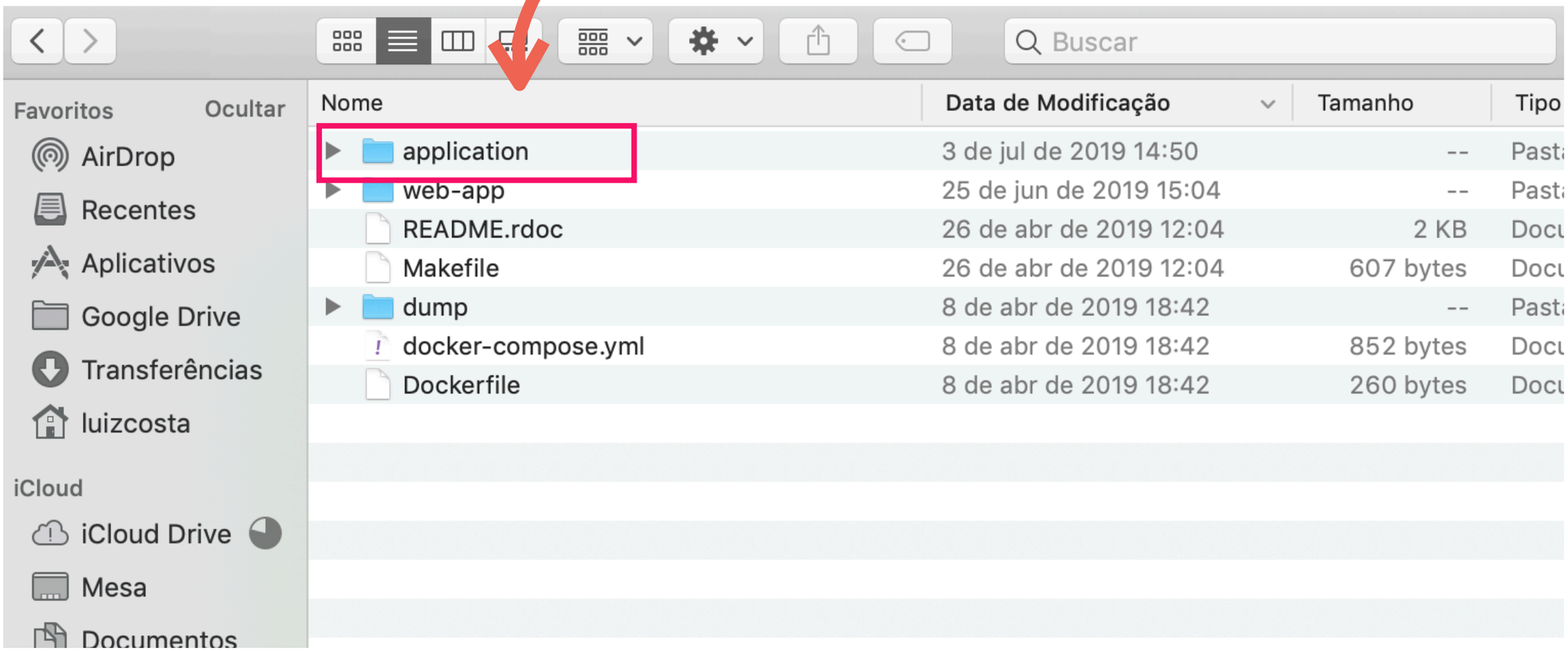

web app
console app
api
...
pacientes
atendimento
agendamento
rotas
enfermeiras
....
gateway pagamento
google maps
mandrill
banco de dados

como implementar
com rails?

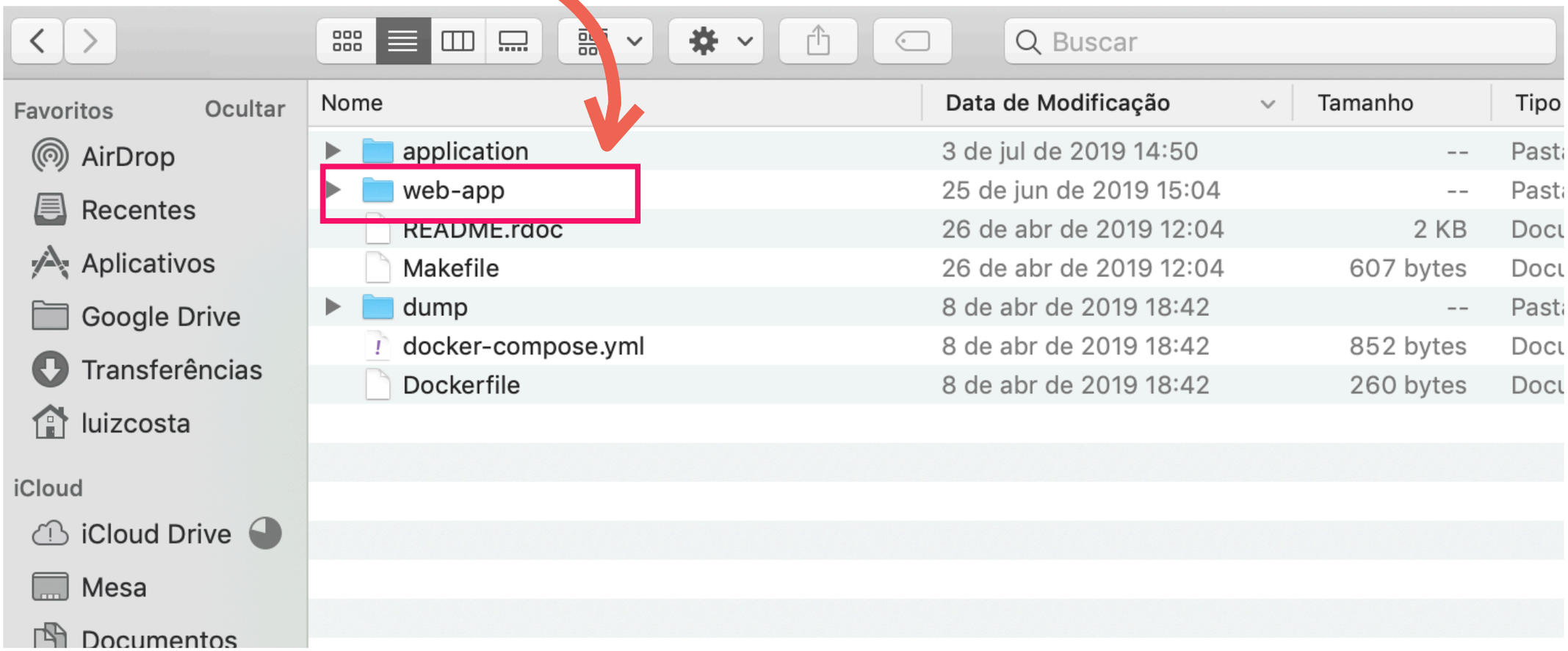
1

separar o código de negócio do framework

O código de negócio fica dentro da pasta application. Dentro da pasta, todo acesso ao framework é feito por adaptadores.



O código da aplicação web, escrita em rails, fica dentro de web app. Esta pasta é a implementação de um delivery mechanism. O código de negócio, que vive dentro de application, é acessado através de portas.



2

modularizar o código que vive dentro de application

alguns dos possíveis módulos de uma aplicação que resolve o problema de vacinação domiciliar

o mesmo objeto de domínio em diferentes contextos ou módulos

Atendimento

Paciente

+ qual nome, nascimento…?
+ qual dia do agendamento?
+ qual endereço de atendimento?

Vacinação

Paciente

+ como a caderneta de vacinação está organizada?
+ qual a próxima vacina?
+ qual vacina foi aplicada?

Financeiro

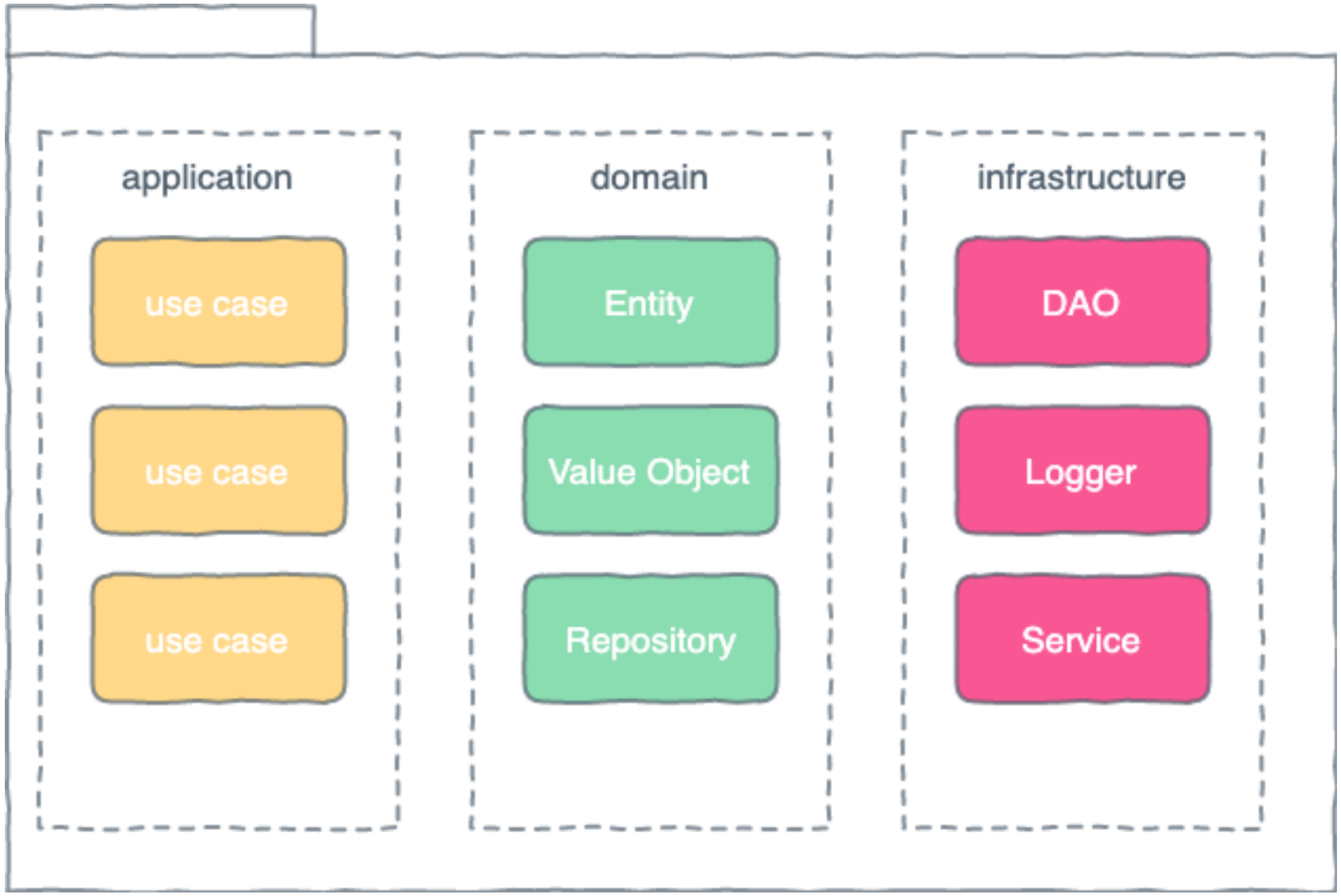
Paciente

+ qual custo das vacinas aplicadas?
+ alguma condição de desconto?
+ qual meio de pagamento utilizado?

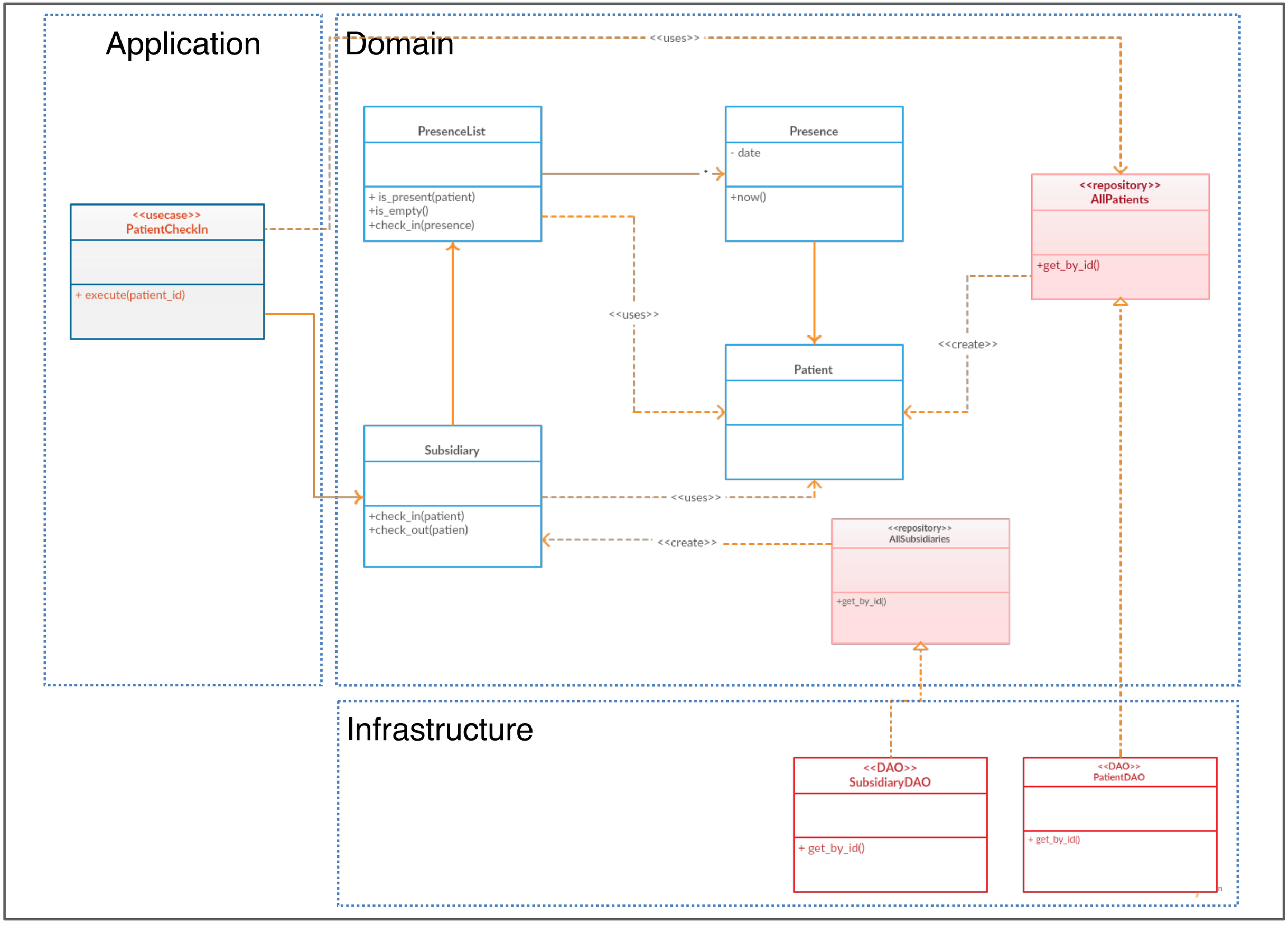
É responsabilidade de cada contexto modelar os dados da melhor maneira, de acordo com as suas responsabilidades.

cada módulo é como se fosse uma implementação do padrão hexagonal

anatomia de um módulo



Atendimento.Fila



talk is cheap

show me the code!

como é a
implementação de um
use case?

implementação de um use case

```ruby
class CheckInItems

  def self.build
    CheckInItems.new(
      change_inventory: Marketplace::Services::ChangeInventory.build,
      availability_checker: Marketplace::Services::AvailabilityChecker.build,
      event_publisher: EventManager::PublisherFactory.build_with_subscribers
    )
  end

  def initialize(change_inventory:, availability_checker:, event_publisher:nil)
    @change_inventory = change_inventory
    @event_publisher = event_publisher
    @availability_checker = availability_checker
  end

  def execute(items:)
    items.map do |item|
      inventory_item = Marketplace::Domain::InventoryItem.from(serialized_item: item)
      inventory_item = @change_inventory.check_in(inventory_item: inventory_item)
      @availability_checker.check(inventory_item: inventory_item)
      publish_events(inventory_item) unless @event_publisher.nil?
      inventory_item
    end
  end

  private

  def publish_events(inventory_item)
    event = Marketplace::Events::InventoryItemCheckedIn.new(inventory_item: inventory_item)
    @event_publisher.publish(event)
  end

end
end
```

Usa um **factory method*** para manter o encapsulamento e diminuir o acoplamento com o objeto cliente

As dependências são injetadas no construtor

O fluxo de execução é simples e limpo

application/src

* https://en.wikipedia.org/wiki/Factory_method_pattern

como o **delivery**
mechanism se conecta
com a **application?**

conectando a web app com o application

```ruby
class Marketplace::CheckInController < ApplicationController

  def create
    begin
      use_case = Marketplace::UseCases::CheckInItems.build
      items = use_case.execute(items: params[:items])
      render status: :ok, json: { status: 200, data: items}
    rescue Exception => e
      render_json_error status: 500, source: self, exception: e
    end
  end

end
```

Aqui o factory method é usado para instanciar o use case.

A interface pública do use case é usada como **Port** para acessar o código da application

`web-app/app/controllers`

* https://en.wikipedia.org/wiki/Factory_method_pattern

foco total no

domain model

domain model != model

objetos de domínio são escritos em código ruby puro (PORO*)

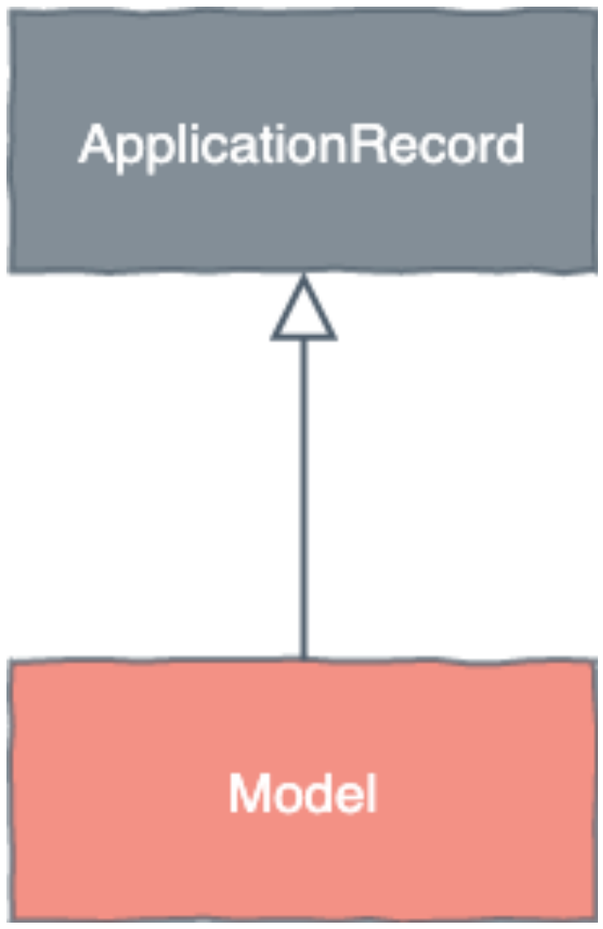
```ruby
module Marketplace
  module Domain
    class CheckoutItem
      attr_reader :inventory_item, :quantity

      def initialize(inventory_item:, quantity:, accepted: false)
        @inventory_item = inventory_item
        @quantity = quantity
        @accepted = accepted
      end

      def accepted?
        @accepted
      end

      def accept!
        @accepted = true
      end

      def not_accepted!
        @accepted = false
      end

      def item_available?
        self.inventory_item.enough_stock?(quantity: self.quantity)
      end

      def to_hash
        {
          inventory_item_id: nil || self.inventory_item.id,
          sales_item_id: self.inventory_item.sales_item_obj_id,
          quantity: self.quantity,
          accepted: self.accepted?,
          inventory_quantity: self.inventory_item.quantity
        }
      end
```

Não existe nenhuma relação direta com o Active Record

*http://blog.jayfields.com/2007/10/ruby-poro.html

e os models? Active Record?

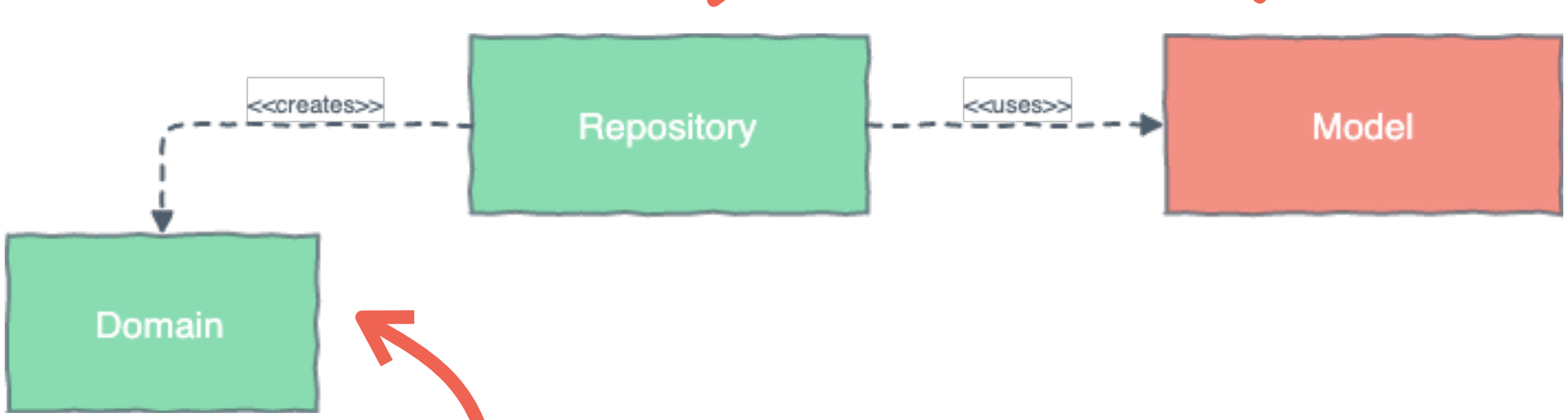
Nas implementações padrão do rails, o model é uma subclasse de **ApplicationRecord,** isso faz com que o acoplamento com o banco de dados seja bem alto.



O domain object é separado do Model e o seu ciclo de vida é controlado por um **Repositório*.**

O repositório é responsável por executar as consultas (queries), e mapear os objetos de domínio. Para isso pode ter a colaboração de uma Factory*.

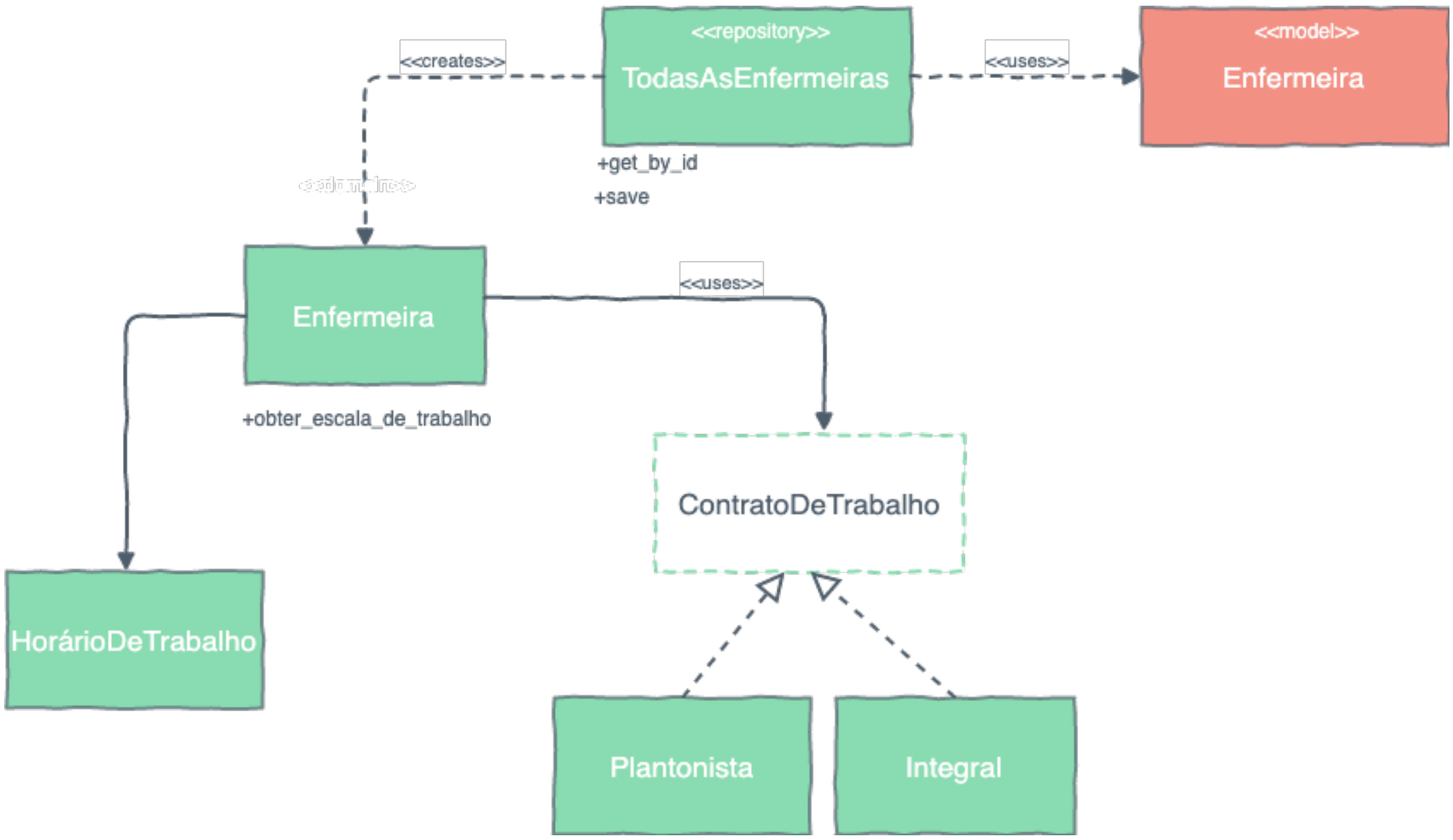
O Model tem a responsabilidade de garantir a consistência dos dados, de acordo com o modelo relacional. Pode definir algumas queries e ter validações de dados



O domain object é quem tem a implementação das lógicas de negócio.

*https://martinfowler.com/eaaCatalog/repository.html

Um único Model pode dar origem a um modelo de domínio bem mais complexo. Neste caso, todos os dados do domain model são persistidos na mesma tabela (Rails Model) do banco de dados.



implementação de um repositório

```ruby
module TeamAllocation
  module Repositories
    class AllNurses

      def get_by_id(id)
        nurse_model = ::Nurse.find(id)
        TeamAllocation::Factories::NurseFactory.build.create_from(nurse_model)
      end

      def get_active_nurses(macro_region_obj_id = nil)
        macro_region_obj_id = nil if macro_region_obj_id.to_i.zero?
        nurses_collection = ::Nurse.active_nurses(macro_region_obj_id)
        TeamAllocation::Factories::NurseFactory.build.create_from_list(nurses_collection)
      end

      def get_by_obj_id(obj_id:)
        begin
          nurse_model = ::Nurse.find_by(obj_id: obj_id)
          TeamAllocation::Factories::NurseFactory.build.create_from(nurse_model)
        rescue ActiveRecord::RecordNotFound => e
          raise Exceptions::NurseNotExists.new
        end
      end

      def save(nurse:)
        nurse_model = ::Nurse.find_or_initialize_by(id: nurse.id)
        nurse_model.from_domain(nurse)
        nurse_model.save
      end
    end
  end
end
```

Aqui o model é usado para tirar vantagem do Active Record. Uma factory é usada para fazer a construção do objeto de domínio.

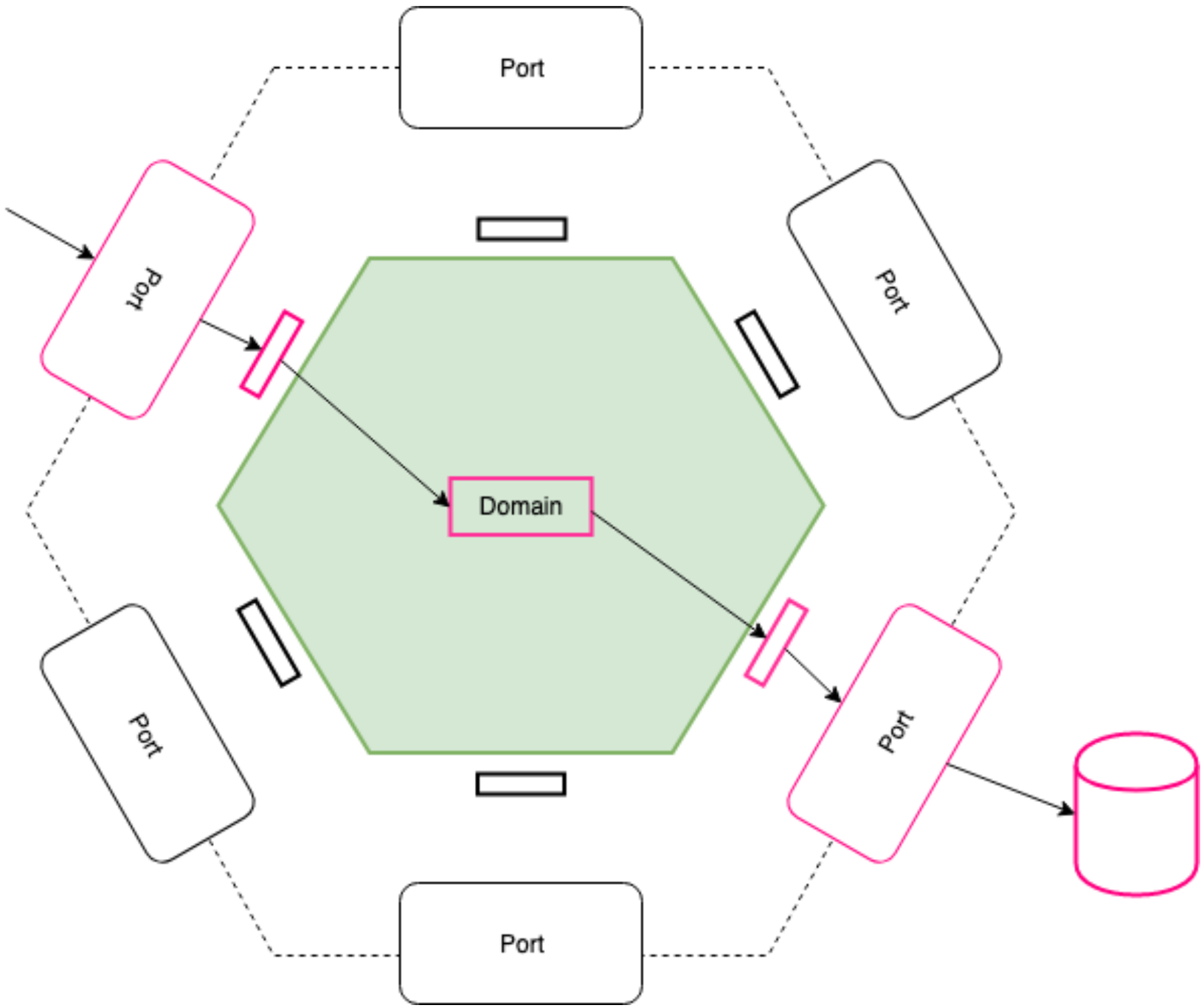
.active_nurses é uma query que está encapsulada no Model.

O save também delega para o Active Record.

onde estamos mesmo?

e a história dos hexágonos?

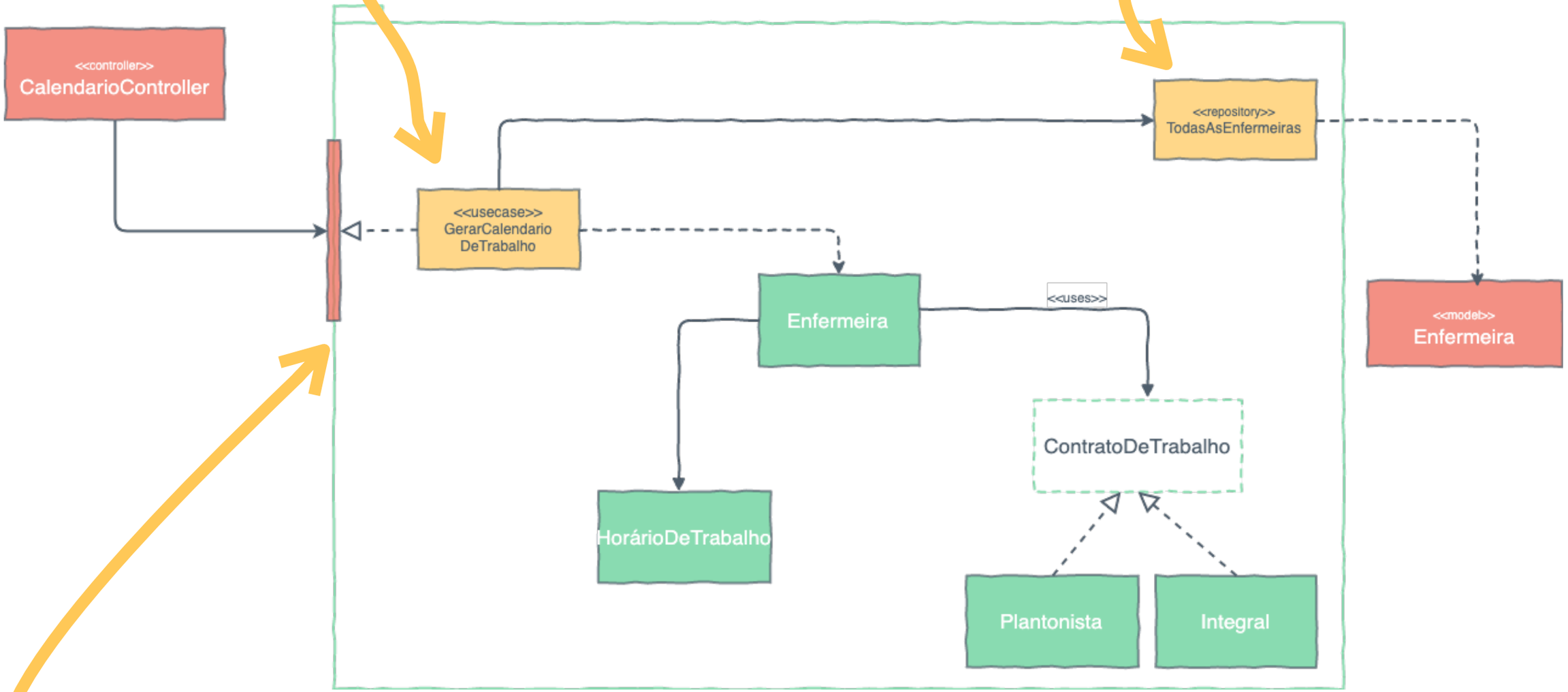
a partir da representação original…



para uma proposta de implementação

A implementação do caso de uso atua como um adaptador.

O repositório encapsula o acesso aos models, funcionando como mais um adaptador.

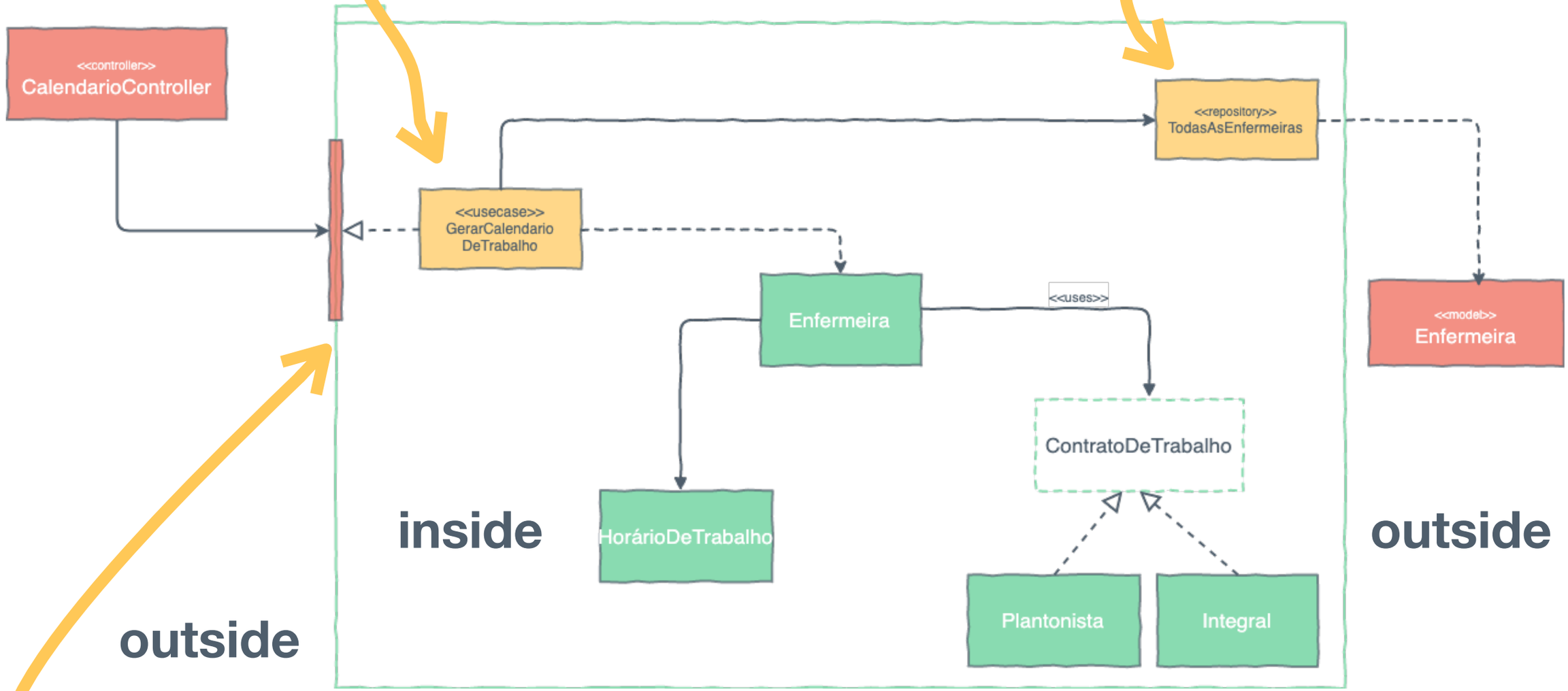


O controlador acessa os módulos através de uma porta que expõe a interface pública de um caso de uso.

para uma proposta de implementação

A implementação do caso de uso atua como um adaptador.

O repositório encapsula o acesso aos models, funcionando como mais um adaptador.



O controlador acessa os módulos através de uma porta que expõe a interface pública de um caso de uso.

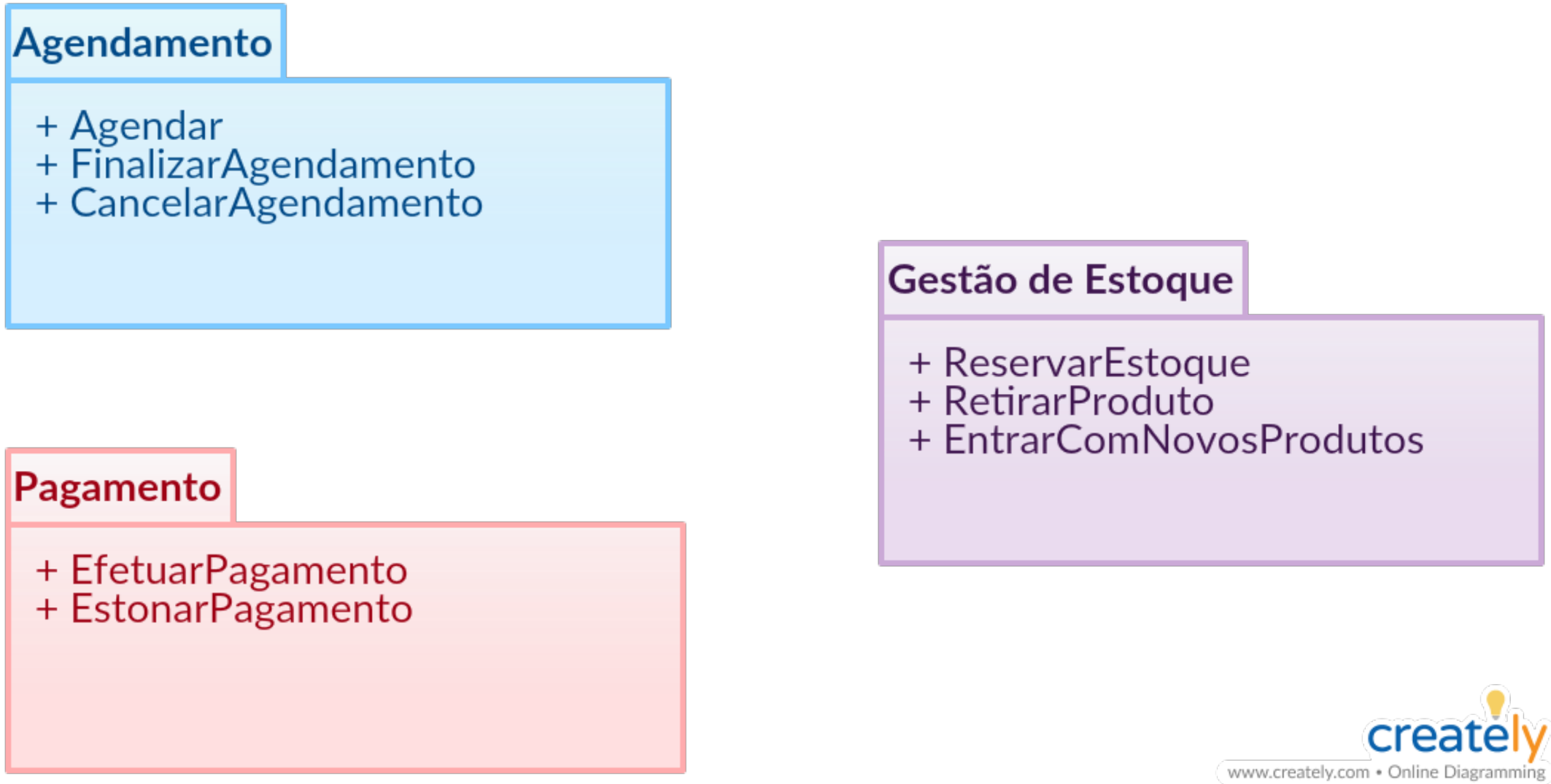
implementando uma User Story

Deve permitir ao paciente agendar a aplicação de vacina.

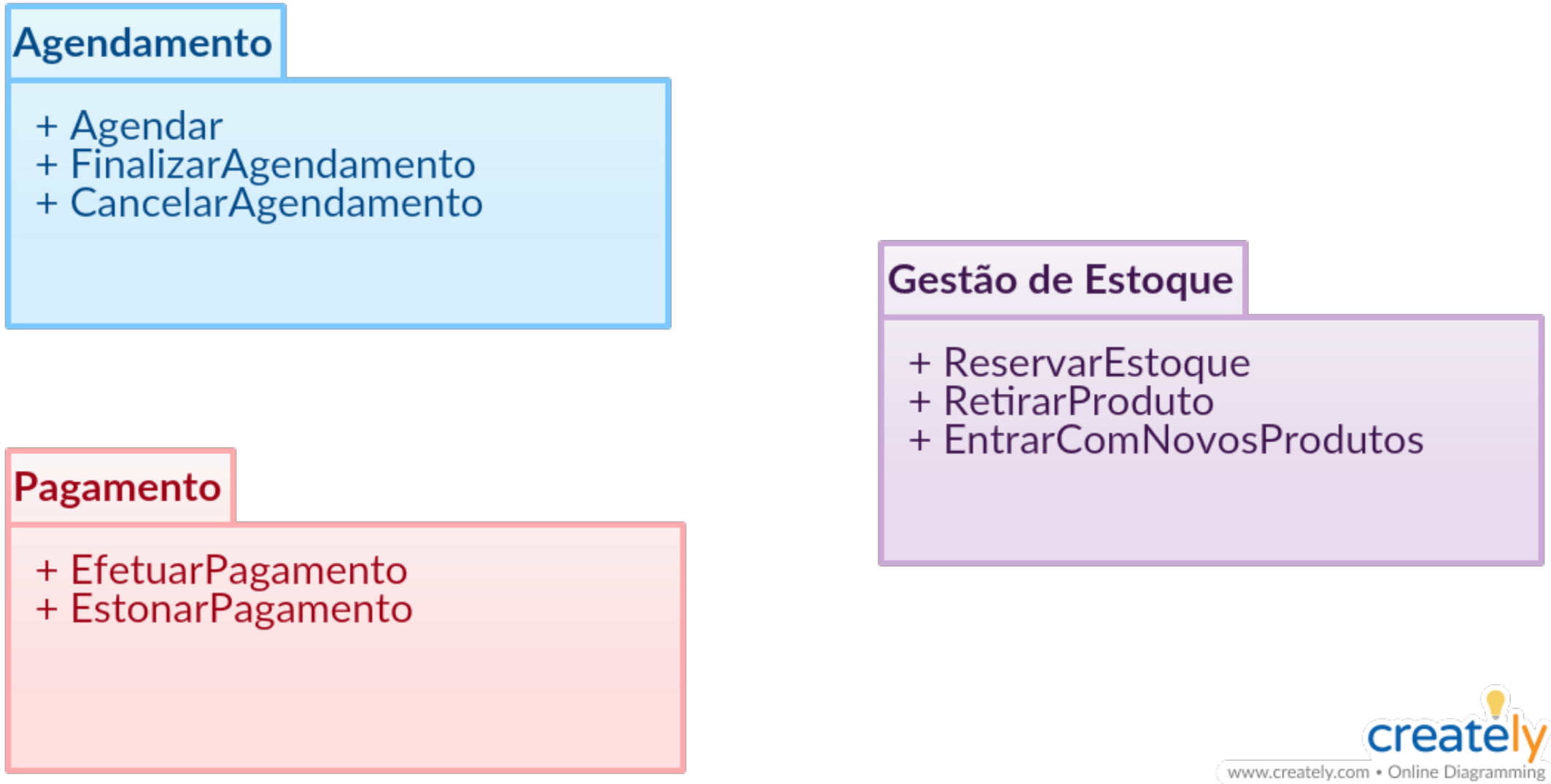
Ao fazer o agendamento deve-se efetuar o pagamento e reservar o estoque dos produtos agendados.

Deve permitir ao paciente agendar a aplicação de vacina.
Ao fazer o agendamento deve-se efetuar o pagamento e reservar o estoque dos produtos agendados.

módulos



~~módulos~~ bounded contexts

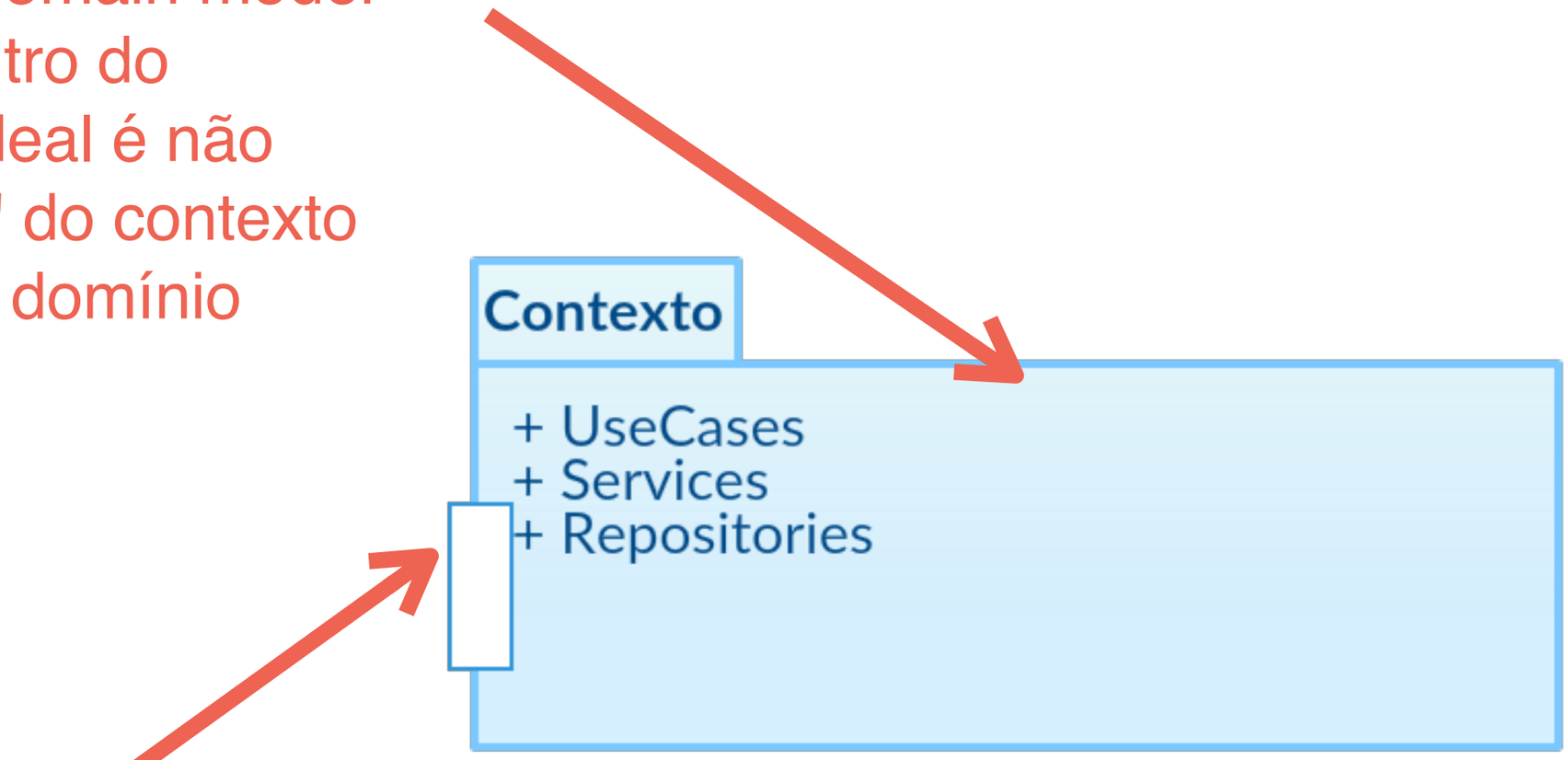


como um contexto executa uma ação em outro contexto?

boundaries

https://martinfowler.com/bliki/ApplicationBoundary.html

Mantenha o domain model protegido dentro do contexto. O ideal é não deixar “vazar" do contexto os objetos de domínio



O ideal é que um contexto só exponha objetos de fronteira. Ex: Use Cases, Services ou Repositories

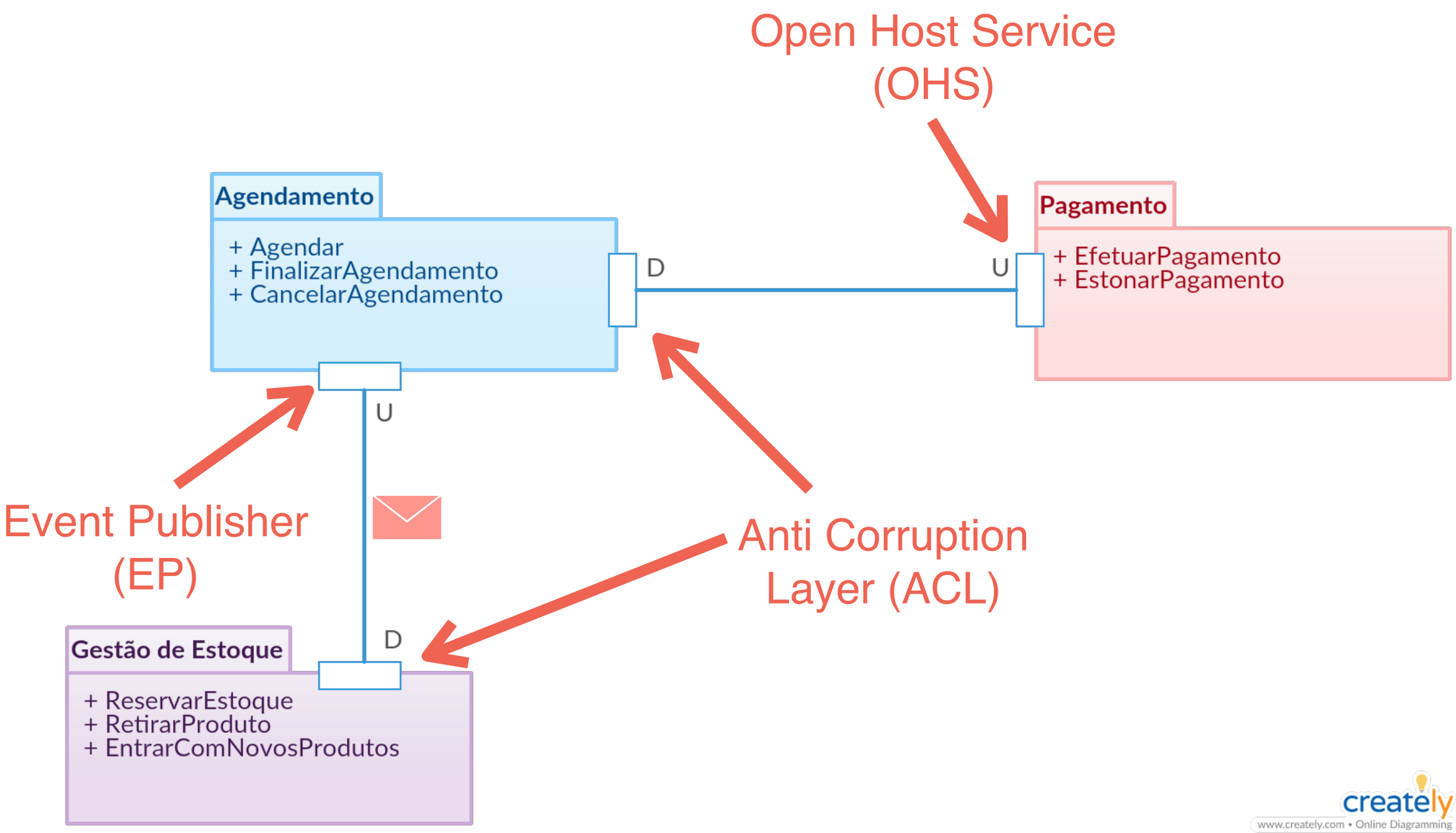
Se precisar retornar um conjunto de dados mais complexo, dê preferência para Hashs ou Tuplas

public class

deve-se reduzir o número de classes públicas para promover o encapsulamento

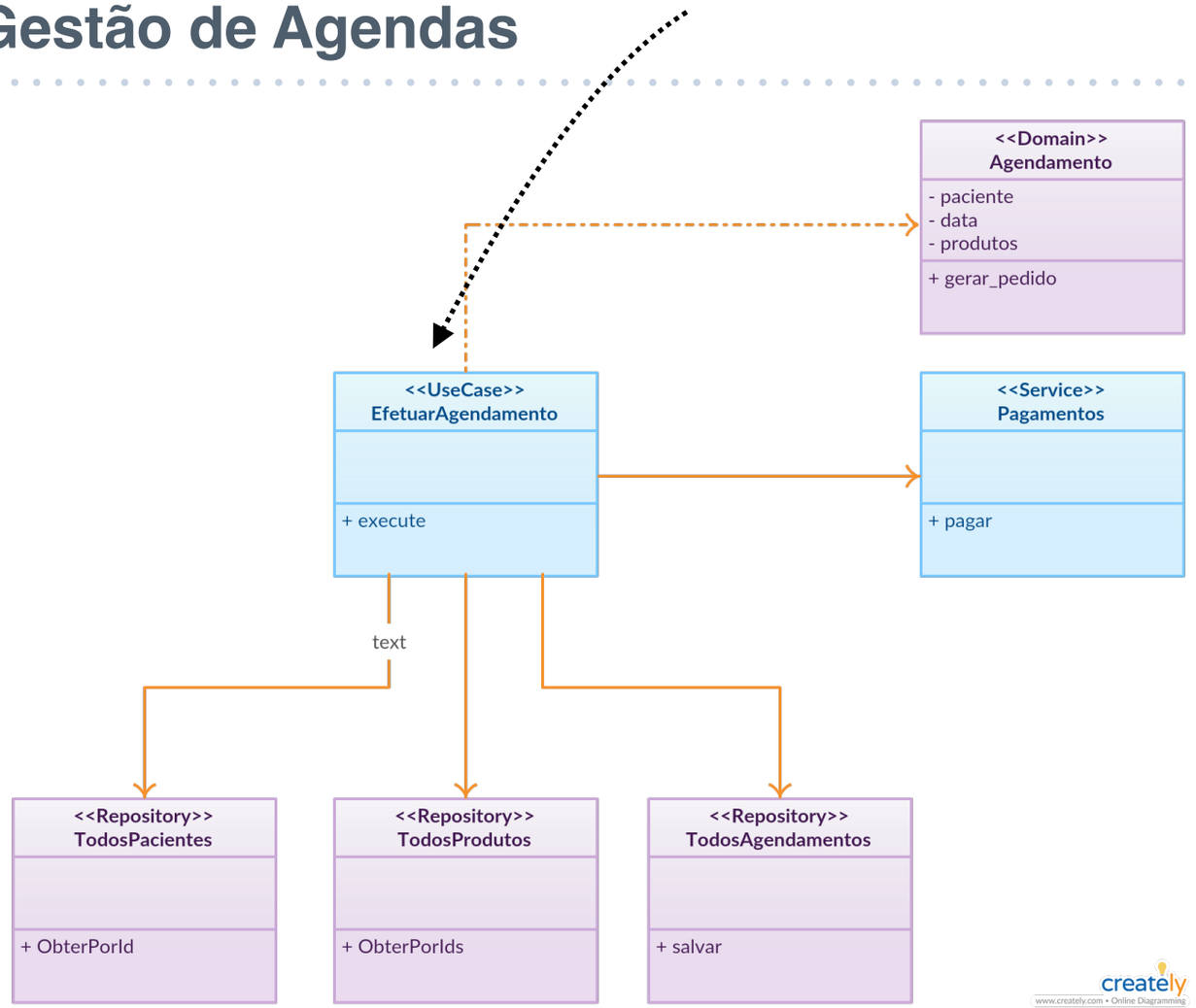
exemplo

context maps



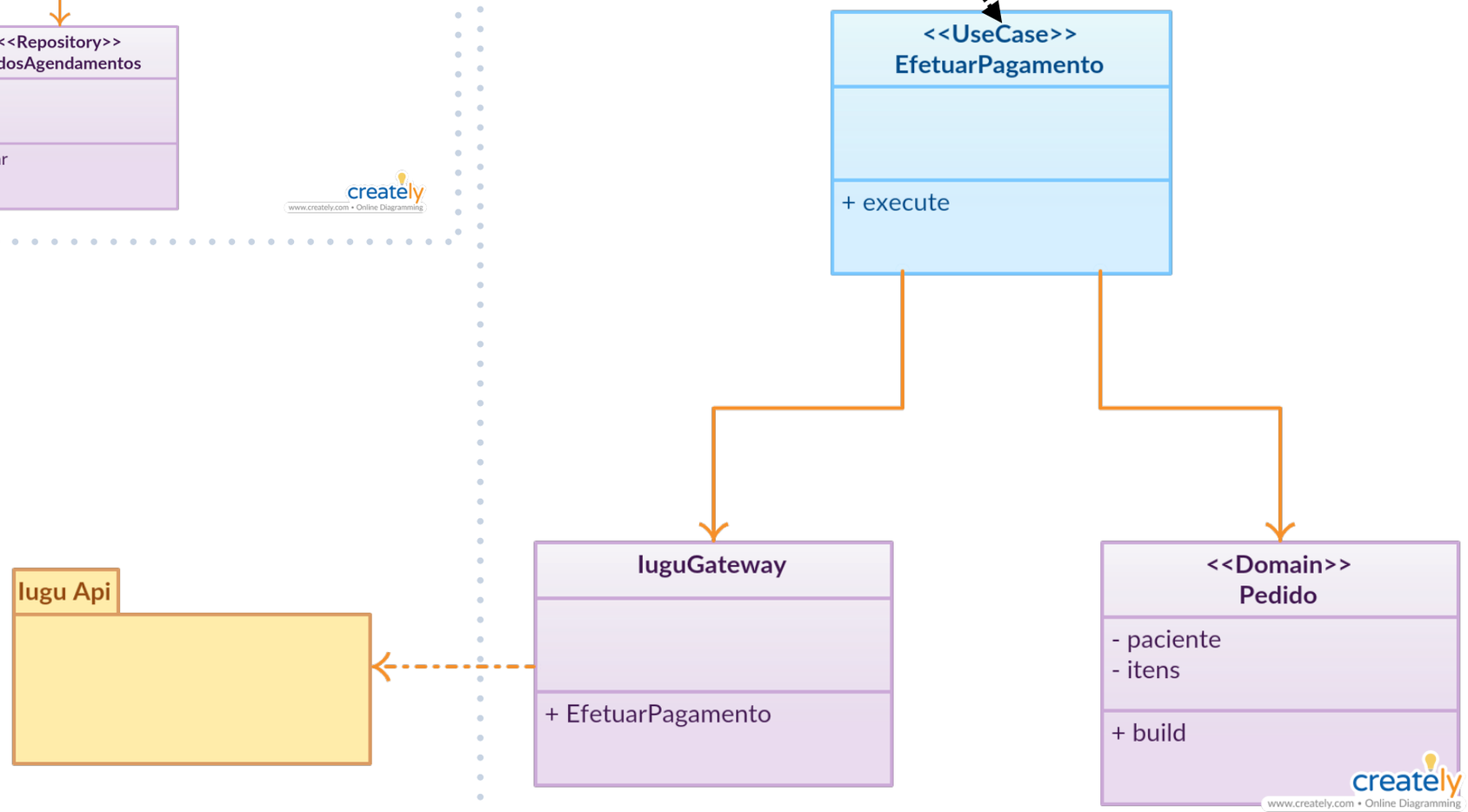
1

Gestão de Agendas



Neste caso os contextos se falam através dos objetos de fronteira: Services e Use Case

Pagamentos



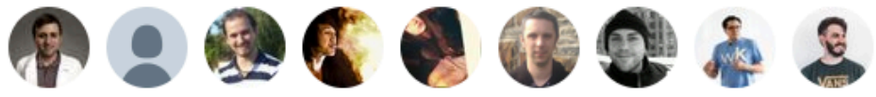
Uncle Bob Martin
@unclebobmartin

Following

A good architecture allows you to defer framework decisions. A good architecture allows frameworks to act as plugins to the app.

4:17 PM - 26 Sep 2011

102 Retweets **27** Likes



9 102 27

construa uma arquitetura que te permita atrasar as decisões

https://8thlight.com/blog/uncle-bob/2011/11/22/Clean-Architecture.html

Indireção para o contexto de pagamento

```ruby
module GestaoDeAgenda
  module CasosDeUso
    class EfetuarAgendamento

      def build
        EfetuarAgendamento.new(
          todos_paciente: GestaoDeAgenda::Repositorios::TodosPacientes.new,
          todos_produtos: GestaoDeAgenda::Repositorios::TodosProdutos.new,
          todos_agendamentos: GestaoDeAgenda::Repositorios::TodosAgendamentos.new,
          servico_de_pagamento: GestaoDeAgenda::Servicos::Pagamentos.build
        )
      end

      def initialize(todos_pacientes:, todos_produtos, todos_agendamentos:, servico_de_pagamento:)
        @todos_pacientes = todos_pacientes
        @todos_produtos = todos_produtos
        @servico_de_pagamento = servico_de_pagamento
        @todos_agendamentos = @todos_agendamentos
      end
```

Injeção de Dependências

efetuar agendamento

```ruby
def execute(id_paciente:, data_agendamento:, ids_de_produto:)
  paciente      = @todos_pacientes.obter_por_id(id_paciente)
  produtos      = @todos_produtos.obter_por_ids(ids_de_produto)
  agendamento   = GestaoDeAgenda::Dominio::Agendamento.para(paciente, data_agendamento, ids_de_produto)
  pedido        = agendamento.gerar_pedido
  @servico_de_pagamento.pagar(pedido)
  @todos_agendamentos.salvar(agendamento)
  publicar_eventos(agendamento)
end
```

```ruby
module GestaoDeAgenda
  module Servicos
    class Pagamentos

      def pagar(pedido:)
        begin
          uc     = Pagamentos::CasosDeUso::EfetuarPagamento.build
          pedido = GestaoDeAgenda::Tradutores::TradutorDePedidos.traduzir(pedido)
          uc.execute(pedido: pedido)
        rescue Pagamentos::Excecoes::PagamentoRecusado => e
          raise GestaoDeAgenda::Excecoes::ErroNoPagamento.new("[ Pagamentos ] - Pagamento r
        end
      end
    end
  end
end
```

Único ponto de contato com contexto de pagamento

efetuar agendamento

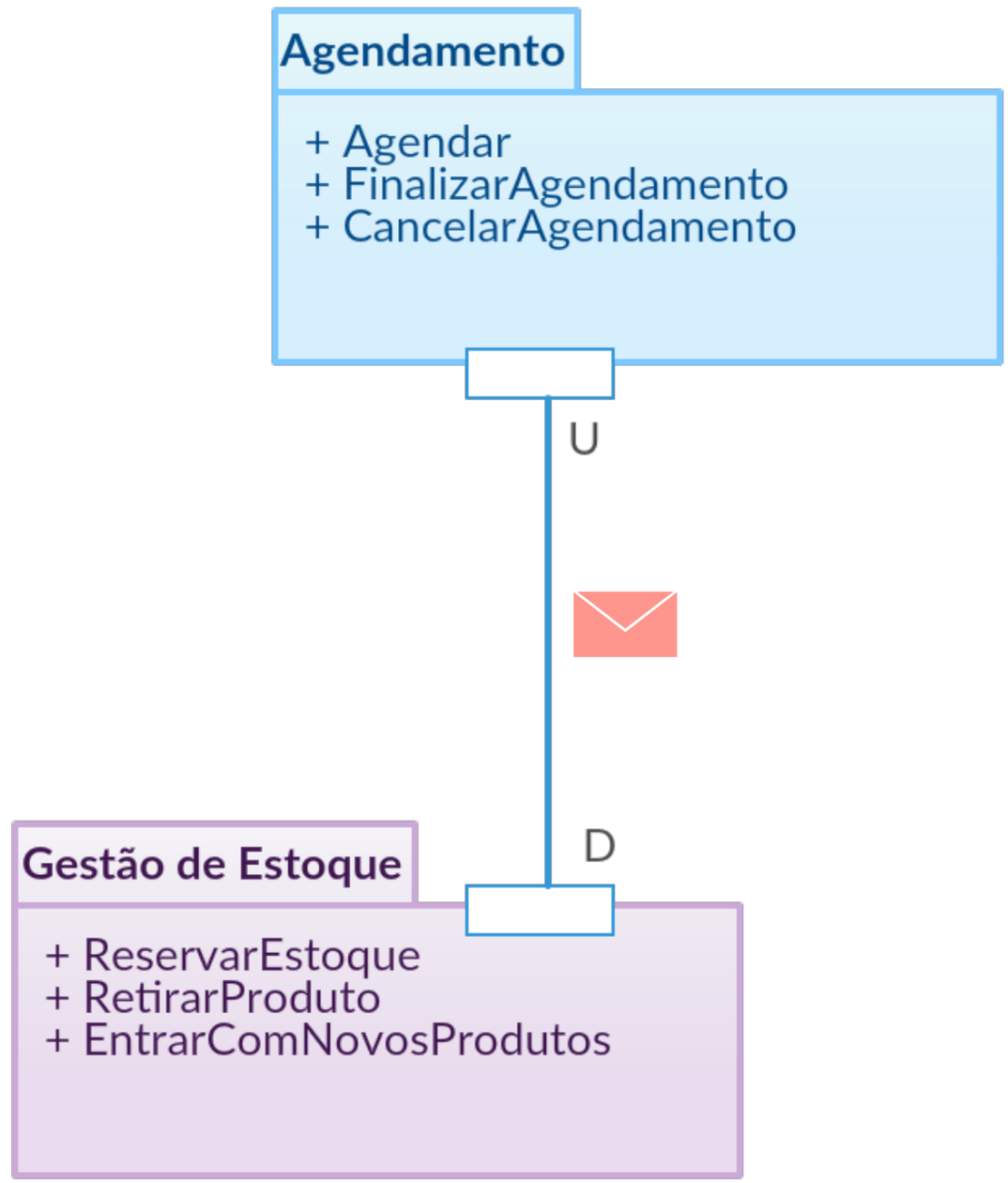
```ruby
def execute(id_paciente:, data_agendamento:, ids_de_produto:)
  paciente      = @todos_pacientes.obter_por_id(id_paciente)
  produtos      = @todos_produtos.obter_por_ids(ids_de_produto)
  agendamento   = GestaoDeAgenda::Dominio::Agendamento.para(
  pedido        = agendamento.gerar_pedido
  @servico_de_pagamento.pagar(pedido)
  @todos_agendamentos.salvar(agendamento)
  publicar_eventos(agendamento)
end
```

```ruby
# {
#   patient_id: 109,
#   date: 20/07/2018 13:20,
#   order_items: [
#     {product_id: 22, price: 340.00},
#     {product_id: 12, price: 234.30}
#   ]
# }
```

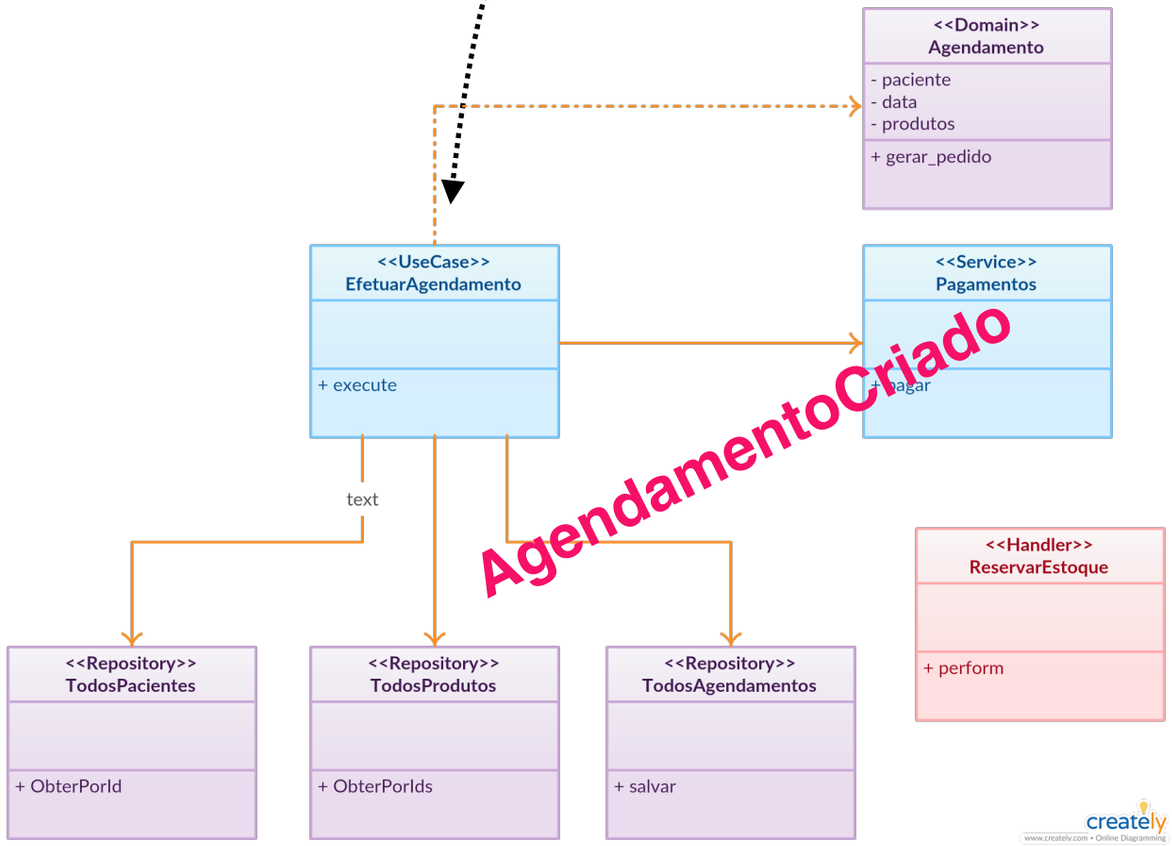
```ruby
module GestaoDeAgenda
  module Servicos
    class Pagamentos

      def pagar(pedido:)
        begin
          uc     = Pagamentos::CasosDeUso::EfetuarPagamento.build
          pedido = GestaoDeAgenda::Tradutores::TradutorDePedidos.traduzir(pedido)
          uc.execute(pedido: pedido)
        rescue Pagamentos::Excecoes::PagamentoRecusado => e
          raise GestaoDeAgenda::Excecoes::ErroNoPagamento.new("[ Pagamentos ] - Pagamento r
        end
      end
    end
  end
end
```

Tradução do modelo entre os dois contextos

2

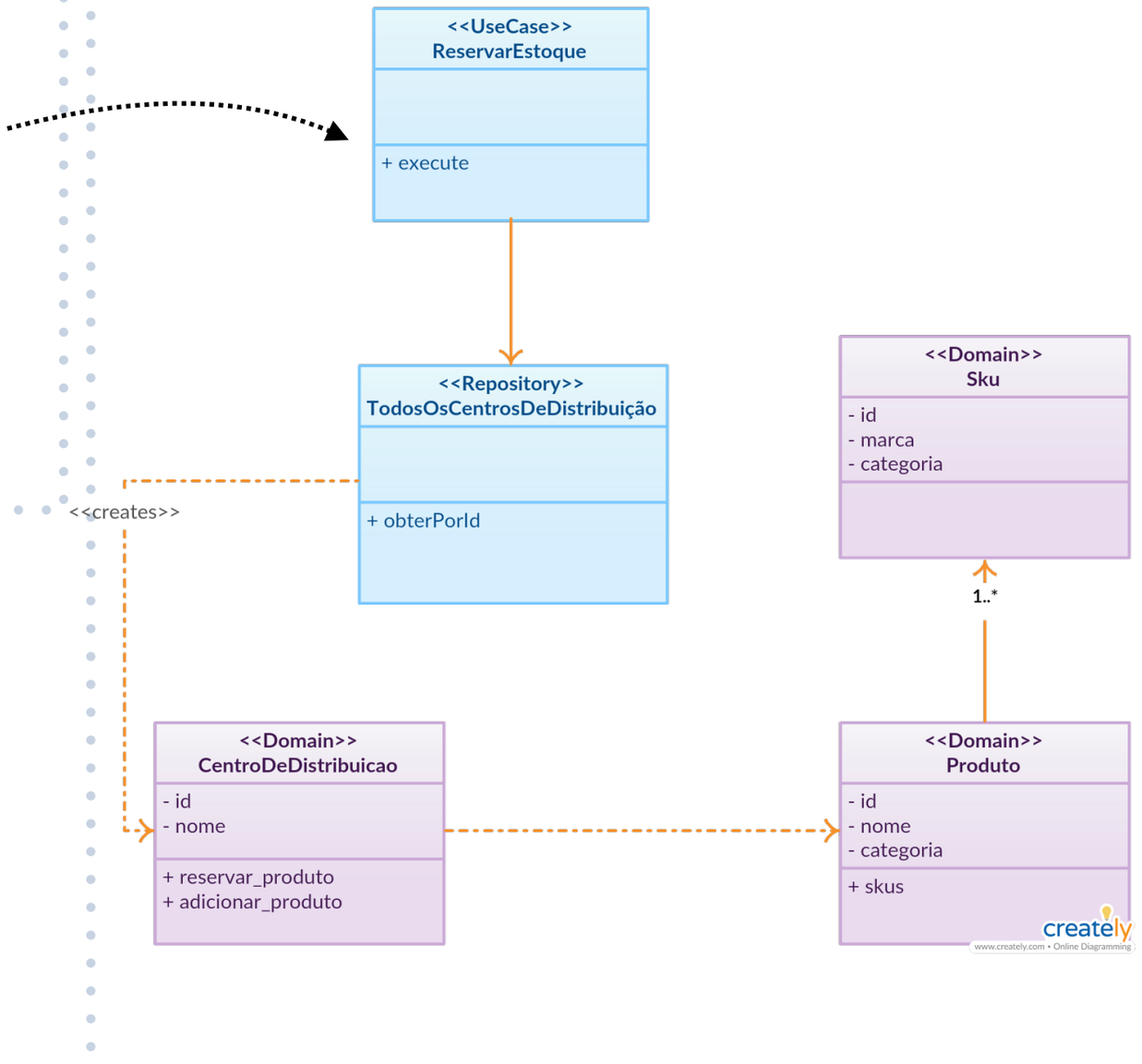

Agendamento
+ Agendar
+ FinalizarAgendamento
+ CancelarAgendamento
U
D
Gestão de Estoque
+ ReservarEstoque
+ RetirarProduto
+ EntrarComNovosProdutos

Gestão de agendas



Os dois contextos se falam através de um domain event

Gestão de estoque



```ruby
def execute(id_paciente:, data_agendamento:, ids_de_produto:)
  paciente      = @todos_pacientes.obter_por_id(id_paciente)
  produtos      = @todos_produtos.obter_por_ids(ids_de_produto)
  agendamento   = GestaoDeAgenda::Dominio::Agendamento.para(paciente, data_agendamento, ids_de_produto)
  pedido        = agendamento.gerar_pedido
  @servico_de_pagamento.pagar(pedido)
  @todos_agendamentos.salvar(agendamento)
  publicar_eventos(agendamento)
end
```

```ruby
private

def publicar_eventos(agendamento)
  event = GestaoDeAgenda::Eventos::AgendamentoCriado.new(agendamento)
  GestorDeEventos::Publicador.publicar(event)
end
```

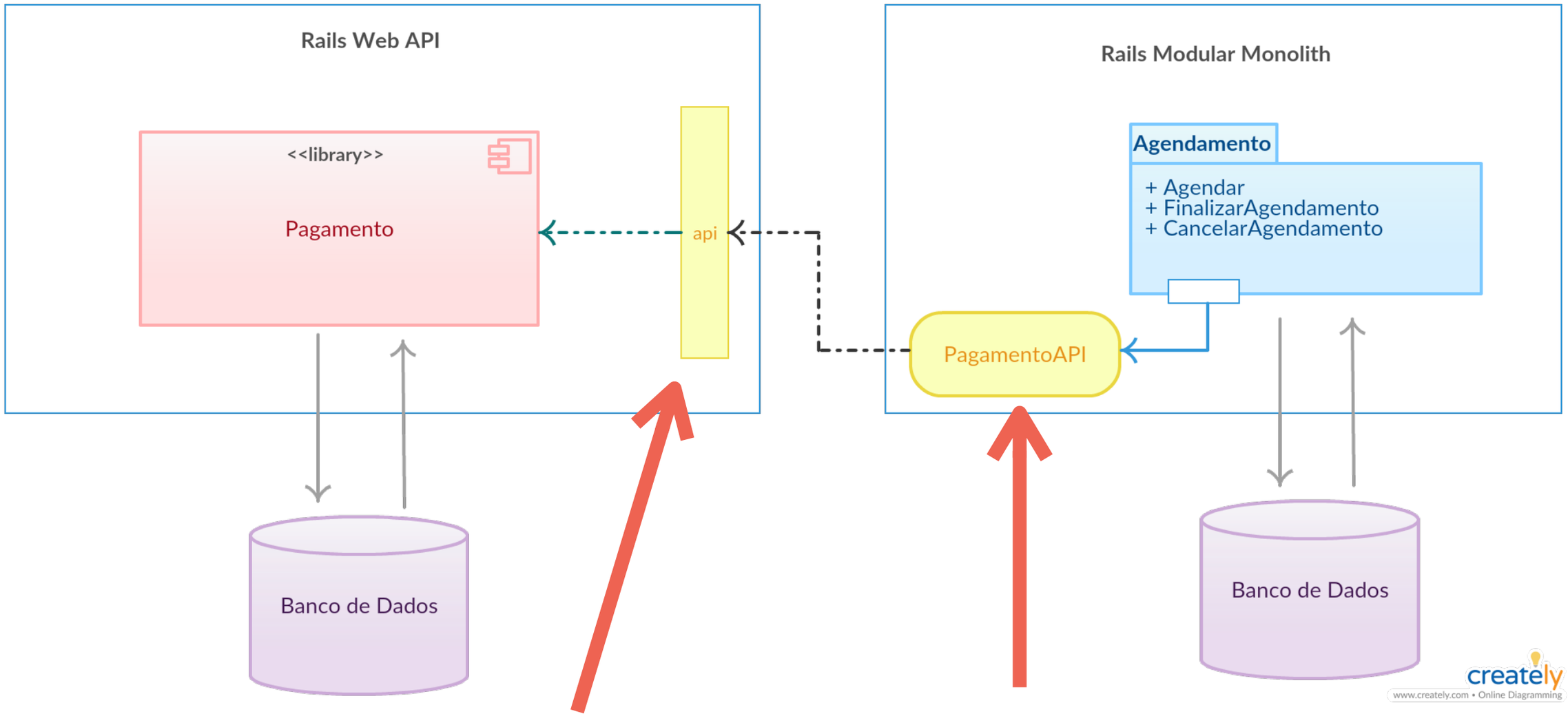
Publica o evento através da EP

```ruby
module Agendamento
  module Handlers
    class ReservarEstoque

      def perform(dados_do_evento:)
        ids_de_produto = dados_do_evento[:ids_de_produto]
        uc = GestaoDeEstoque::CasosDeUso::ReservarEstoque.build
        uc.execute(ids_de_produto: ids_de_produto)
      end
    end
  end
```

Contato com o contexto de gestão de estoque

como tirar vantagem e extrair para um microsserviço?

contexto de pagamento como microsserviço

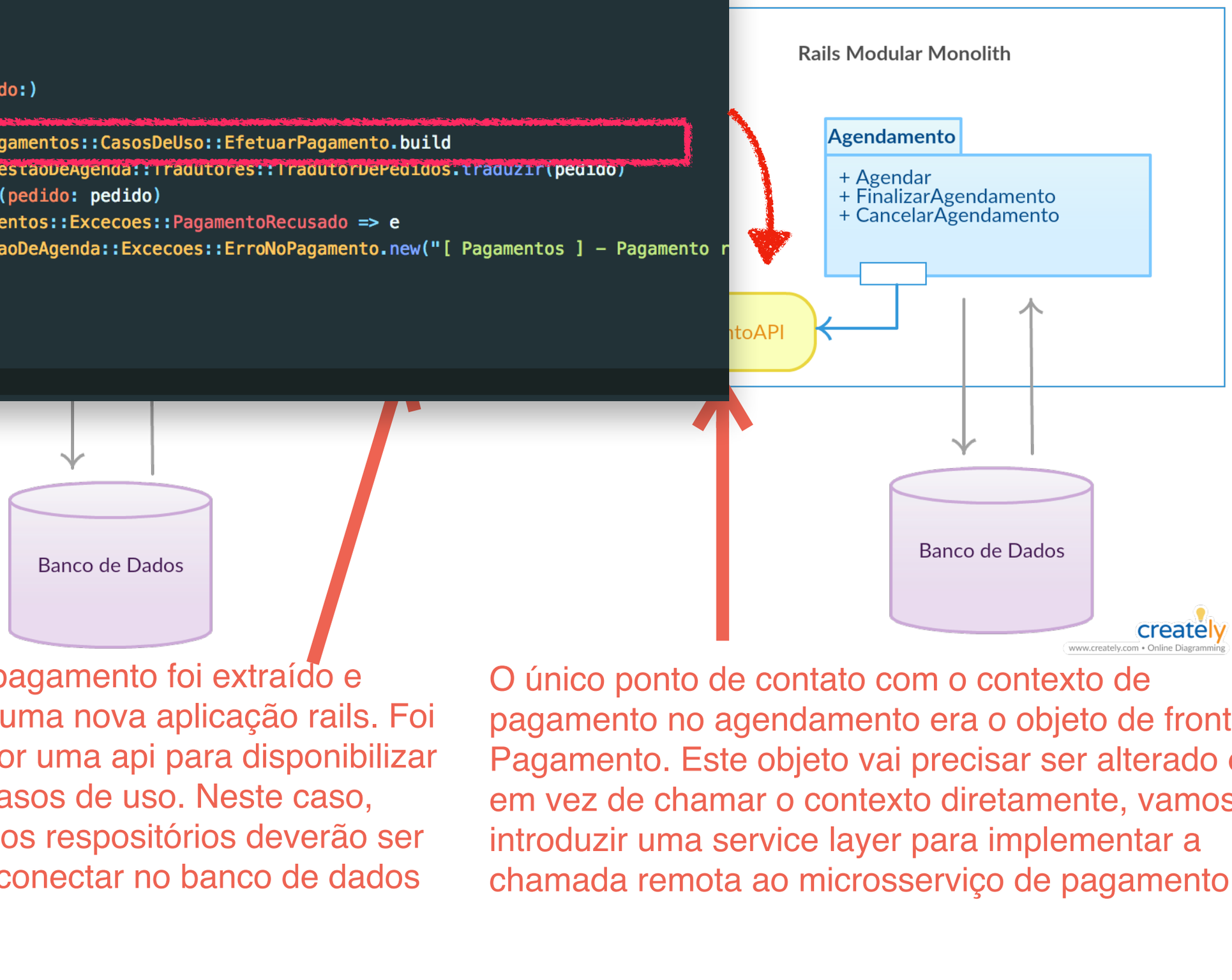


O contexto de pagamento foi extraído e adicionado em uma nova aplicação rails. Foi necessário expor uma api para disponibilizar o acesso aos casos de uso. Neste caso, provavelmente os respositórios deverão ser alterados para conectar no banco de dados diferente.

O único ponto de contato com o contexto de pagamento no agendamento era o objeto de fronteira Pagamento. Este objeto vai precisar ser alterado e, em vez de chamar o contexto diretamente, vamos introduzir uma service layer para implementar a chamada remota ao microsserviço de pagamento

contexto de pagamento como microsserviço

```ruby
module GestaoDeAgenda
  module Servicos
    class Pagamentos

      def pagar(pedido:)
        begin
          uc     = Pagamentos::CasosDeUso::EfetuarPagamento.build
          pedido = GestaoDeAgenda::Tradutores::TradutorDePedidos.traduzir(pedido)
          uc.execute(pedido: pedido)
        rescue Pagamentos::Excecoes::PagamentoRecusado => e
          raise GestaoDeAgenda::Excecoes::ErroNoPagamento.new("[ Pagamentos ] - Pagamento r
        end
      end
    end
  end
end
```



O contexto de pagamento foi extraído e adicionado em uma nova aplicação rails. Foi necessário expor uma api para disponibilizar o acesso aos casos de uso. Neste caso, provavelmente os respositórios deverão ser alterados para conectar no banco de dados diferente.

O único ponto de contato com o contexto de pagamento no agendamento era o objeto de fronteira Pagamento. Este objeto vai precisar ser alterado e, em vez de chamar o contexto diretamente, vamos introduzir uma service layer para implementar a chamada remota ao microsserviço de pagamento

como extrair o contexto de Gestão de Estoque?

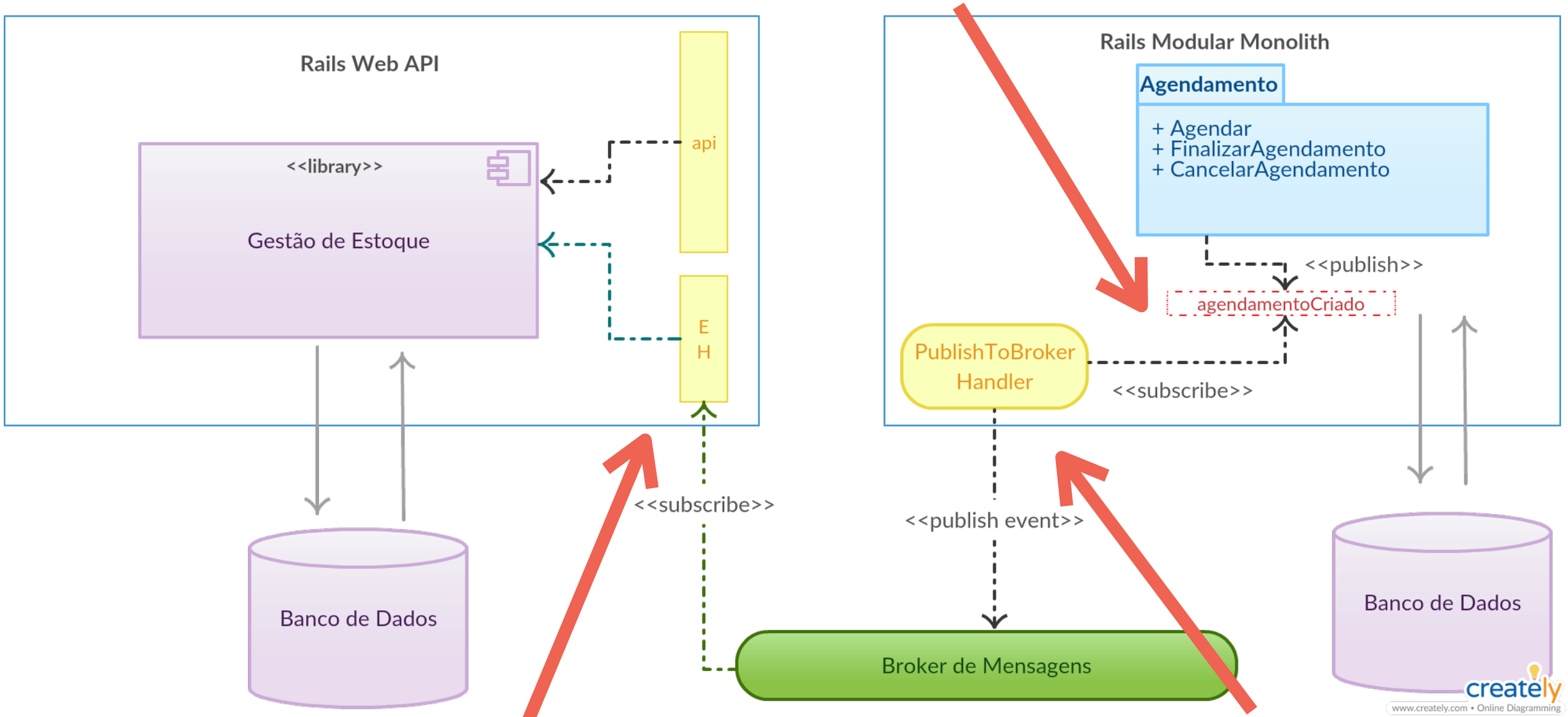
Como lidar com o Domain Event AgendamentoCriado?

publish/subscribe ou observers

https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern
https://en.wikipedia.org/wiki/Observer_pattern

contexto de estoque como microsserviço

O contexto de agendamento continua publicando o domain event AgendamentoCriado da mesma forma que antes, nada muda aqui.



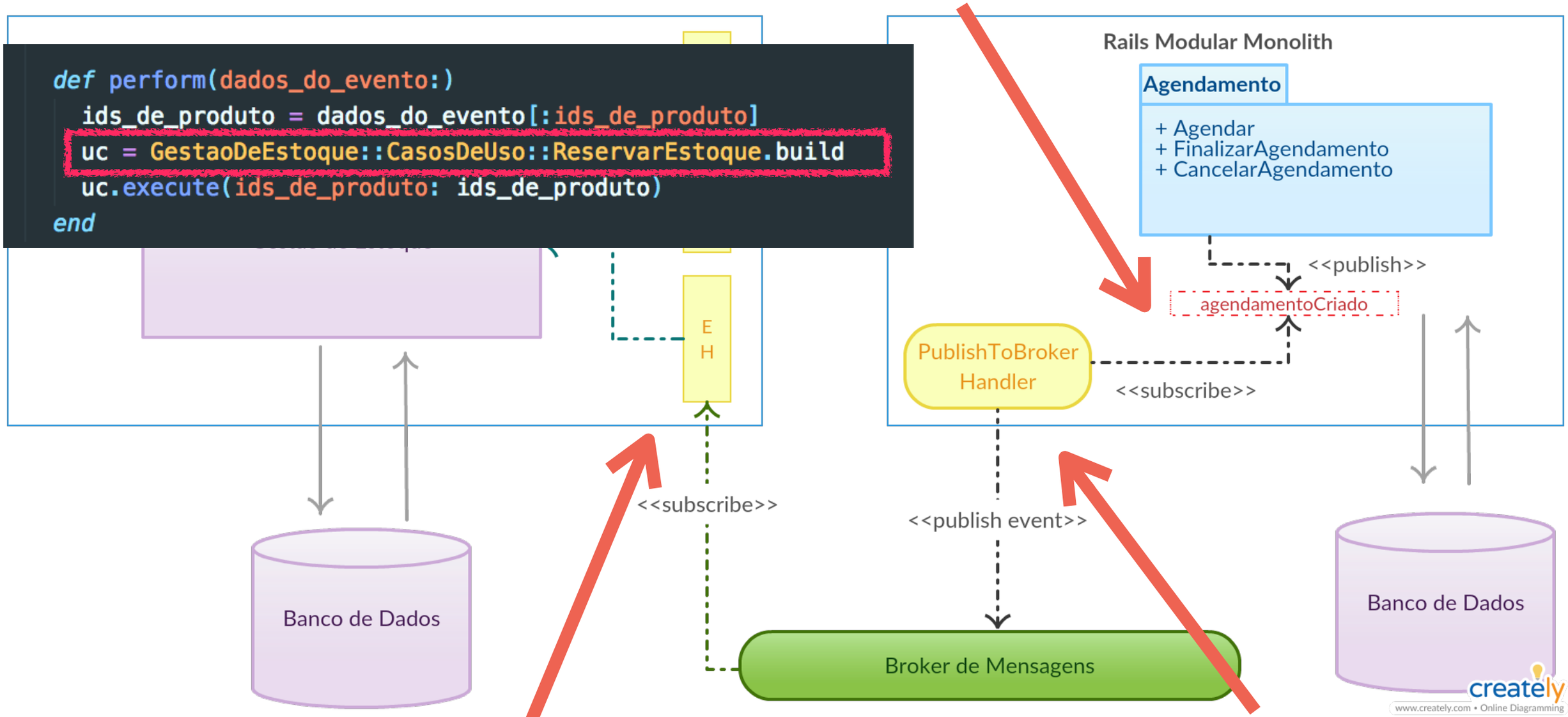
No microsserviço de estoque, é necessário adicionar na ACL, handlers que serão estimulados pelas mensagens entregues pelo broker. Estes handlers, delegam a execução para os caso de uso.

O Event Handler original é substituído por outro que publica o evento em um Broker de mensagem. Ex: RabbitMQ, Kafka, ActveMQ

contexto de estoque como microsserviço

O contexto de agendamento continua publicando o domain event AgendamentoCriado da mesma forma que antes, nada muda aqui.

```ruby
def perform(dados_do_evento:)
  ids_de_produto = dados_do_evento[:ids_de_produto]
  uc = GestaoDeEstoque::CasosDeUso::ReservarEstoque.build
  uc.execute(ids_de_produto: ids_de_produto)
end
```



No microsserviço de estoque, é necessário adicionar na ACL, handlers que serão estimulados pelas mensagens entregues pelo broker. Estes handlers, delegam a execução para os caso de uso.

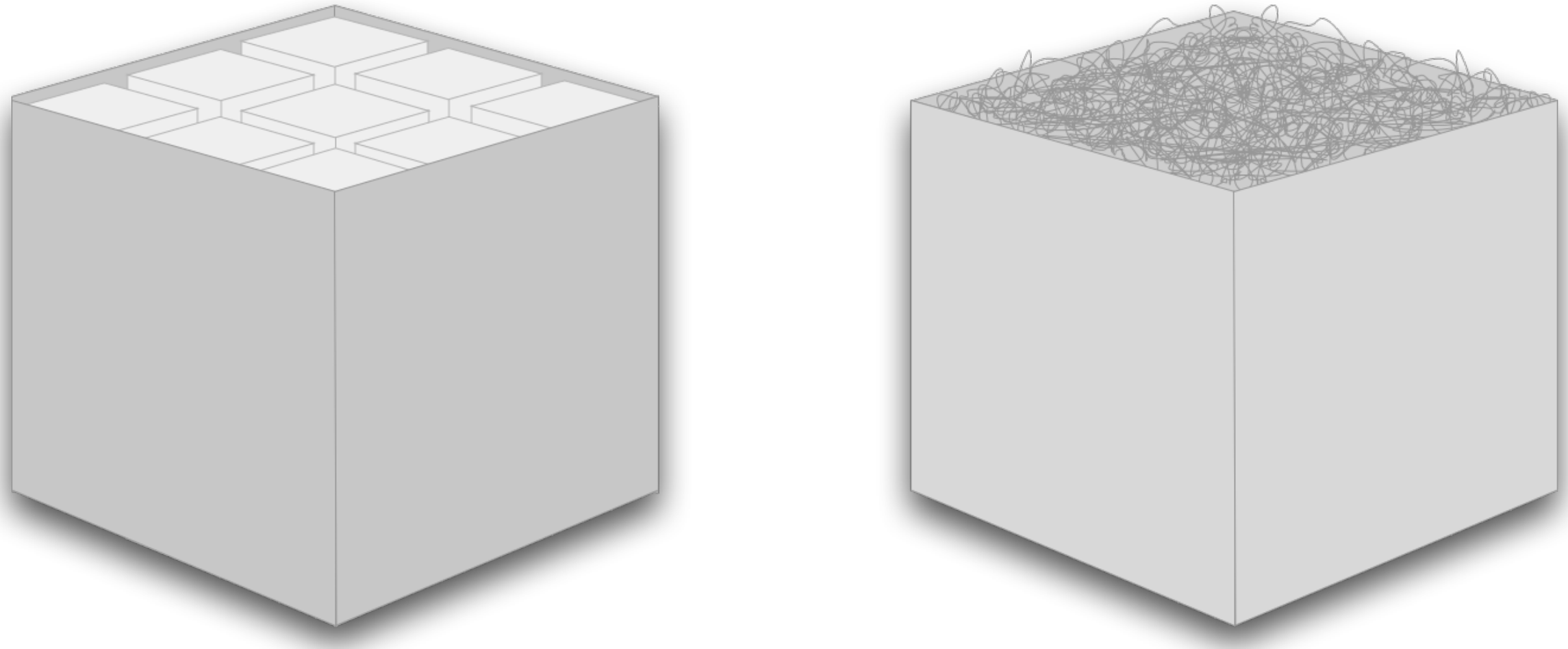
O Event Handler original é substituído por outro que publica o evento em um Broker de mensagem. Ex: RabbitMQ, Kafka, ActveMQ

considerações Finais

modelagem de domínio não é simples

O controle de acoplamento é extremamente importante

mas lembre-se:



https://martinfowler.com/articles/dont-start-monolith.html

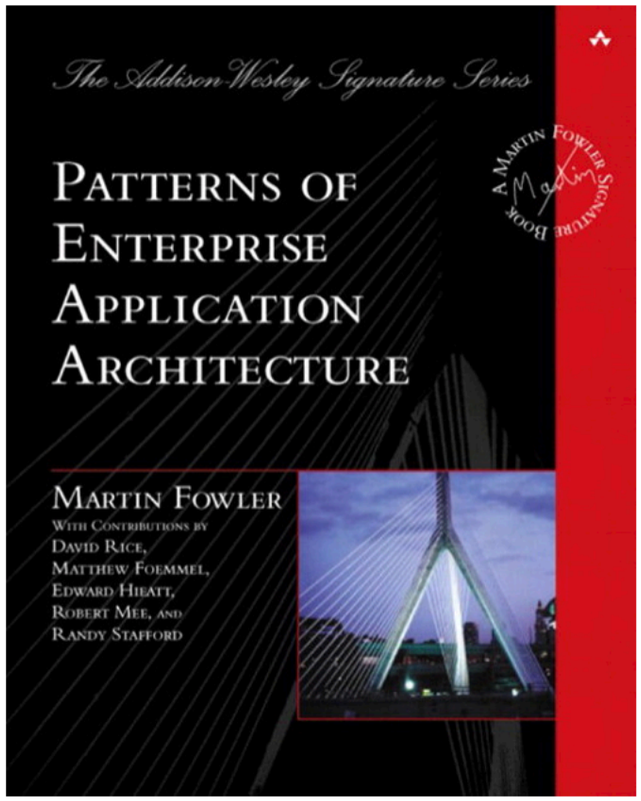
obrigado!

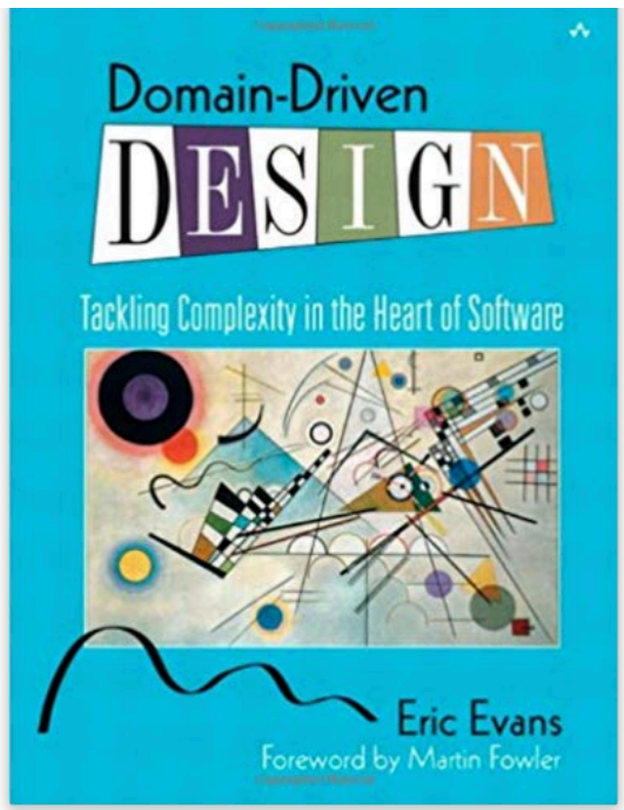
Luiz Costa

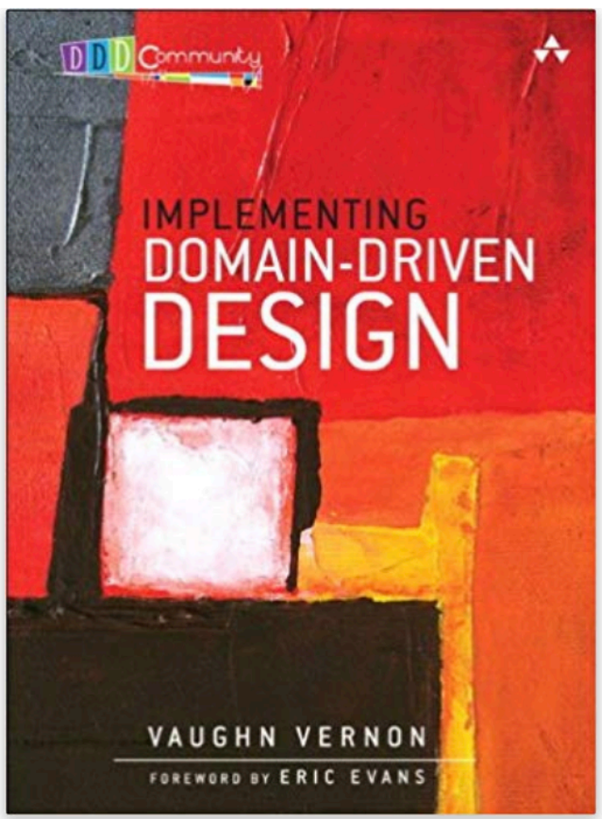
gutomcosta@gmail.com
@gutomcosta

THE
DEVELOPER'S
CONFERENCE

Referências







https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

https://blog.cleancoder.com/uncle-bob/2011/09/30/Screaming-Architecture.html

http://blog.firsthand.ca/2011/10/rails-is-not-your-application.html

https://cleancoders.com/video-details/clean-code-episode-7

https://www.youtube.com/watch?v=WpkDN78P884