Arquitetura Hexagonal + Eventos (parte 1)

7COMm
Serviços e Soluções em TI

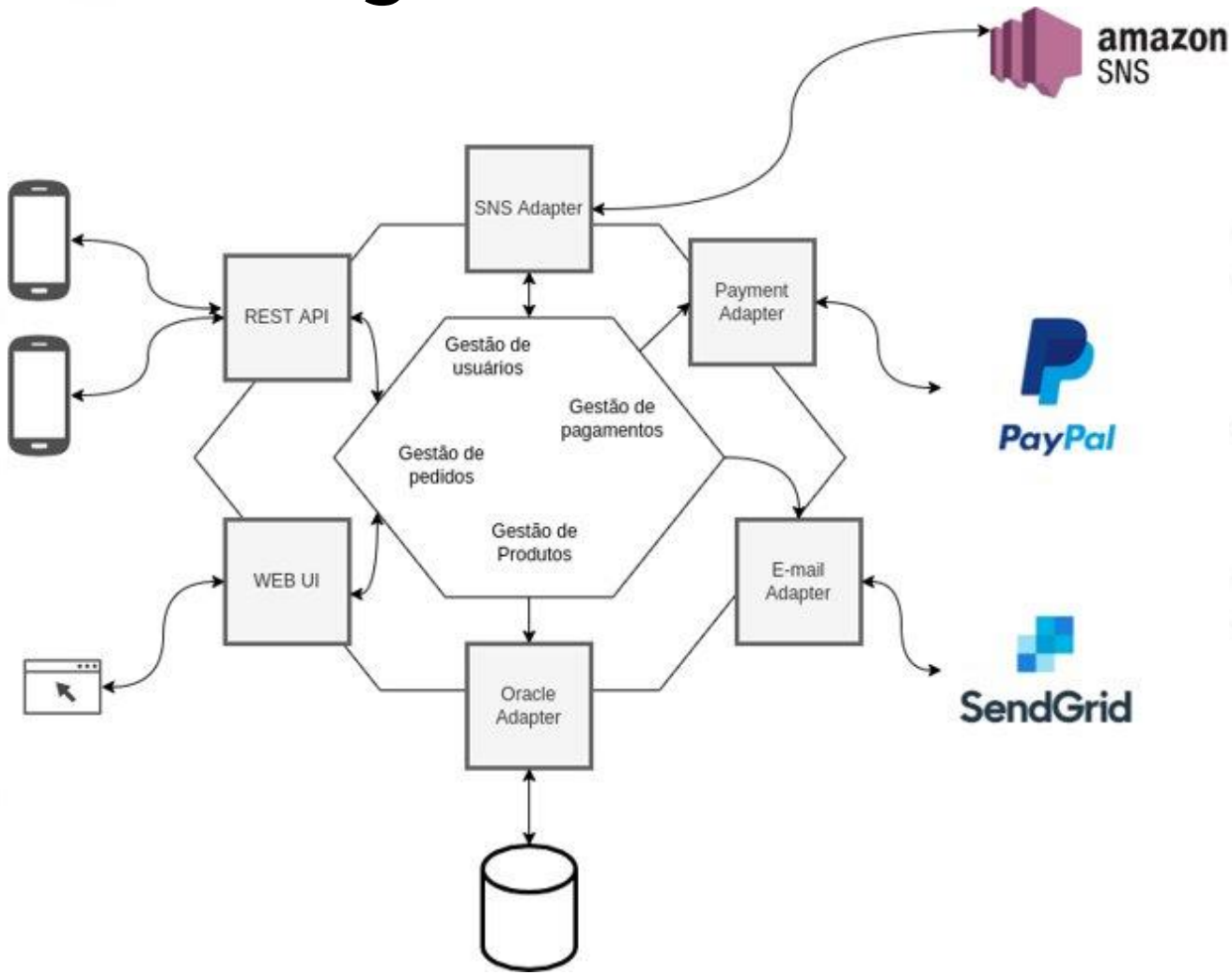
Arquitetura Hexagonal

7COMm
Serviços e Soluções em TI

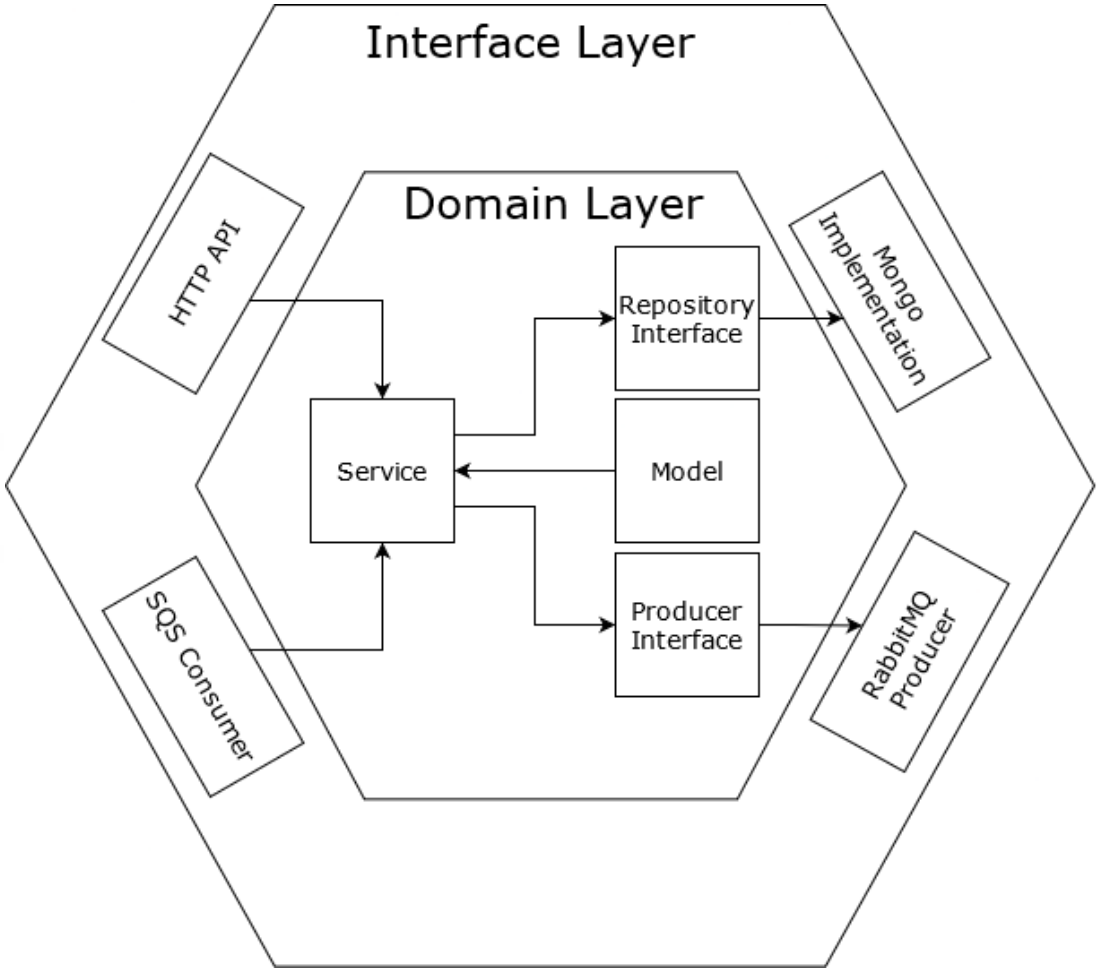
Arquitetura Hexagonal

- Divide as classes de sistema em dois tipos: *domínio* e *infraestrutura*
- Classes de **domínio** fornecem as **regras de negocio**
- Classes de **infraestrutura** fornecem as **interfaces com entidades externas**
- As classes de domínio não conhecem as tecnologias e com isso mudanças de tecnologia podem ser feitas sem impactá-las
- Mais robusta que o modelo em camadas
- A comunicação entre os dois tipos de classes é feita através de *adapters e ports*

Arquitetura Hexagonal



Arquitetura Hexagonal



Arquitetura Hexagonal - Vantagens

- Isola o core da aplicação
- Minimiza o impacto da tecnologia de frameworks
- Permite adiar decisões técnicas
- Facilita a implementação do DDD e de Agregados (visto na aula anterior)
- Se beneficia do uso de um bom encapsulamento

Arquitetura Hexagonal - Desvantagens

- Complexidade Adicional
- Custo de manutenção e esforço inicial
- Não existência de um padrão de organzação de código

Eventos

7COMm
Serviços e Soluções em TI

Eventos

- Acontecimento relevante.
- Ocasião ou atividade social.
- Adversidade.

Eventos e DDD

- Geralmente representa uma mudança de estado.
- É algo que aconteceu a um agregado.
- É representado por uma classe no modelo de domínio.
- Publicados através do ecossistema.

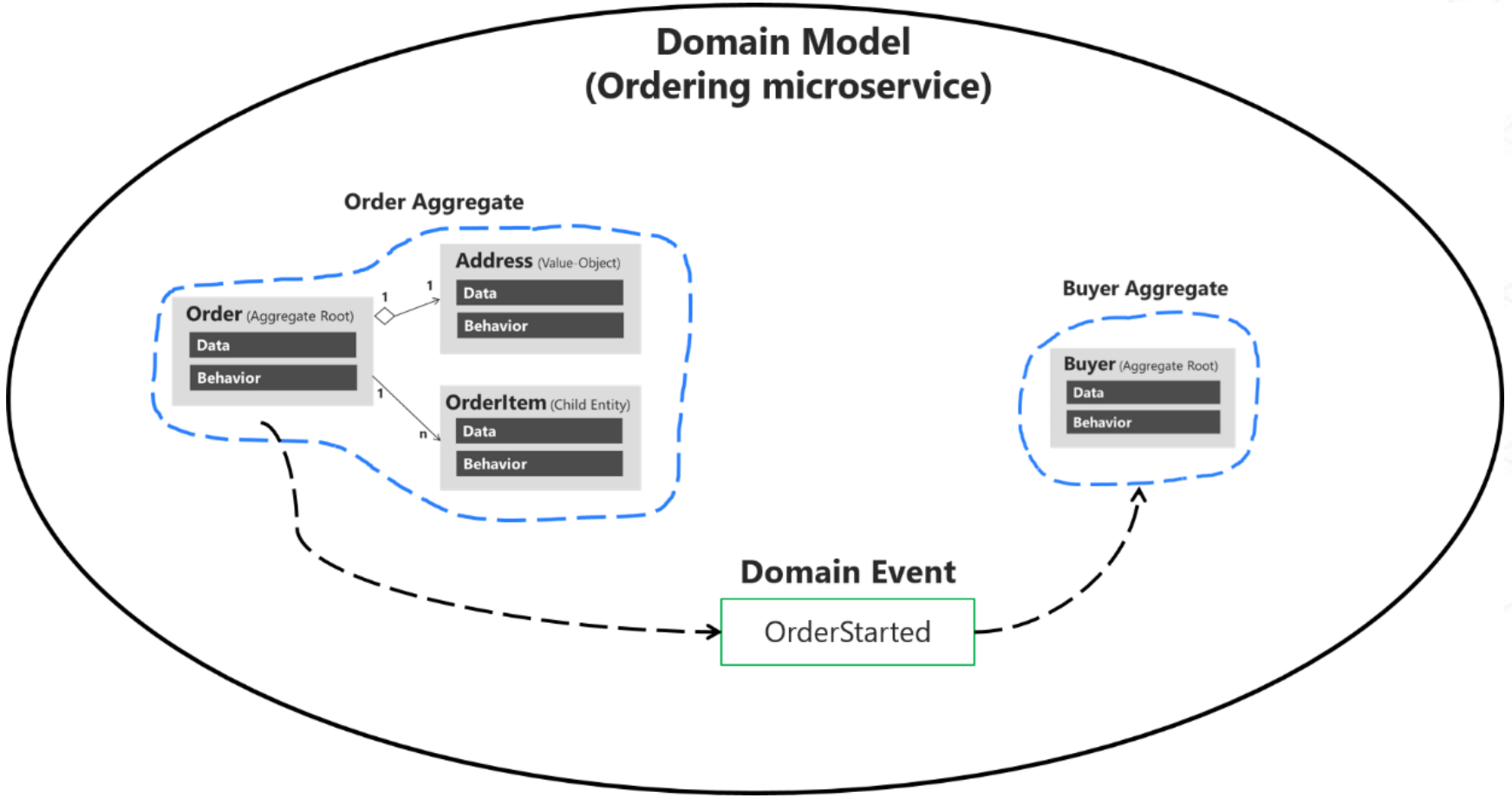
Eventos – Por quê são úteis?

- Alguns componentes têm interesse na mudança de estado de outros componentes
  - Manter consistência de dados em Sagas
  - Notificar serviços que mantém os dados alterados (CQRS)
  - Notificar webhooks/websockets
  - Envio de push notifications/emails/sms/...
  - Cálculo de métricas
  - Comportamento de usuários
  - ...

Eventos de Domínio

- Registro da ocorrencia de algo significativo naquele subdomínio
  - Pedido Criado
  - Pedido com mudança de status
  - Adição de novo produto
- Um contexto delimitado informa a outro contexto que algo aconteceu

Eventos de Domínio



Eventos de Domínio

```
class OrderCreated implements OrderEvent {
  private List<OrderLineItem> lineItems;
  private DeliveryInformation deliveryInformation;
  private PaymentInformation paymentInformation;
  ...
}
```

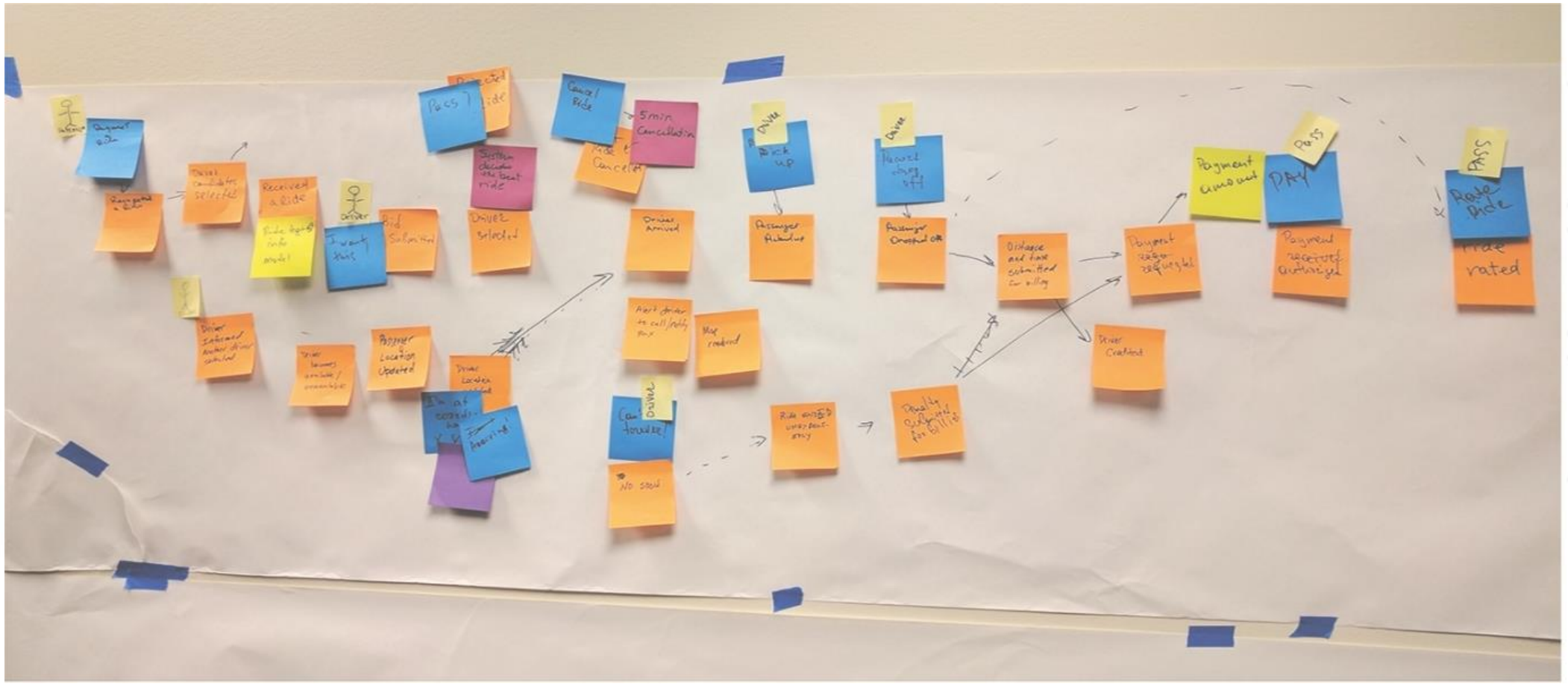
Identificando Eventos de Domínio

- Os requisitos são bons indicativos dos eventos
  - Quando X então Y
  - *Quando o pagamento for aprovado, então alterar o status do pedido*
- Event Storming.

Event Storm

1. Brainstorm
   - Especialistas no domínio especificam os eventos (laranja)
2. Identificar os gatilhos
   - Ações de usuários (azul), sistemas externos (lilás), passagem do tempo...
3. Identificar os agregados
   - Especialistas do domínio identificam agregados que recebem comandos e emitem eventos

Event Storm


Request ride
Driver candidate selected
Received a ride
Ride logged into system
driver
I want this
Bid Submitted
Driver selected
Driver Informed Another driver canceled
Driver leaves vicinity / unavailable
Passenger Location Updated
Driver Location updated
Arrived
Pass?
System decides the best ride
Cancel ride
Cancel?
5min Cancellation
Driver Arrived
Alert driver to call/reply msg
Max reached
Driver
Can't locate?
No show
Passenger Picked up
Driver
Pick up
Passenger Dropped off
Driver
Travel step started
Distance and time submitted for billing
Ride ended successfully
Payment requested
Driver Credited
Payment amount
PAY
Pass
Payment received successful
Rate Ride
Pass
rate
rated

Eventos Baseados em Requisitos

Serviço	Operação	Colaboradores
User Service	• CreateUser(); • UpdateUser(); • Login(); • Logout(); • ValidateUser();	-
OrderService	CreateOrder()	• ProductService.getProduct(); • ProductService.processProduct(); • UserService.validateUser(); • PaymentService.ReceivePayment()
OrderService	ChangeStatus()	ProductService.processProduct()

Eventos Baseados em Requisitos

Serviço	Operação	Colaboradores
PaymentService	ReceivePayment()	-
PaymentService	AuthorizePayment()	OrderService.ChangeStatus()
PaymentService	DenyPayment()	OrderService.ChangeStatus()
PaymentService	CancelPayment	OrderService.ChangeStatus()
ProductService	ListProducts()	
ProductService	GetProduct()	
ProductService	ProcessProducts()	
ProductService	CreateProduct()	

Event Sourcing

7COMm
Serviços e Soluções em TI

Event Sourcing

- Maneira de estruturar regras de negócio e persistência.
- Cada evento representa uma mudança de estado.
- O estado atual pode ser recriado.
- http://microservices.io/patterns/data/event-sourcing.html

Event Sourcing

- Vantagens
  - Preserva o histórico
  - Auditoria
- Desvantagens
  - Curva de aprendizado
  - Consulta a base de eventos é dificultosa

Problemas de persistência

- Uma classe é mapeada para uma tabela.
- Utiliza-se, normalmente, um ORM.

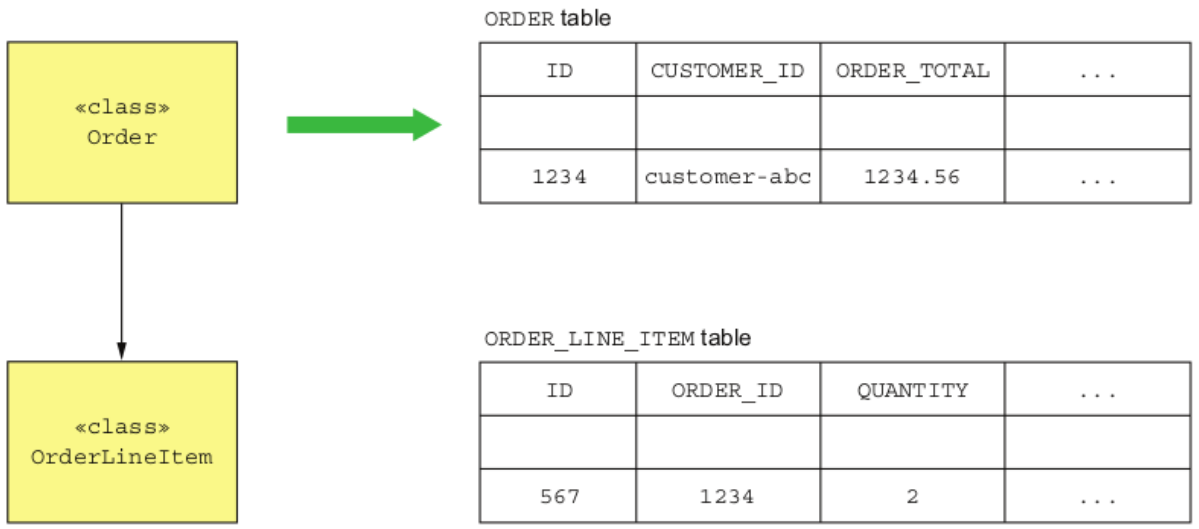


Figure 6.1 The traditional approach to persistence maps classes to tables and objects to rows in those tables.

Problemas de persistência

- Incompatibilidade conceitual entre tabelas e classes
  - “Object-Relational mapping is the Vietnam of Computer Science”, http://blogs.tedneward.com/post/the-vietnam-of-computer-science/
- Perda de histórico de alterações
  - Apenas o estado atual é persistido
  - Histórico deve ser implementado pelo time de devs
- Falta de suporte a publicação de eventos
  - Geração de eventos deve ser implementada pelo time de devs
  - Normalmente é incorporado na lógica de negócios

Event Sourcing - Visão geral

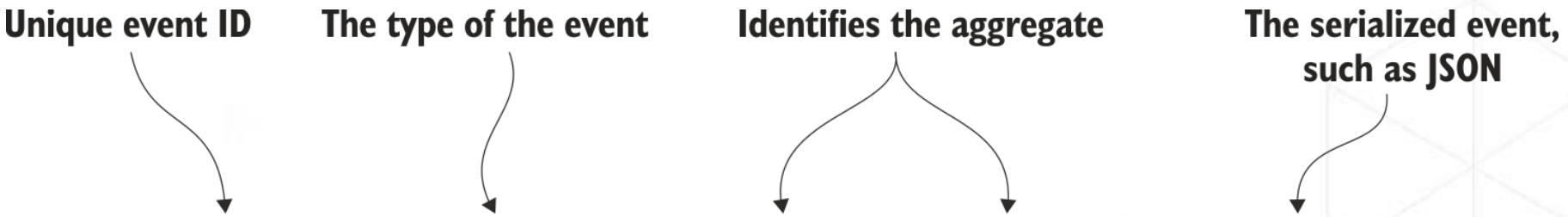
- Técnica centralizada em eventos.
- Agregados são armazenados como uma série de eventos.
- Cada evento representa mudança de estado do agregado.
- Regra de negócios estruturada no sentido de consumir estes eventos.

Persistindo com eventos

- Diferente da abordagem tradicional.
- Cada agregado é uma sequência de eventos (event store).
- Sempre que um agregado é atualizado, é salvo na tabela de eventos.

Persistindo com eventos

Figure 6.2. Event sourcing persists each aggregate as a sequence of events. A RDBMS-based application can, for example, store the events in an `EVENTS` table.



event_id	event_type	entity_type	entity_id	event_data
102	Order Created	Order	101	{...}
103	Order Approved	Order	101	{...}
104	Order Shipped	Order	101	{...}
105	Order Delivered	Order	101	{...}
...	...	...	...	...

EVENTS table

Persistindo com eventos

- Quando uma aplicação cria ou atualiza um agregado é inserido um evento na tabela de eventos.
- Portanto, para se reconstruir um agregado
    1. Carrega os eventos do agregado
    2. Cria uma instancia utilizando o construtor default
    3. Interage por todos os eventos, aplicando a mudança de estados

Eventos como mudança de estados

- Podem ser adaptáveis no seu conteúdo.
- Pode ser direcionado para a necessidade dos consumidores.
- Em Event Sourcing, eventos ***NÃO SÃO OPCIONAIS***.
- Se o estado de um agregado muda, ou mesmo é criado, este **DEVE** emitir um evento.
- Mudanças de estado podem ser mínimas, como trocar o valor de um campo; Ou grandes, quando adicionar e remover itens de um pedido, por exemplo.

Eventos como mudança de estados

- Podem conter dados mínimos
  - Id do evento
  - Id do aggregate
- Podem conter dados completos
  - Objeto agregado
- Definições de dados e publicação do evento são responsabilidades do próprio agregado.

Eventos como mudança de estados

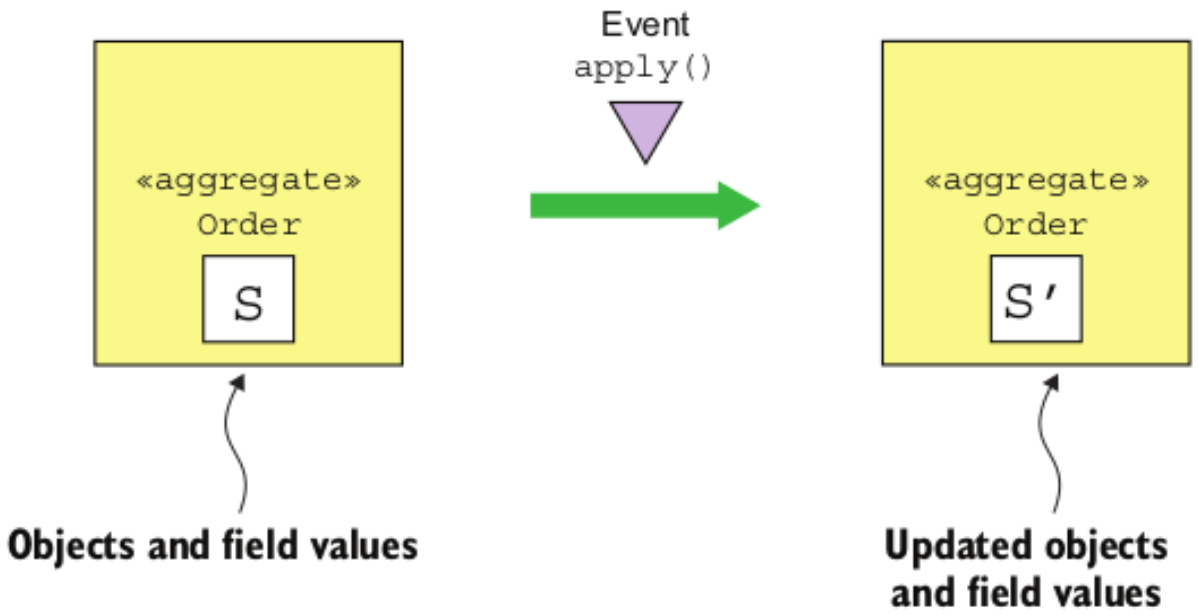


Figure 6.3 Applying event `E` when the `Order` is in state `S` must change the `Order` state to `S'`. The event must contain the data necessary to perform the state change.

Event Sourcing e Agregados

- Cada método de um agregado deve ser *focado* em eventos
    1. Recebem comando como parâmetro
    2. Determinam qual mudança deve ser realizada
    3. Retorna uma lista de eventos que devem ser executados
- Métodos focados em eventos **NÃO PODEM FALHAR**
    - Representam uma mudança de estado que já aconteceu

Event Sourcing e Agregados

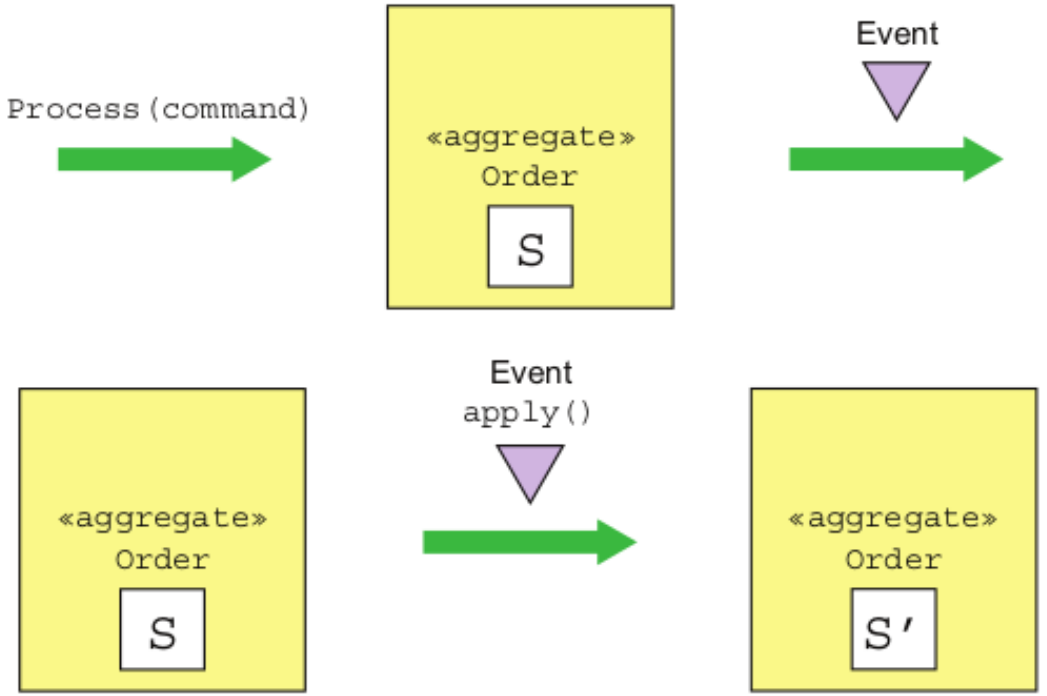


Figure 6.4 Processing a command generates events without changing the state of the aggregate. An aggregate is updated by applying an event.

Event Sourcing e publicação de eventos

- Já verificamos em aulas anteriores maneiras de publicação de eventos
  - Pooling
  - Transaction logs

Pooling

- Pode-se salvar os eventos em uma tabela.
- O elemento interessado, deve, então, realizar buscas regularmente nesta tabela.
- SELECT * FROM EVENTS where event_id > ? ORDER BY event_id ASC.
- Problema dessa abordagem é que as transações podem ser comitadas em uma ordem diferente da ordem em que os eventos são gerados.

Transaction Log Tailing

- Eventuate Local.
- Lê os eventos da tabela e os publica em um message broker.
- Garante que eventos vão ser publicados e também tem mais performance e é mais escalável.

Event Store

- Híbrido entre um banco de dados e um message broker
  - Possui uma API para inserir e recuperar eventos por chave
  - É permitido subscribe em uma API de eventos
- Pode-se utilizar um RDBMS e implementar a estratégia de pooling.
- Pode-se utilizar um framework de mercado.

Event Store

- Híbrido entre um banco de dados e um message broker
  - Possui uma API para inserir e recuperar eventos por chave
  - É permitido subscribe em uma API de eventos
- Pode-se utilizar um RDBMS e implementar a estratégia de pooling.
- Pode-se utilizar um framework de mercado.

Event Store

- Event Store (https://eventstore.org)
- Lagom (http://www.lightbend.com/lagom-framework)
- Axon (http://www.axonframework.org)

Próximos Passos

- Implementação de Domain Events
- Continuação de Event Sourcing

OBRIGADO!

Centro
Rua Formosa, 367 - 29º andar Centro, São Paulo - SP, 01049-000

Alphaville
Avenida Ipanema, 165 - Conj. 113/114 Alphaville, São Paulo - SP,06472-002

+55 (11) 3358-7700
contact@7comm.com.br

7COMm
Serviços e Soluções em TI

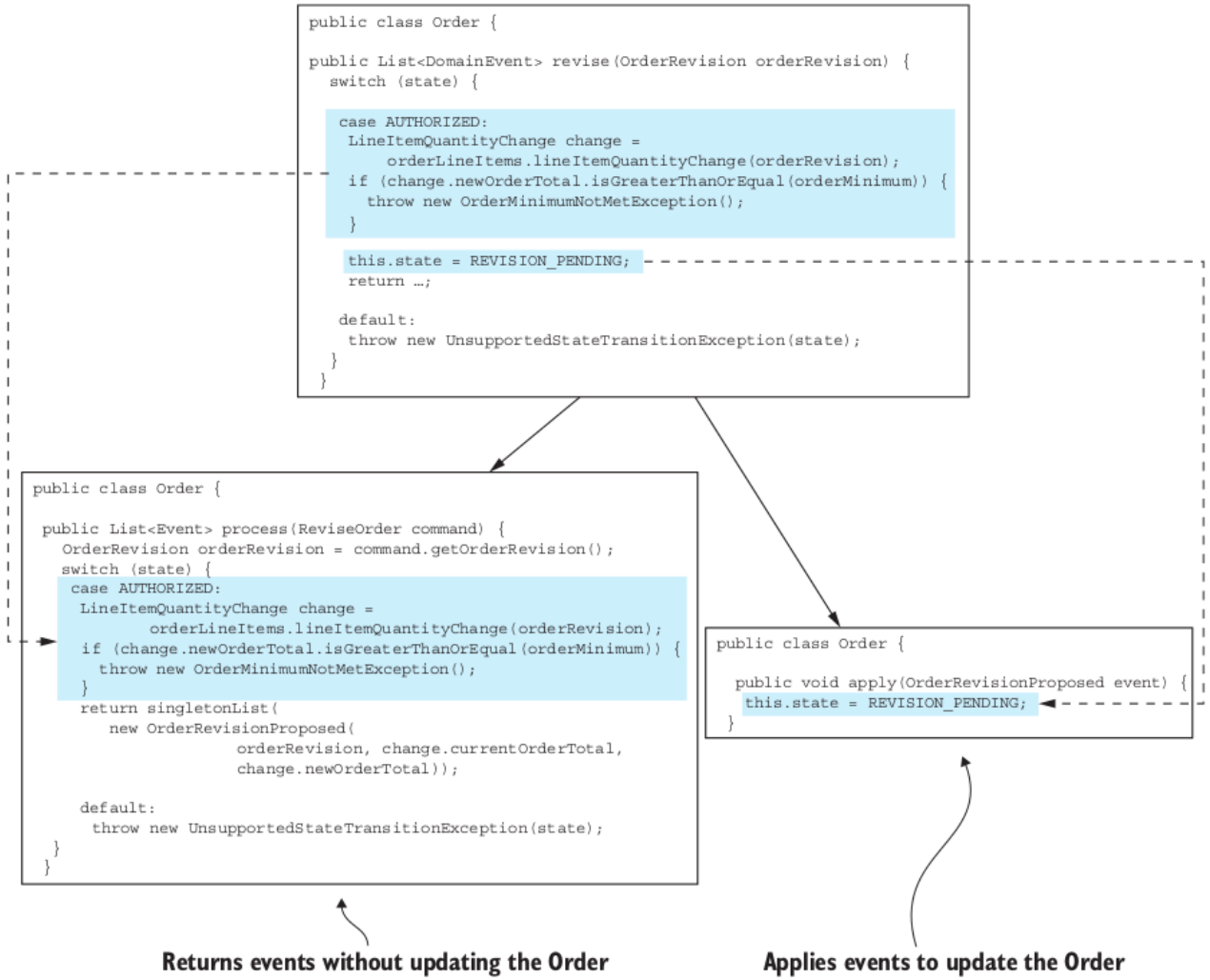


Figure 6.5 Event sourcing splits a method that updates an aggregate into a `process()` method, which takes a command and returns events, and one or more `apply()` methods, which take an event and update the aggregate.

Optimistic Locking

- Não é incomum dois ou mais comandos atualizarem simultaneamente o mesmo agregado.
- Nessas situações podemos utilizar o optimistic lock
  - lê-se o agregado, com um número de versão
  - Faz-se a alteração de estado
  - Salva-se o agregado com o número de versão incrementado

Optimistic Locking

- Uma coluna de *versão* do agregado.
- Sempre que duas transações tentarem atualizar o mesmo agregado, somente a primeira conseguirá.
- A segunda transação deve falhar devido o número da versão ser diferente do que foi lido inicialmente.
- Desse modo, uma segunda transação não substituirá as mudanças da primeira transação.

Event Sourcing e publicação de eventos

- Já verificamos em aulas anteriores maneiras de publicação de eventos
  - Pooling
  - Transaction logs

Pooling

- Pode-se salvar os eventos em uma tabela.
- O elemento interessado, deve, então, realizar buscas regularmente nesta tabela.
- SELECT * FROM EVENTS where event_id > ? ORDER BY event_id ASC.
- Problema dessa abordagem é que as transações podem ser comitadas em uma ordem diferente da ordem em que os eventos são gerados.

Transaction Log Tailing

- Eventuate Local.
- Lê os eventos da tabela e os publica em um message broker.
- Garante que eventos vão ser publicados e também tem mais performance e é mais escalável.

Event Store

- Híbrido entre um banco de dados e um message broker
  - Possui uma API para inserir e recuperar eventos por chave
  - É permitido subscribe em uma API de eventos
- Pode-se utilizar um RDBMS e implementar a estratégia de pooling.
- Pode-se utilizar um framework de mercado.

Event Store

- Event Store (https://eventstore.org)
- Lagom (http://www.lightbend.com/lagom-framework)
- Axon (http://www.axonframework.org)

Snapshots

- Agregados com poucas transições de estados são restaurados de maneira simples.
- Agregados com vida longa e muitas transições podem ser problemáticos para restaurar através de eventos.
- Com o tempo seria extremamente ineficiente.
- Uma solução é o uso de snapshots.

Snapshots

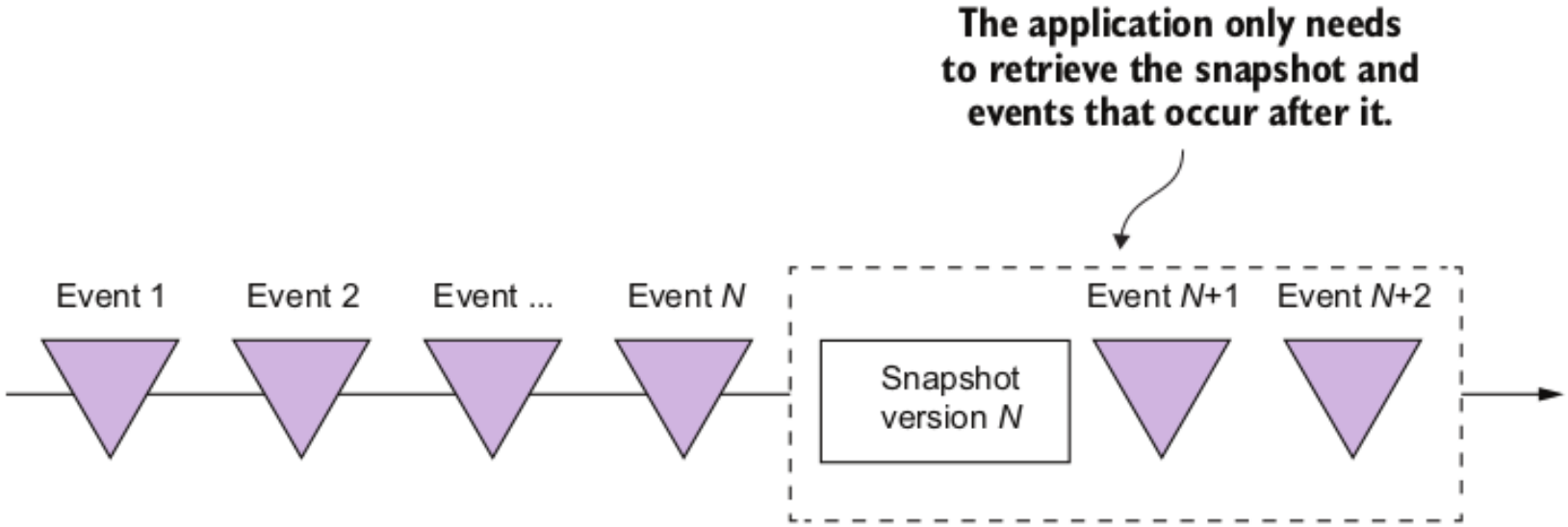


Figure 6.7 Using a snapshot improves performance by eliminating the need to load all events. An application only needs to load the snapshot and the events that occur after it.

Snapshots

- No exemplo acima, foi criado um snapshot N.
- É uma escrita de todos os eventos até então.
- Para se restaurar o estado atual, utilizaria este snapshot + a aplicação de dois eventos.

Snapshots

EVENTS

event_id	event_type	entity_type	entity_id	event_data
...	...	...	...	...
103	...	Customer	101	{...}
104	Credit Reserved	Customer	101	{...}
105	Address Changed	Customer	101	{...}
106	Credit Reserved	Customer	101	{...}

SNAPSHOTS

event_id	entity_type	event_id	snapshot_data
...	...	...	...
103	Customer	101	{name: "..." , ...}
...	...	...	...
...	...	...	...

Figure 6.8 The `Customer Service` recreates the `Customer` by deserializing the snapshot's JSON and then loading and applying events #104 through #106.

Evolução dos eventos

- Como lidar com a mudança nos eventos?
  - Precisamos lidar com um potencial número de versões de eventos
- Uma aplicação com event sourcing possui três níveis de schema de eventos
  - Consiste de um ou mais agregados
  - Define quais eventos os agregados emitem
  - Define as estruturas dos eventos

Evolução dos eventos

Table 6.1 The different ways that an application's events can evolve

Level	Change	Backward compatible
Schema	Define a new aggregate type	Yes
Remove aggregate	Remove an existing aggregate	No
Rename aggregate	Change the name of an aggregate type	No
Aggregate	Add a new event type	Yes
Remove event	Remove an event type	No
Rename event	Change the name of an event type	No
Event	Add a new field	Yes
Delete field	Delete a field	No
Rename field	Rename a field	No
Change type of field	Change the type of a field	No

Upcasting

- Metodologia análoga às migrations em bancos de dados.
- Event Sourcing não migra os schemas para uma nova versão, mas sim os próprios eventos.
- Um componente chamado upcaster atualiza eventos individualmente para uma nova versão.

Benefícios de Event Sourcing

- Publica eventos confiáveis.
- Preserva a história dos agregados.
- Provê uma *máquina do tempo*.
- Evita o problema de incompatibilidade de impedância objeto-relacional.

Pontos negativos de Event Sourcing

- Paradigma de linguagem diferente do convencional
  - Pode ser utilizado durante a conversão da aplicação em microsserviços
- Curva de aprendizado maior.
- Evolução dos eventos pode ser problemática de tratar.
- Buscas na tabela de eventos é complicado, uma vez que os agregados devem ser *reconstruídos* no select.

Exclusão de dados com Event sourcing

- Deve ser tratado de maneira especial.
- Um dos objetivos de Event Sourcing é gravar dados pra sempre.
- Pode-se utilizar *soft delete*, através de flags
  - Potencialmente mais complicado com dados sensíveis, como os tratados pela LGPD
  - Nesse caso, pode-se pensar em uma solução para *soft delete* utilizando criptografia