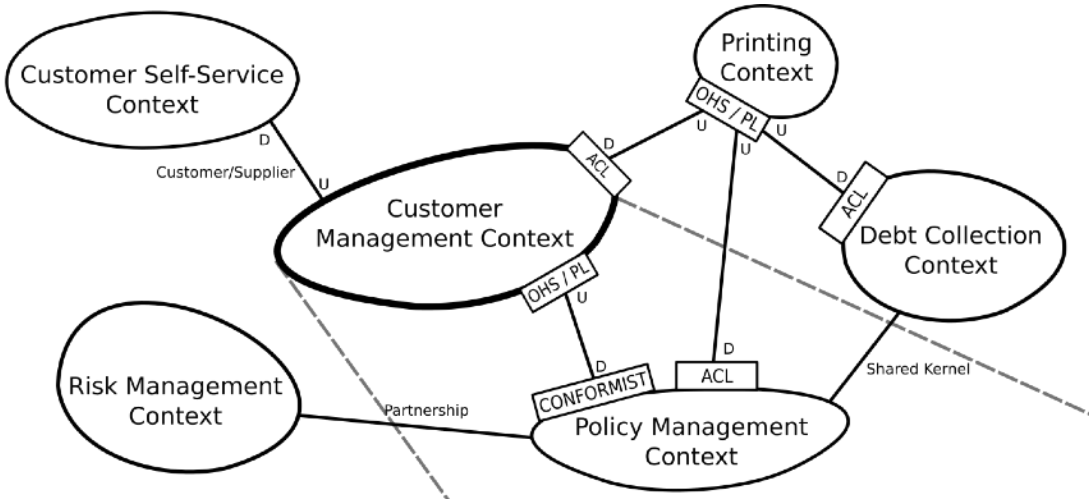
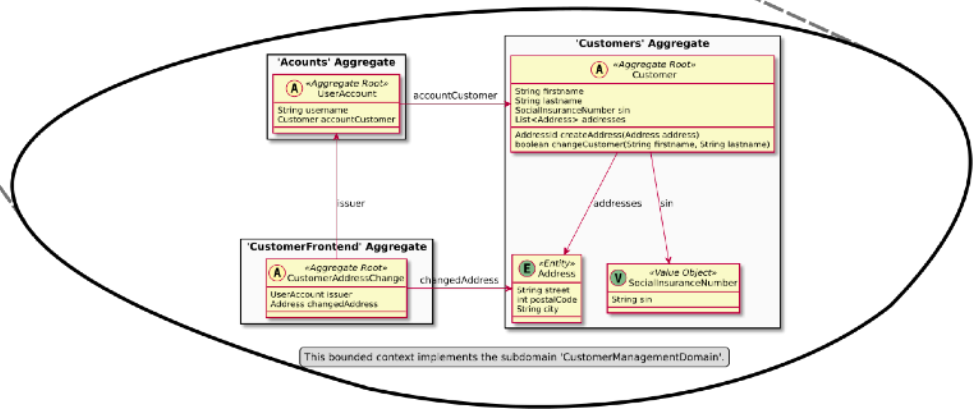
Domain-Driven Design - Parte 1

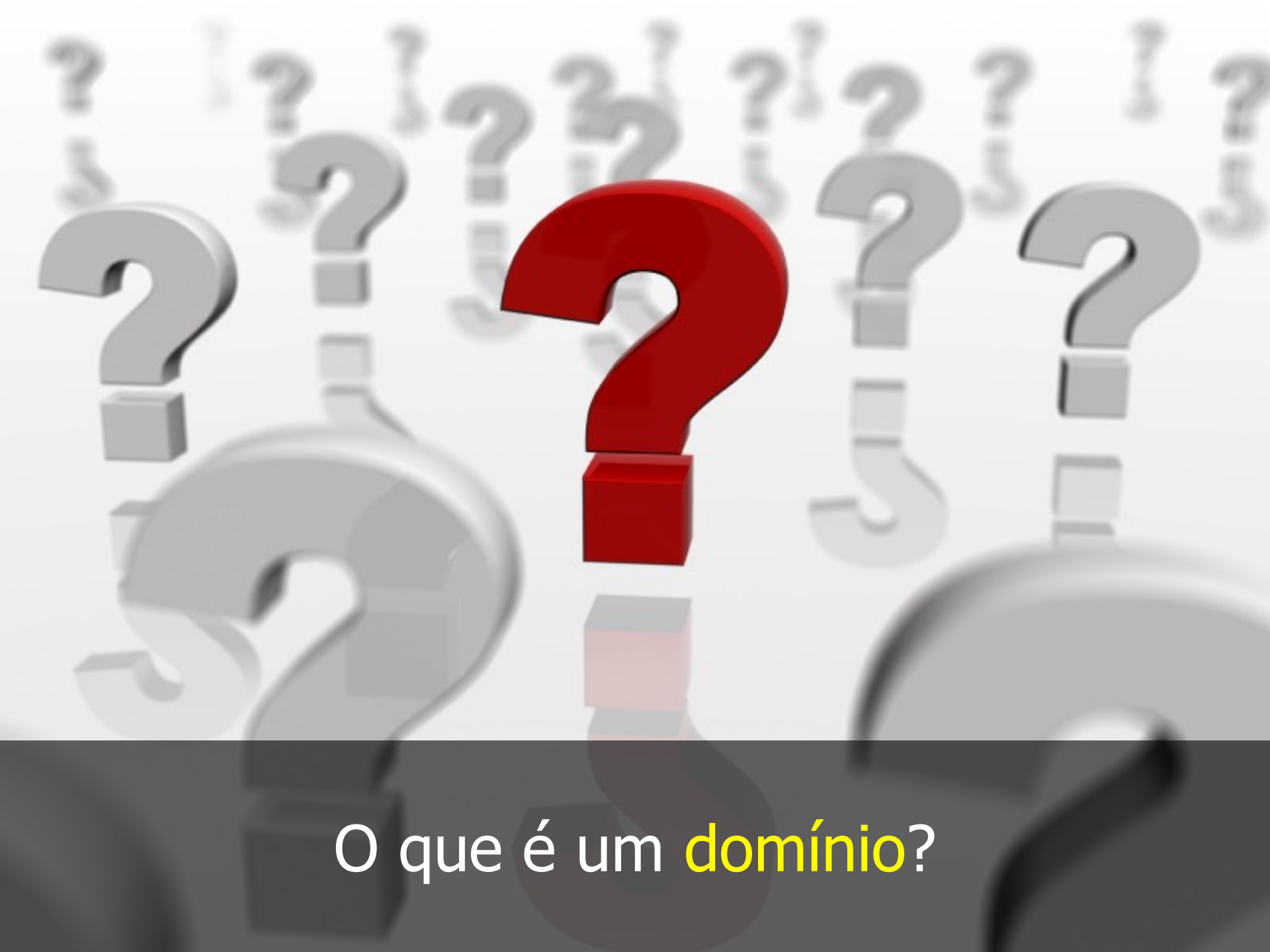
O Domain-Driven Design se divide em duas partes:



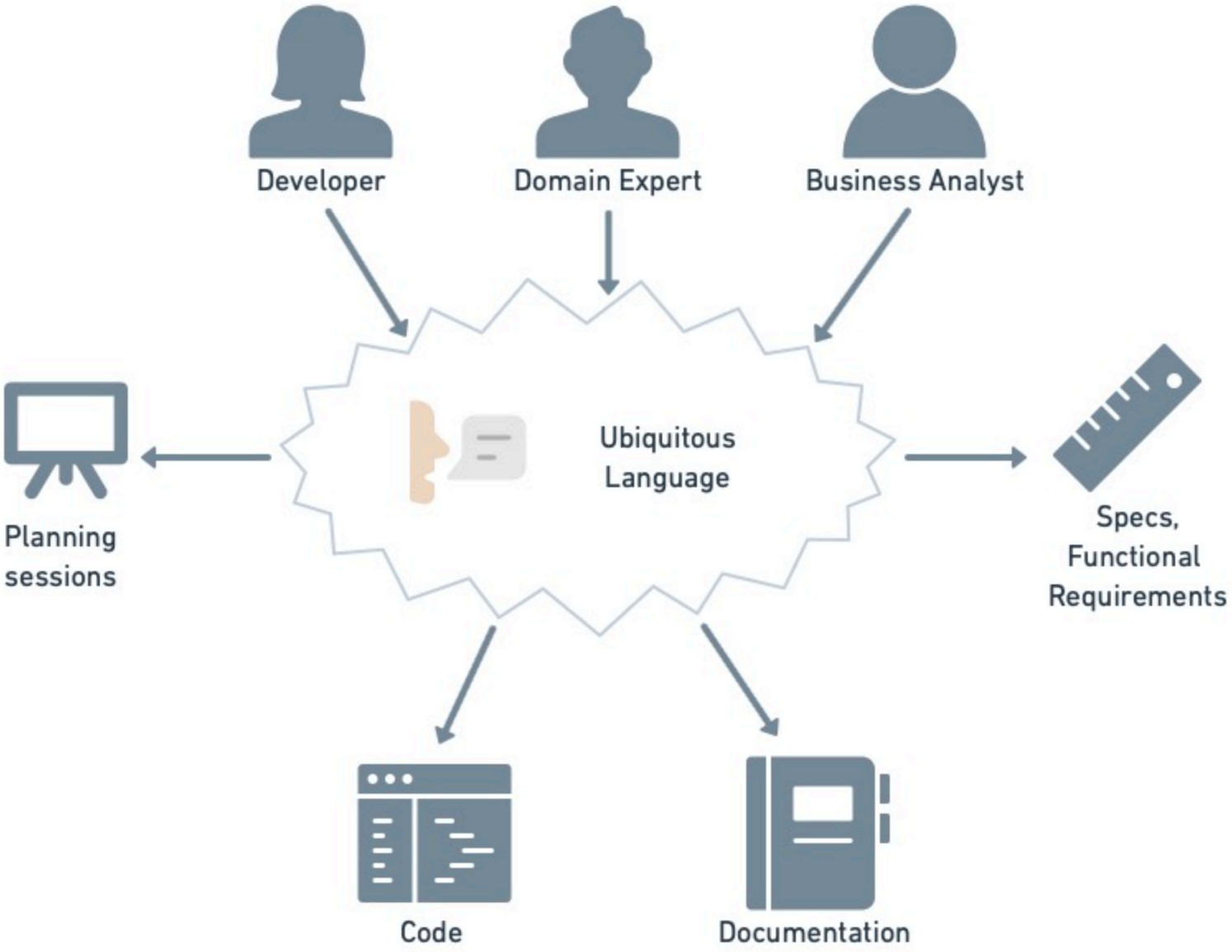
Strategic DDD:
Context Map

Tactical DDD:
Domain Model inside a Bounded Context

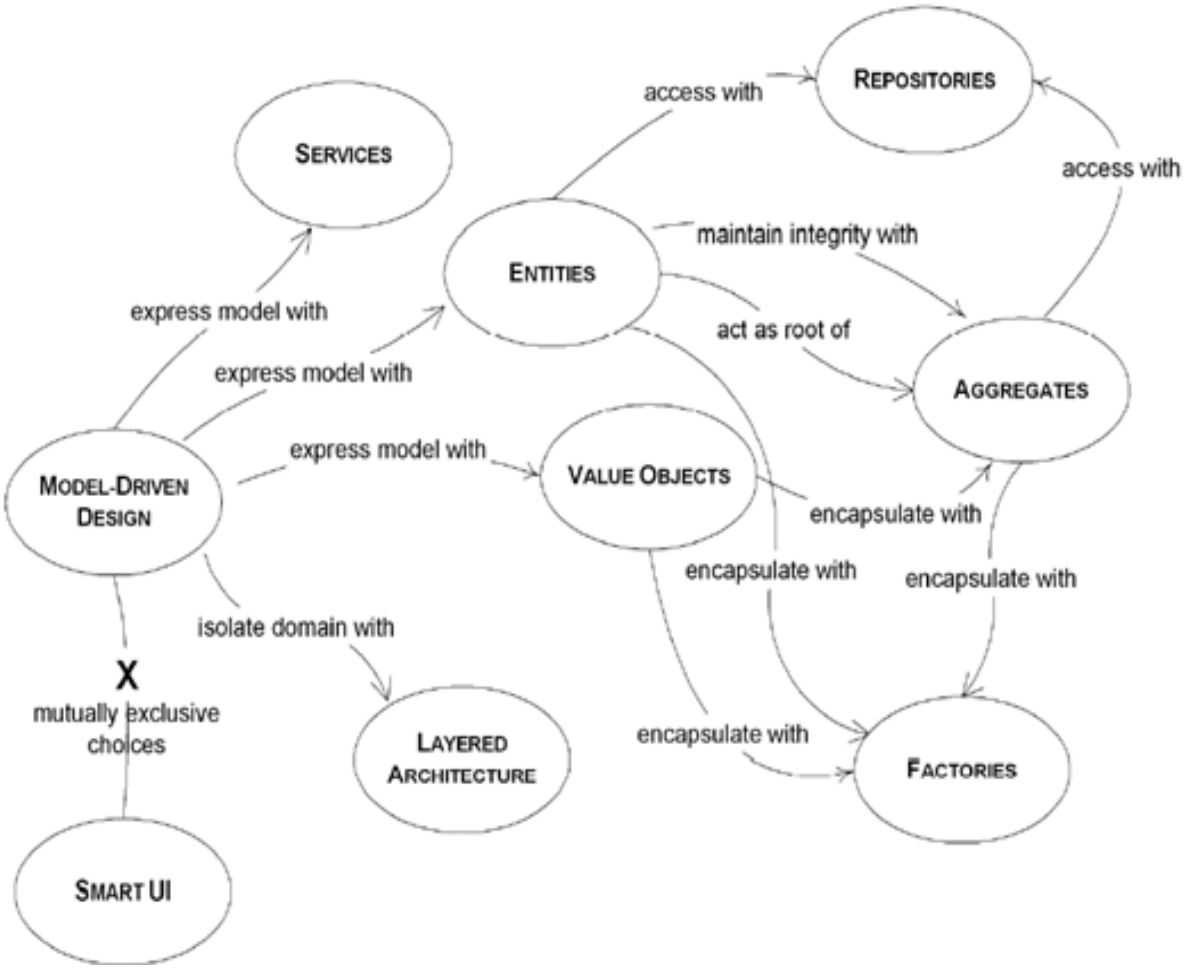



O que é um domínio?

O domínio é o problema, em termos de negócio, que o precisa ser resolvido independente da tecnologia que será utilizada

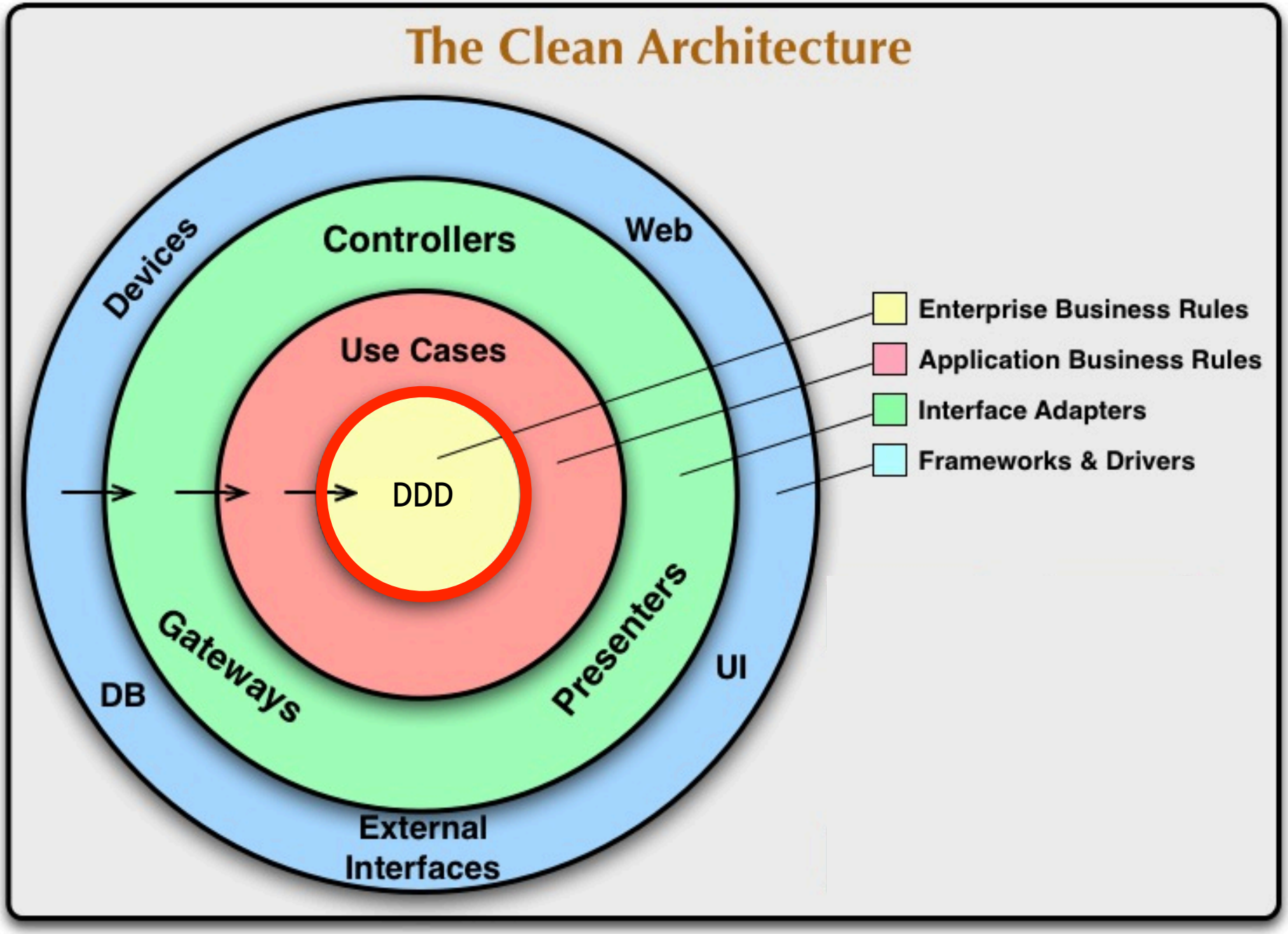

Developer
Domain Expert
Business Analyst
Ubiquitous Language
Planning sessions
Specs, Functional Requirements
Code
Documentation

"Domain experts should object to terms or structures that are awkward or inadequate to convey domain understanding, developers should watch for ambiguity or inconsistency by using the model-based language and not being satisfied until the approach is complete and comprehensible"



A **modelagem tática** é utilizada para construir a camada de domínio

Ela complementa a camada de Entities do Clean Architecture


The Clean Architecture
Devices
Controllers
Web
Use Cases
DDD
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

Objetos de Domínio

- Entities
- Value Objects
- Domain Services
- Aggregates
- Repositories*

Entities <E>

Abstraem regras de negócio independentes, tem identidade e estado, podendo sofrer mutação ao longo do tempo

Exemplos

Account: O passageiro ou motorista pode ter a sua conta bloqueada, a placa do carro modificada, a senha redefinida

Ride: Uma corrida pode ter o status em andamento ou finalizada, após ser finalizada o valor da tarifa é atualizado

Como gerar a identidade?

Manualmente: O próprio usuário pode gerar a identidade da entidade, por exemplo, utilizando o email ou um documento de identificação
Aplicação: A aplicação pode utilizar um algoritmo para gerar a identidade como um gerador de UUID
Banco de dados: O banco de dados por meio de uma sequência ou outro tipo de registro, centralizando a geração da identidade

Como comparar?

A comparação entre entities se dá pela identidade, sem levar em consideração as suas características

Value Objects <VO>

Também contém regras de negócio independentes, no entanto são identificados pelo seu valor, sendo imutáveis, ou seja, a mudança implica na sua substituição

Características

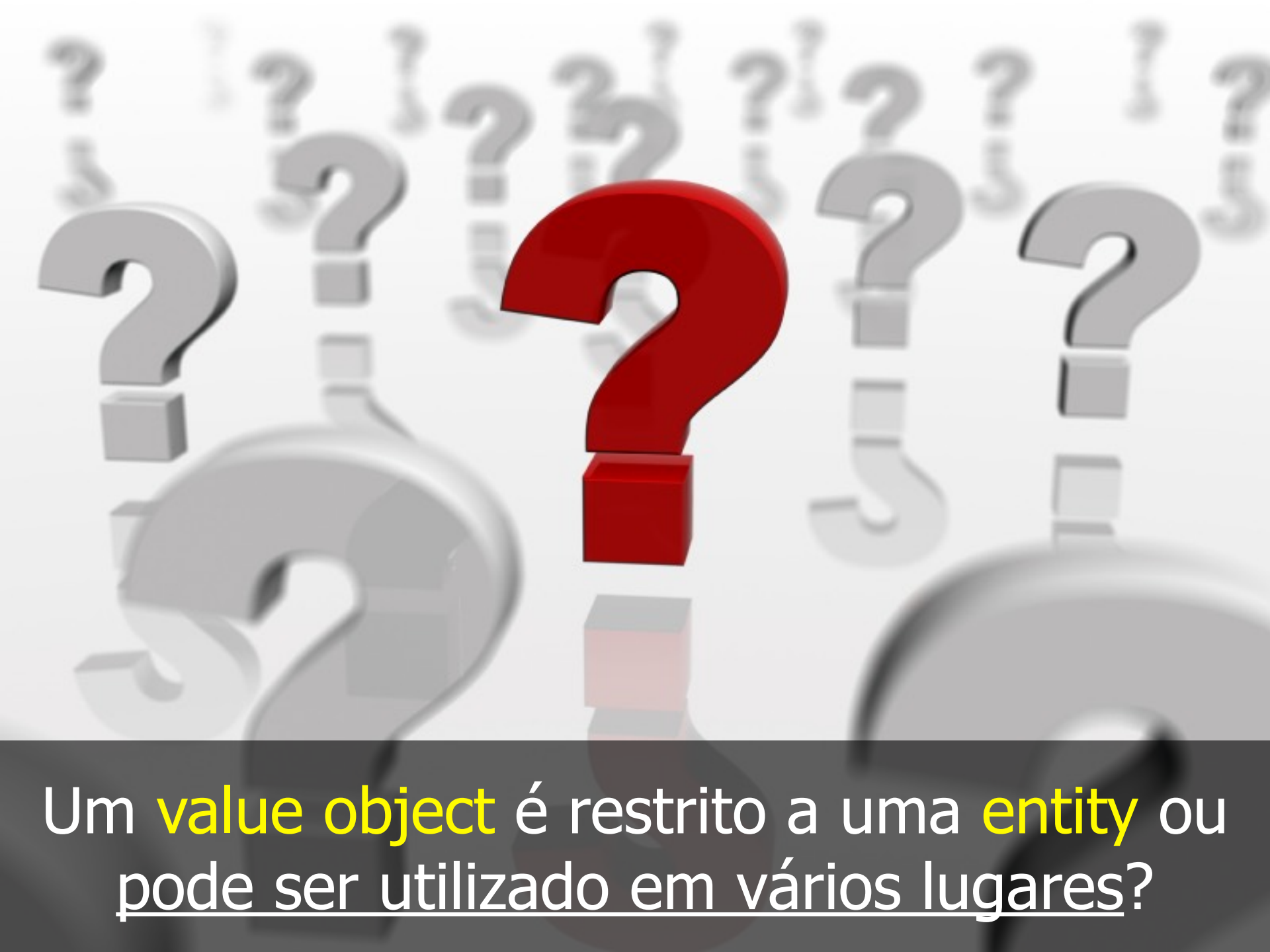
- Mede, quantifica ou descreve alguma coisa
- Seu valor é imutável
- É substituido quando seu valor mudar
- Pode ser comparado pelo valor que representa

Exemplos

Code: Representa uma determinada regra de formação de um número
Cpf: Garante que o número do documento é válido
Dimension: Abstrai a largura, altura, profundidade e peso de um item
Password: Representa uma senha
Color: Uma cor no formato RGB
Coord: A latitude e longitude
Email: Representa um email
Segment: Representa duas posições geográficas no tempo

Substituir por um ou mais primitivos

Uma técnica para identificar um value object é tentar substituí-lo por um tipo primitivo como uma string ou um número



Um value object é restrito a uma entity ou pode ser utilizado em vários lugares?

Domain Service <DS>

Realiza tarefas específicas do domínio, não tendo estado. É indicado quando a operação que você quer executar não pertence a uma entity ou a um value object

Exemplos

DistanceCalculator: Pegando duas coordenadas retorna a distância
FareCalculator: Calcula o valor de um segmento da corrida
TokenGenerator: Gera um token de acordo com um email

Utilize em operações que envolvem múltiplos objetos de domínio

Normalmente quando uma operação afeta múltiplos objetos de domínio, não pertencendo a nenhum deles, ela deve ser descrita em um domain service

Não crie serviços no lugar de entities e value objects, favorecendo um modelo anêmico


Como definir o relacionamento entre diferentes objetos de domínio?

A relação entre os objetos de domínio não é a mesma utilizada no banco de dados

Grandes aggregates podem trazer desperdício de memória, além de sobrecarregar o banco de dados sem necessidade já que nem sempre a camada de aplicação estará interessada em utilizá-lo na íntegra


O desafio é balancear a preservação da invariância com o consumo de recursos

Aggregates <A>

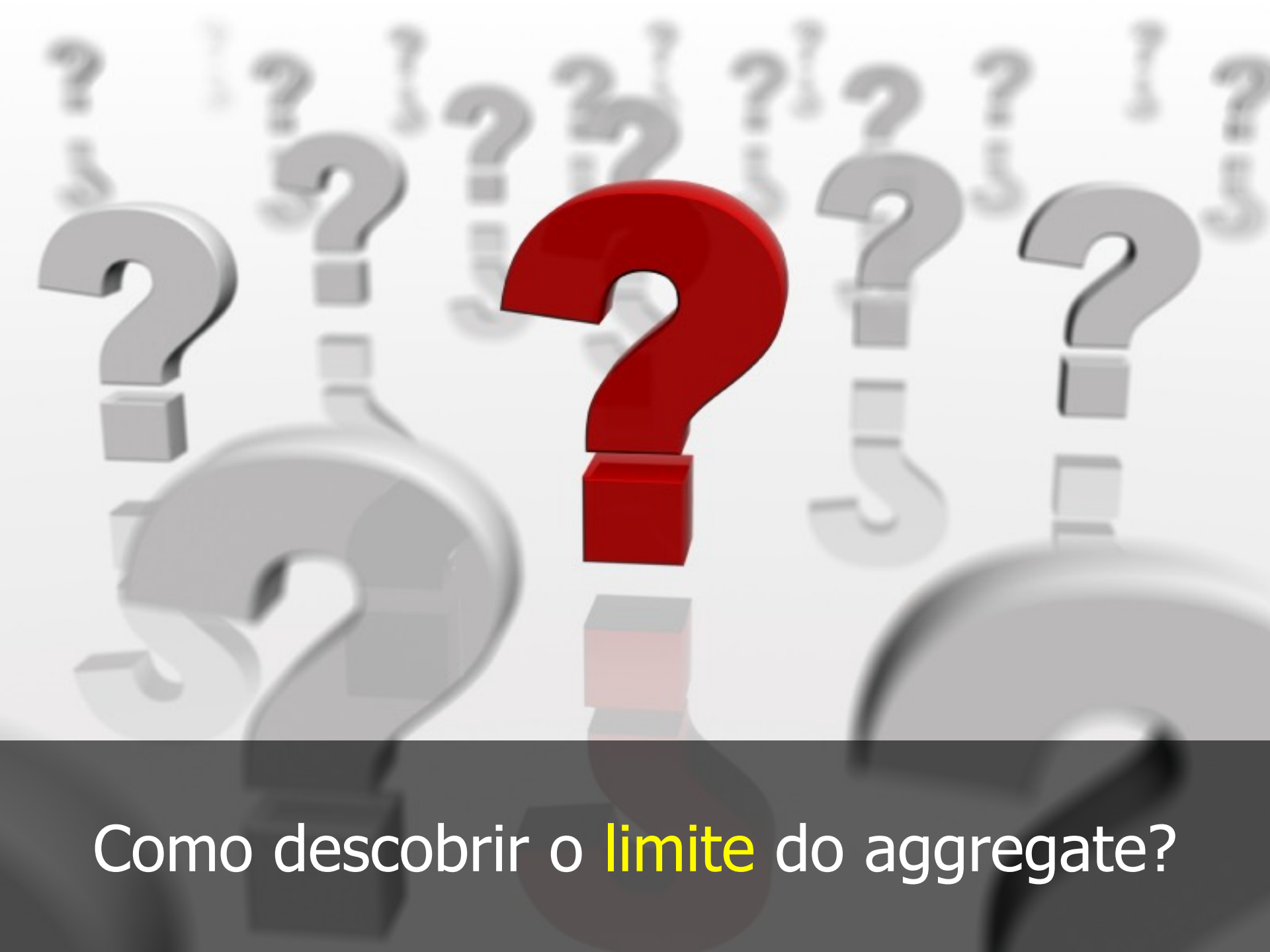
Um aggregate é um agrupamento, ou cluster, de objetos de domínio como entities e value objects, estabelecendo o relacionamentos entre eles

Todas as operações são realizadas por meio da raíz, que é uma entity ou aggregate root <AR>

Boas práticas na criação de aggregates

Crie aggregates pequenos: Comece sempre com apenas uma entidade e cresça de acordo com as necessidades

Referencie outros aggregates por identidade: Mantenha apenas a referência para outros aggregates, isso reduz a quantidade de memória e o esforço que o repositório faz para recuperá-los


Como descobrir o limite do aggregate?

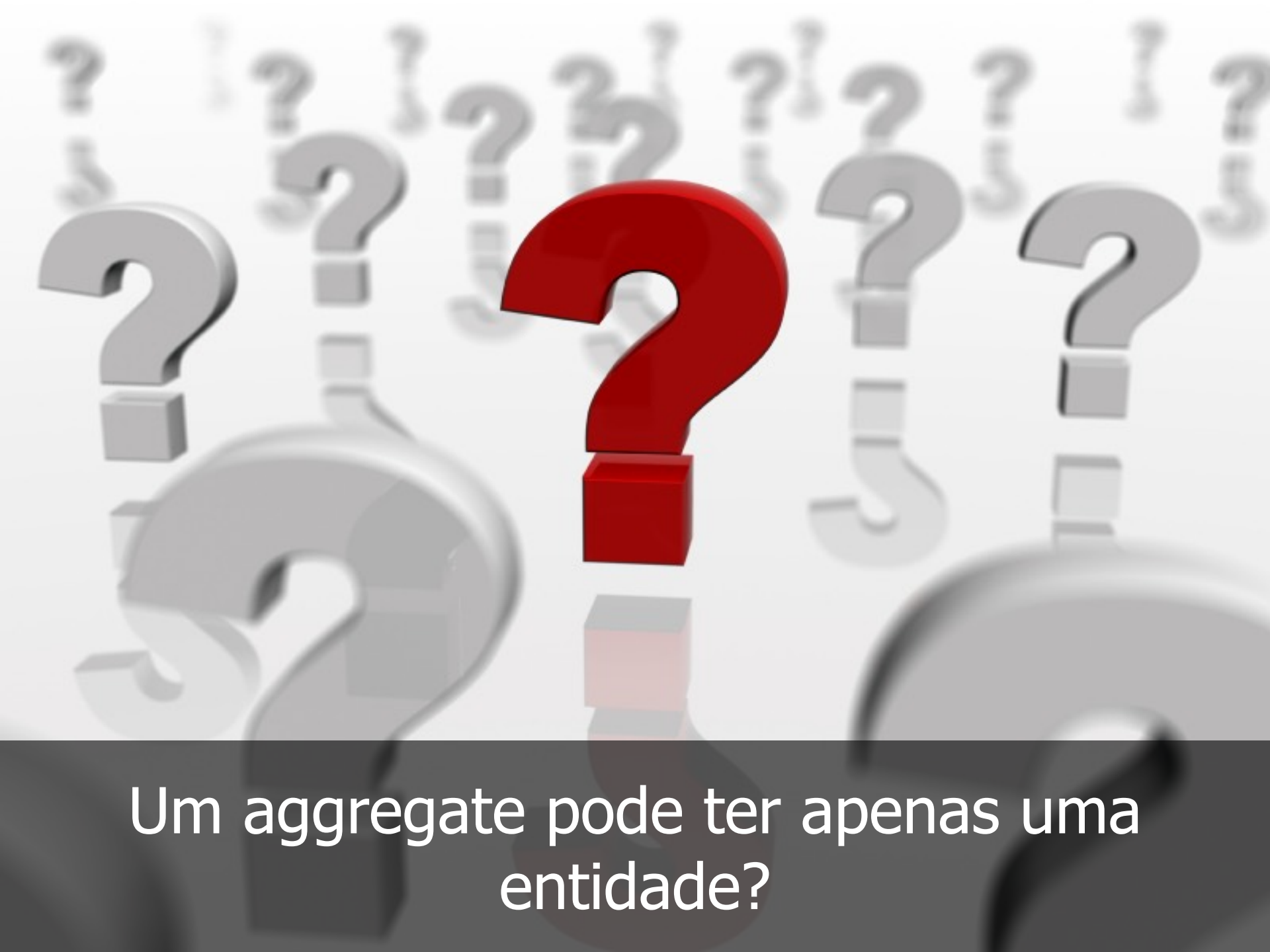
Se estiver difícil de implementar o repositório, talvez o aggregate seja muito grande e possa ser separado

Os aggregates deve refletir a base de dados?

Isso faria o aggregate ser muito grande e consumir muita memória, tornando o repository mais complexo do que deveria

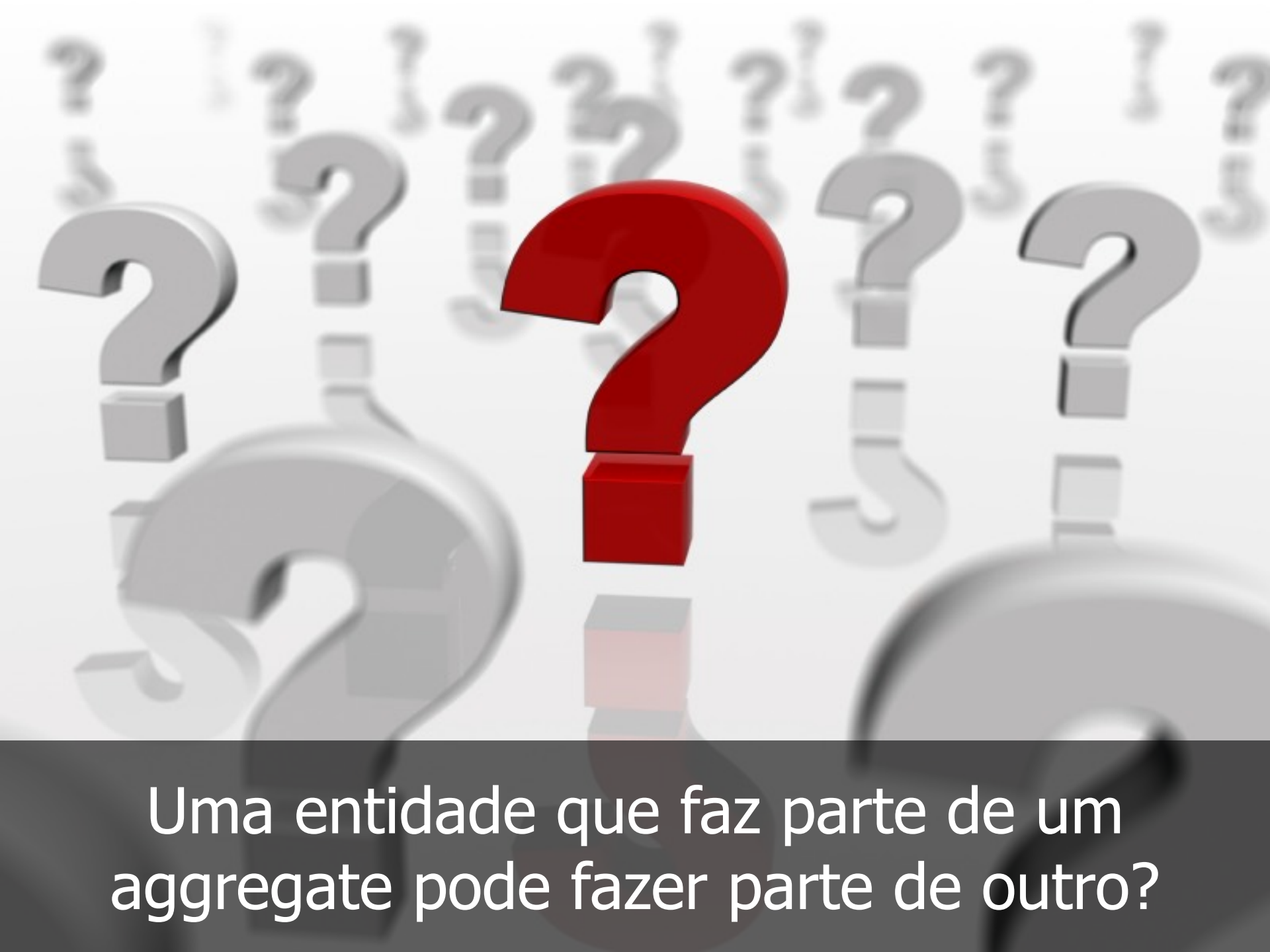
Um aggregate pode referenciar outros aggregates?

Pode, mas por identidade



Um aggregate pode ter apenas uma entidade?

Pode e quanto menor melhor



Uma entidade que faz parte de um aggregate pode fazer parte de outro?

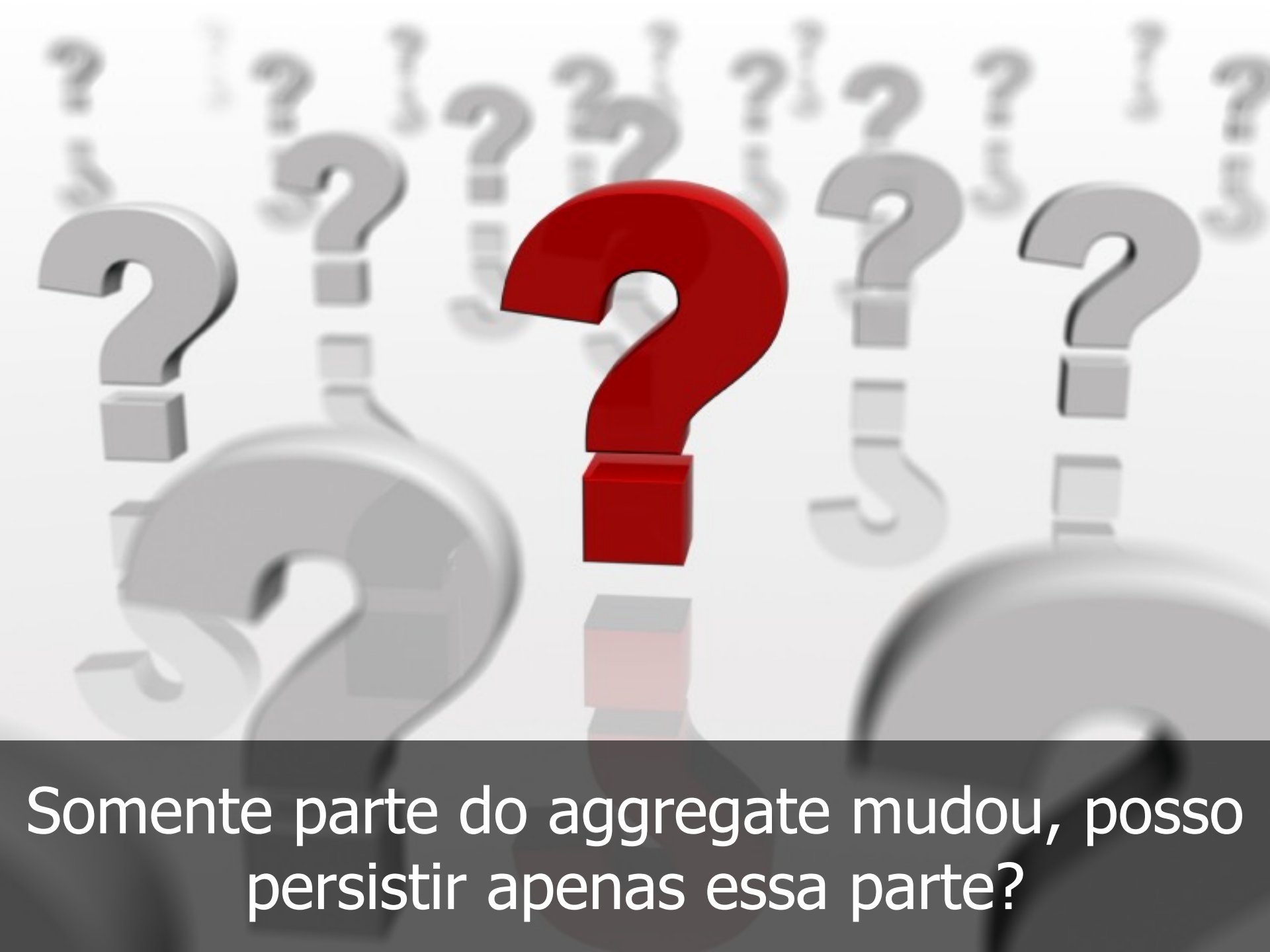
Não faz muito sentido, uma mudança na entidade utilizada por um aggregate poderia causar a quebra em outro

Repositories

É uma extensão do domínio responsável por realizar a persistência dos aggregates, separando o domínio da infraestrutura

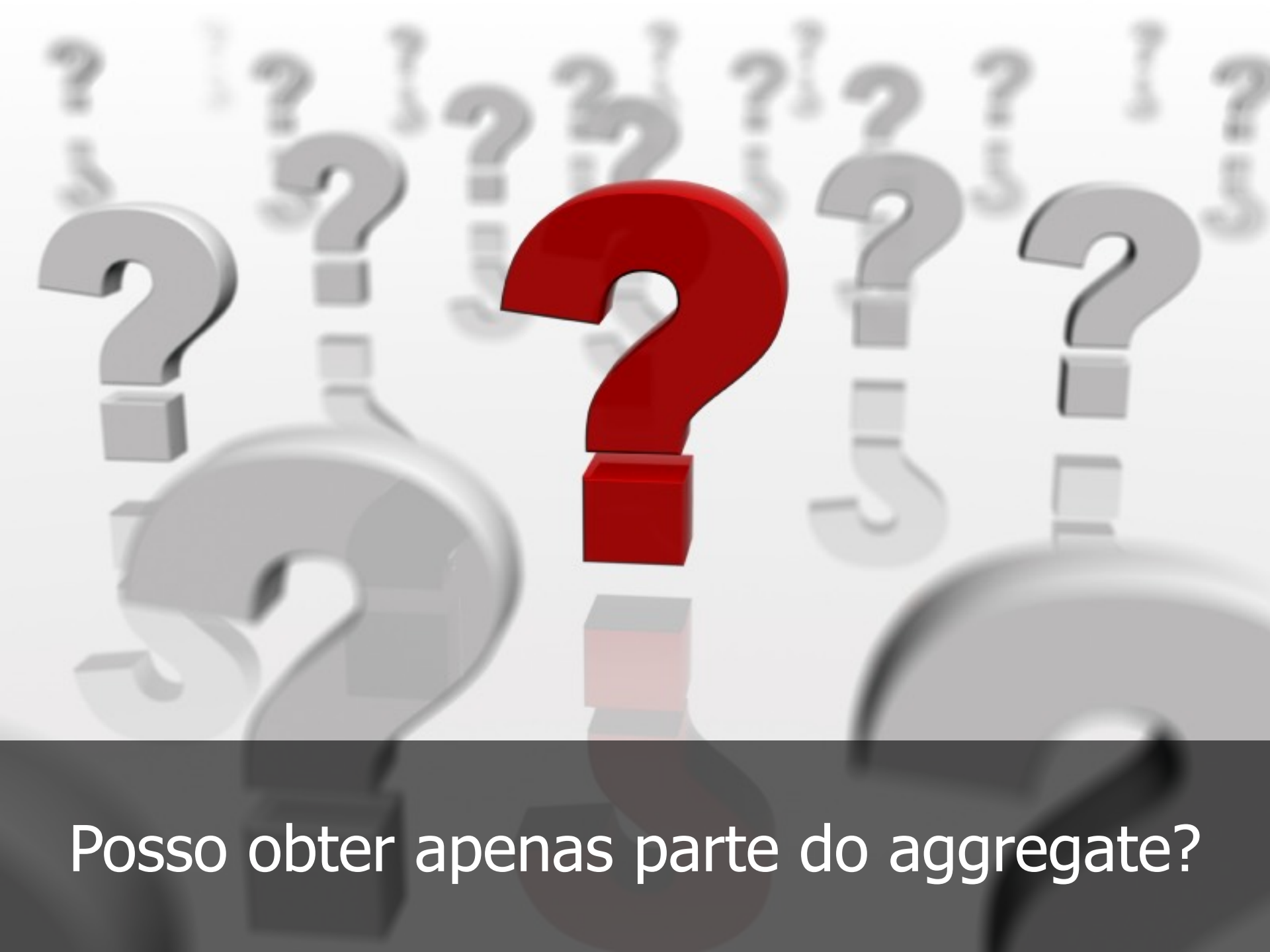
Qual é a diferença do padrão Repository no Domain-Driven Design e o DAO?

Um repository lida a persistência de um aggregate inteiro, enquanto um DAO não tem uma granularidade definida



Somente parte do aggregate mudou, posso persistir apenas essa parte?

A persistência é sempre realizada sobre o aggregate inteiro, no entanto a implementação do repository pode decidir quais registros banco de dados devem ser atualizados

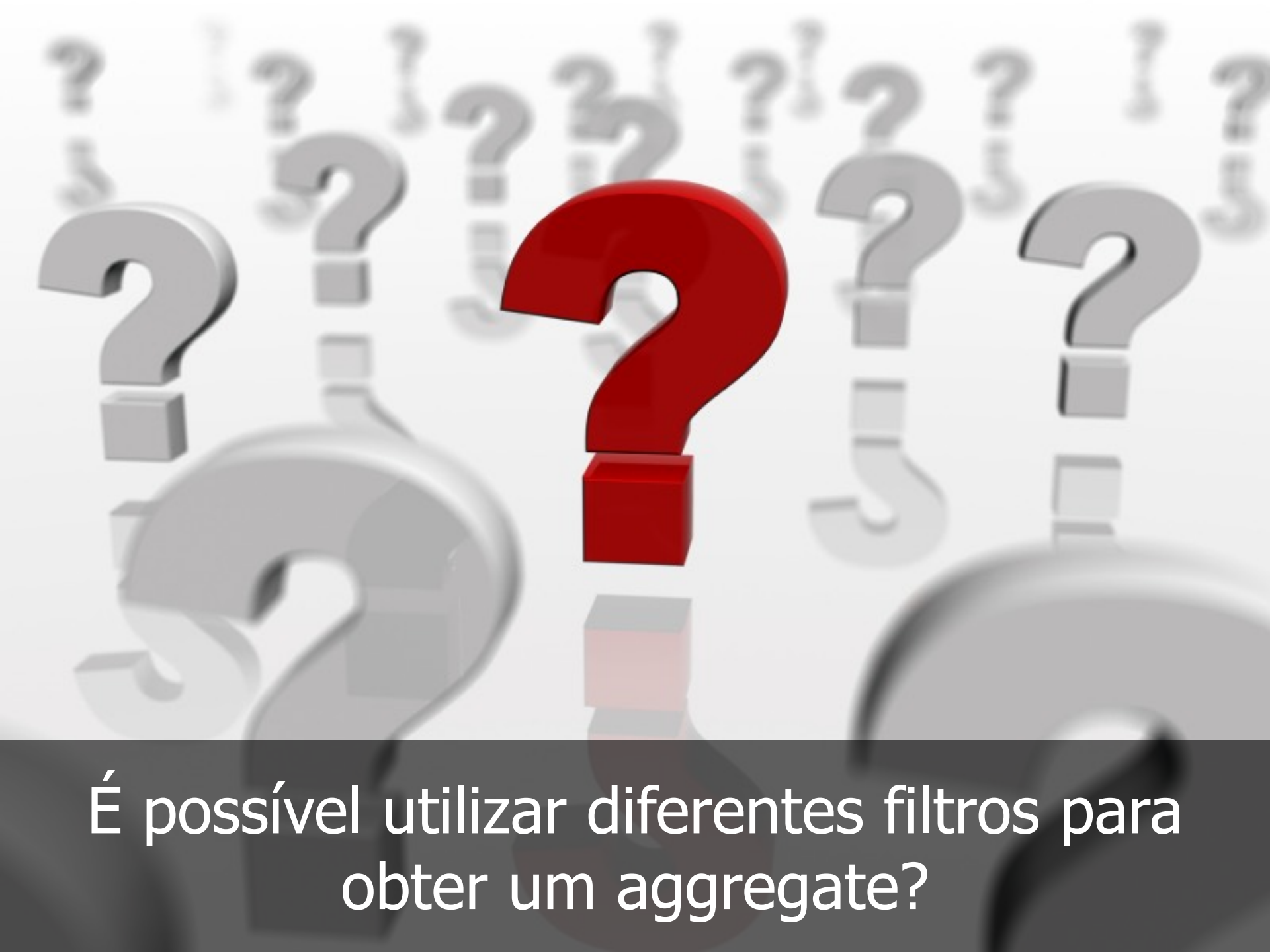


Posso obter apenas parte do aggregate?

Isso significa que pode ser que o aggregate seja grande mais e poderia ser quebrado em aggregates menores

Posso utilizar lazy loading dentro do aggregate?

A presenvação da invariância depende da integridade do aggregate, se parte dele não estiver populado pode perder o sentido



É possível utilizar diferentes filtros para obter um aggregate?

Com certeza, na obtenção do aggregate diversos filtros podem ser utilizados

Posso gerar dados para a emissão de um relatório a partir de um repository?

A granularidade de um relatório é diferente da utilizada pelo aggregate e renderizar relatórios a partir de repositories pode ser excessivamente complexo, prefira a utilização de CQRS com a criação de consultas separadas