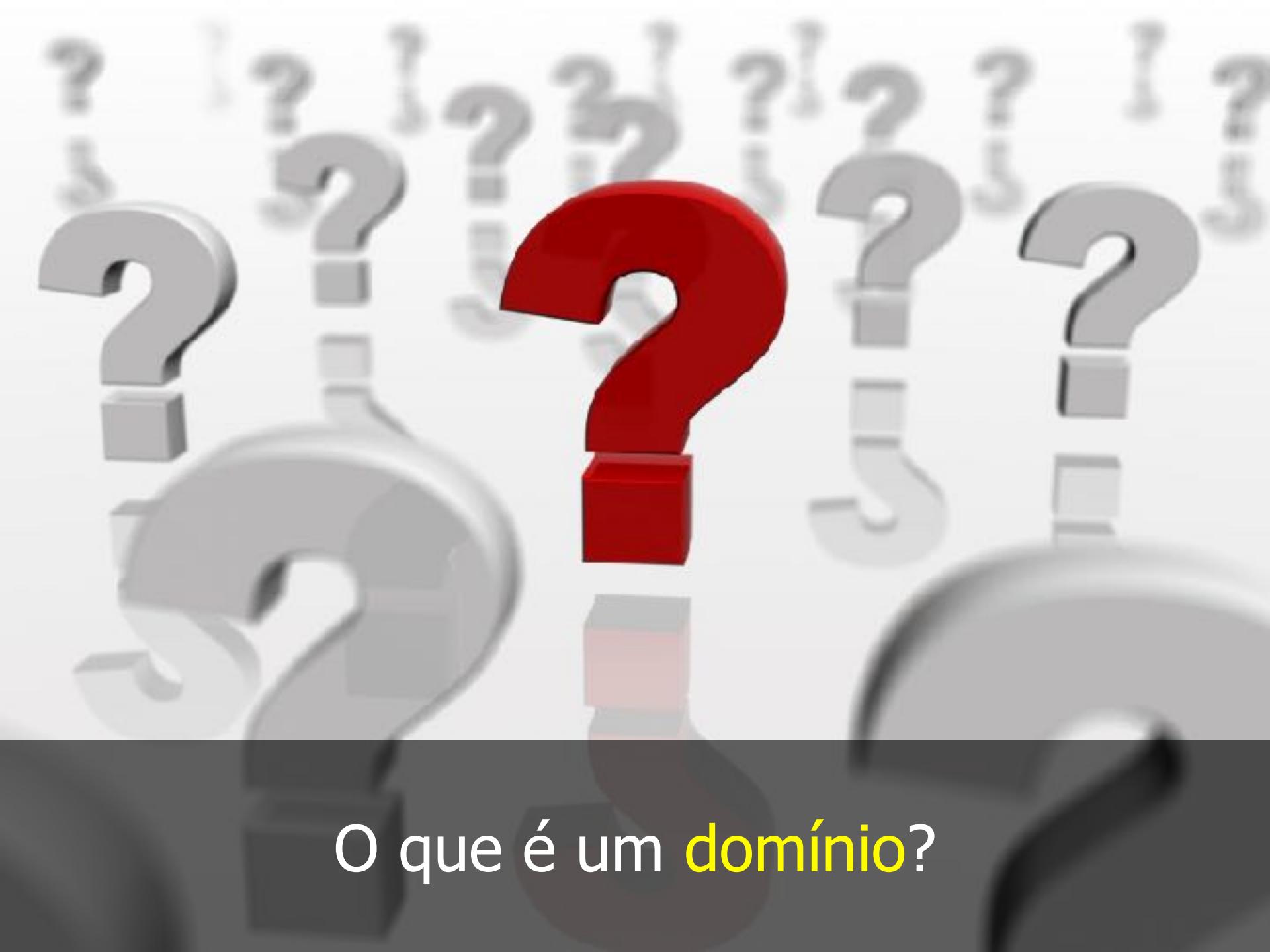


A blurry background image showing a person's hands working on a large jigsaw puzzle spread out on a table. Several puzzle pieces are scattered in the foreground, and a box with a picture of the completed puzzle is visible in the upper right corner.

Domain-Driven Design - Parte 1



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

Exemplos

- Emissão de nota fiscal de serviço
- Geração da folha de pagamento
- Concessão de um financiamento imobiliário
- Apuração de impostos
- Escrituração contábil
- Cálculo de comissão de vendas
- Precificação de seguro automotivo
- Aprovação de um aluno
- Processamento de pagamento recorrente no cartão de crédito
- Realização de uma venda online



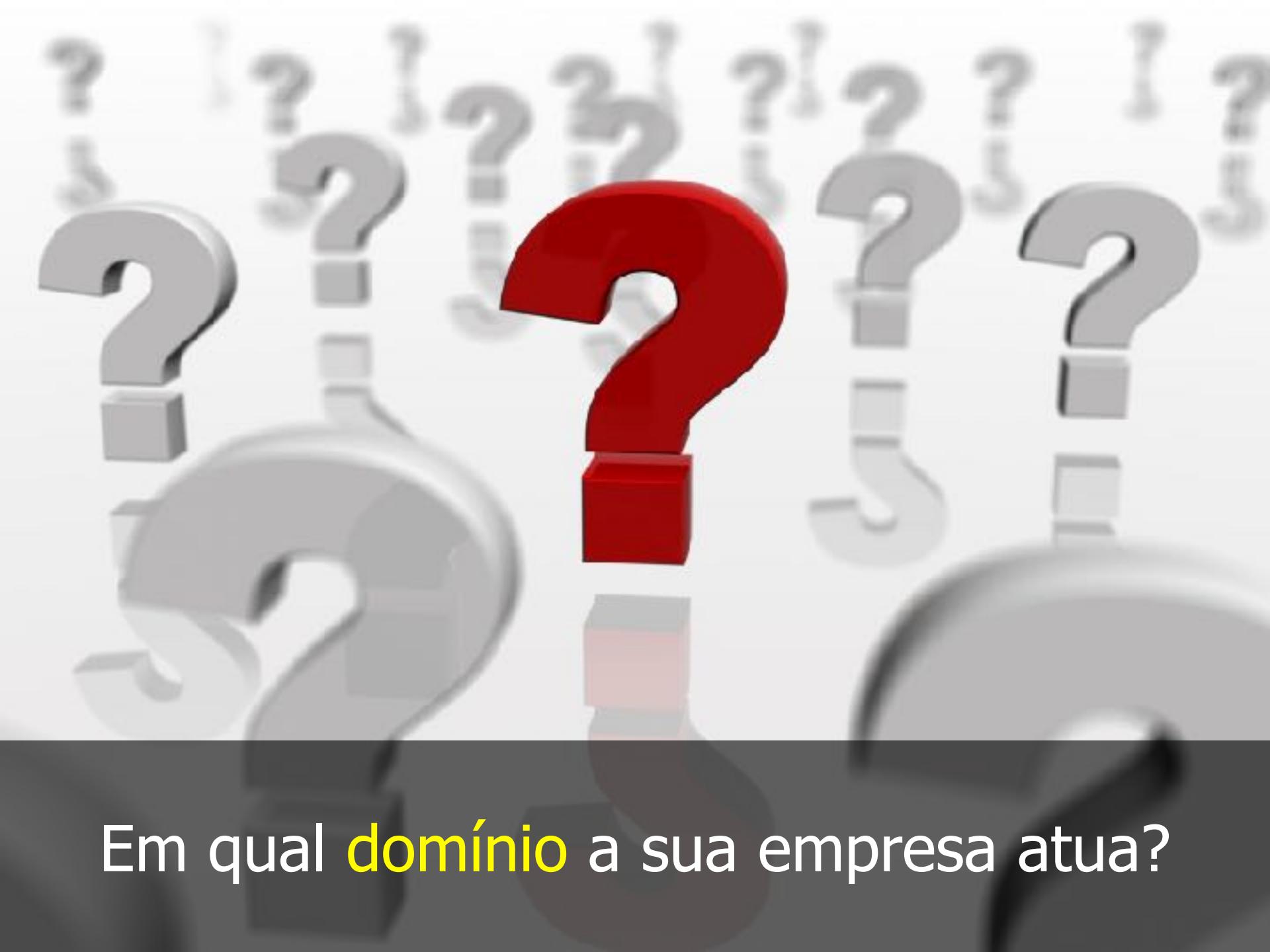
Com o tempo, principalmente em um **domínio complexo**, existe o risco disso acontecer...



São muitas pessoas envolvidas, existe a integração de diversas áreas de negócio



Normalmente acontece um fenômeno
conhecido como **Big Ball of Mud**



Em qual **domínio** a sua empresa atua?



CAUTION

Nem sempre é fácil extrair o
conhecimento relacionado ao domínio



Já te pediram para desenvolver algo que
ninguém sabia exatamente o que era?



HOW THE CUSTOMER
EXPLAINED IT



HOW THE PROJECT
LEADER UNDERSTOOD IT



HOW THE ANALYST
DESIGNED IT



HOW THE PROGRAMMER
WROTE IT



WHAT THE BETA
TESTERS RECEIVED



HOW THE CONSULTANT
DESCRIBED IT



HOW THE PROJECT
WAS DOCUMENTED



WHAT OPERATIONS
INSTALLED



HOW THE CUSTOMER
WAS BILLED



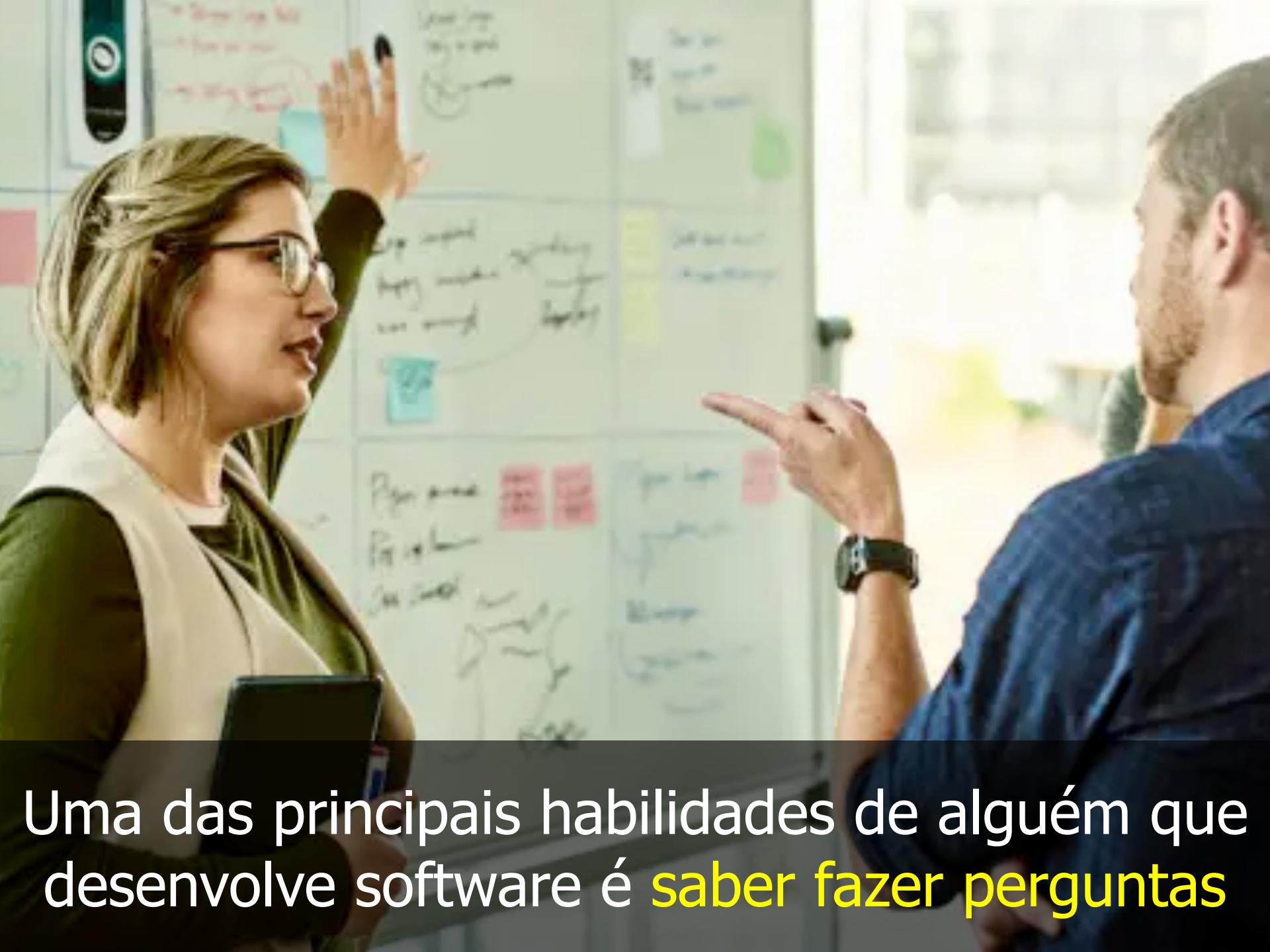
HOW THE PROJECT
WAS SUPPORTED



WHAT MARKETING
ADVERTISED

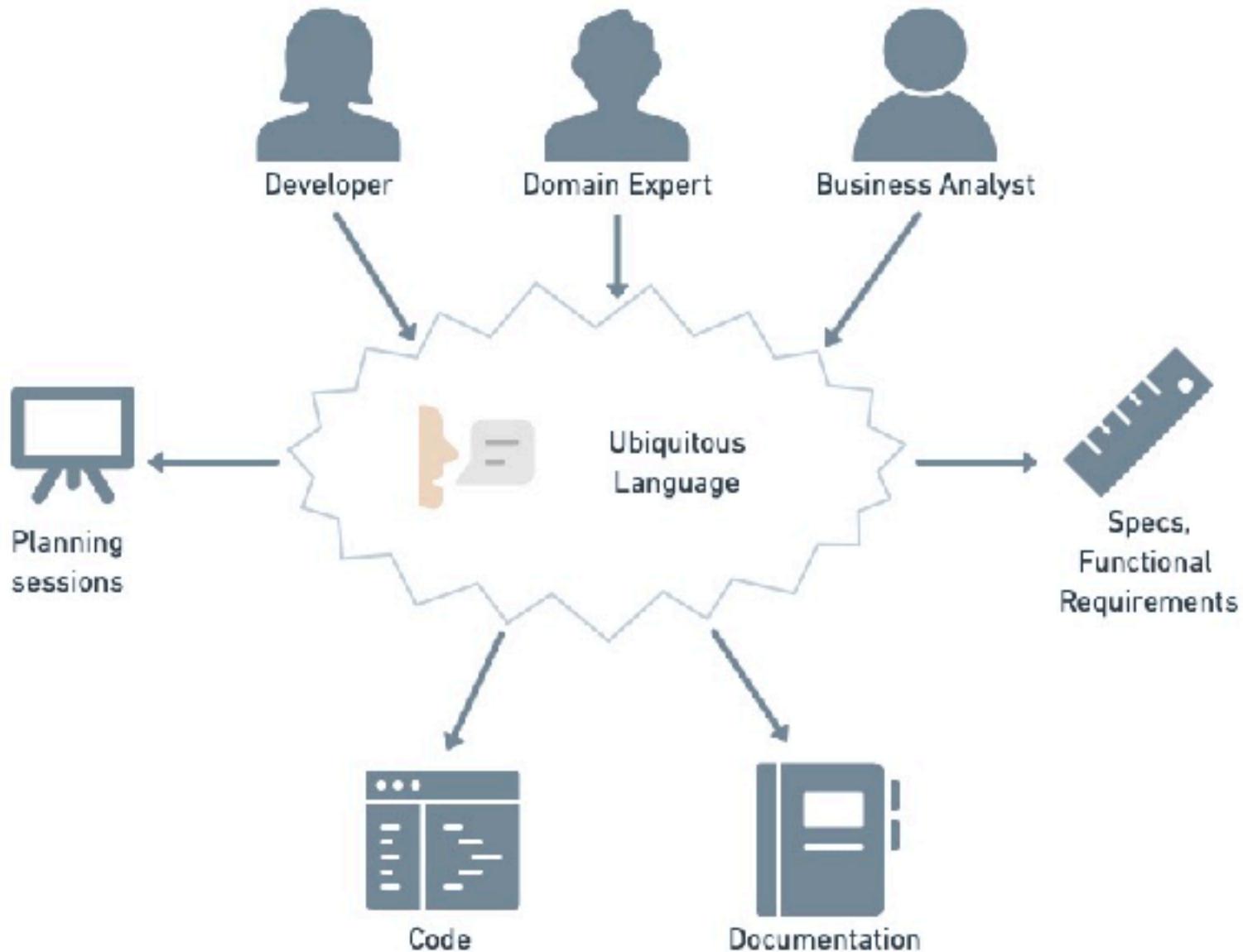


WHAT THE CUSTOMER
REALLY NEEDED

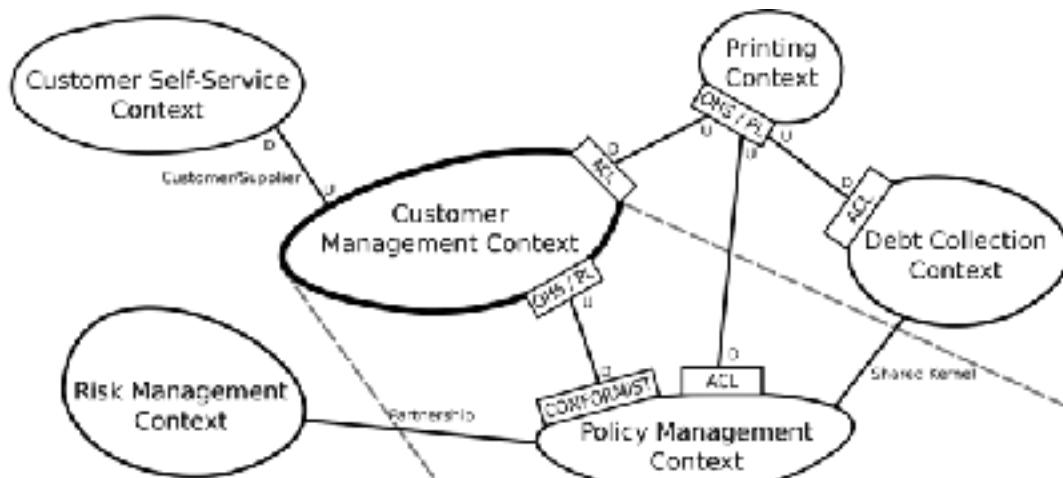


Uma das principais habilidades de alguém que desenvolve software é **saber fazer perguntas**

A linguagem ubíqua é falada entre os especialistas no domínio e a equipe de desenvolvimento dentro de um determinado contexto delimitado, devendo ser refletida dentro do código-fonte

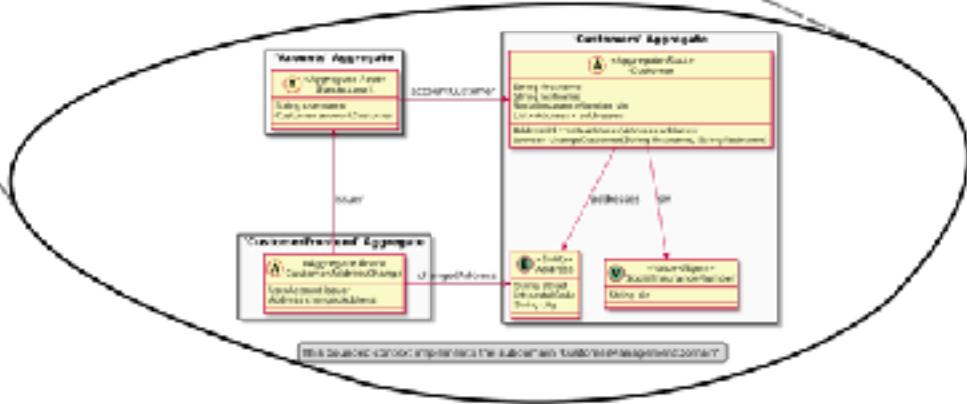


O Domain-Driven Design se divide em duas partes:



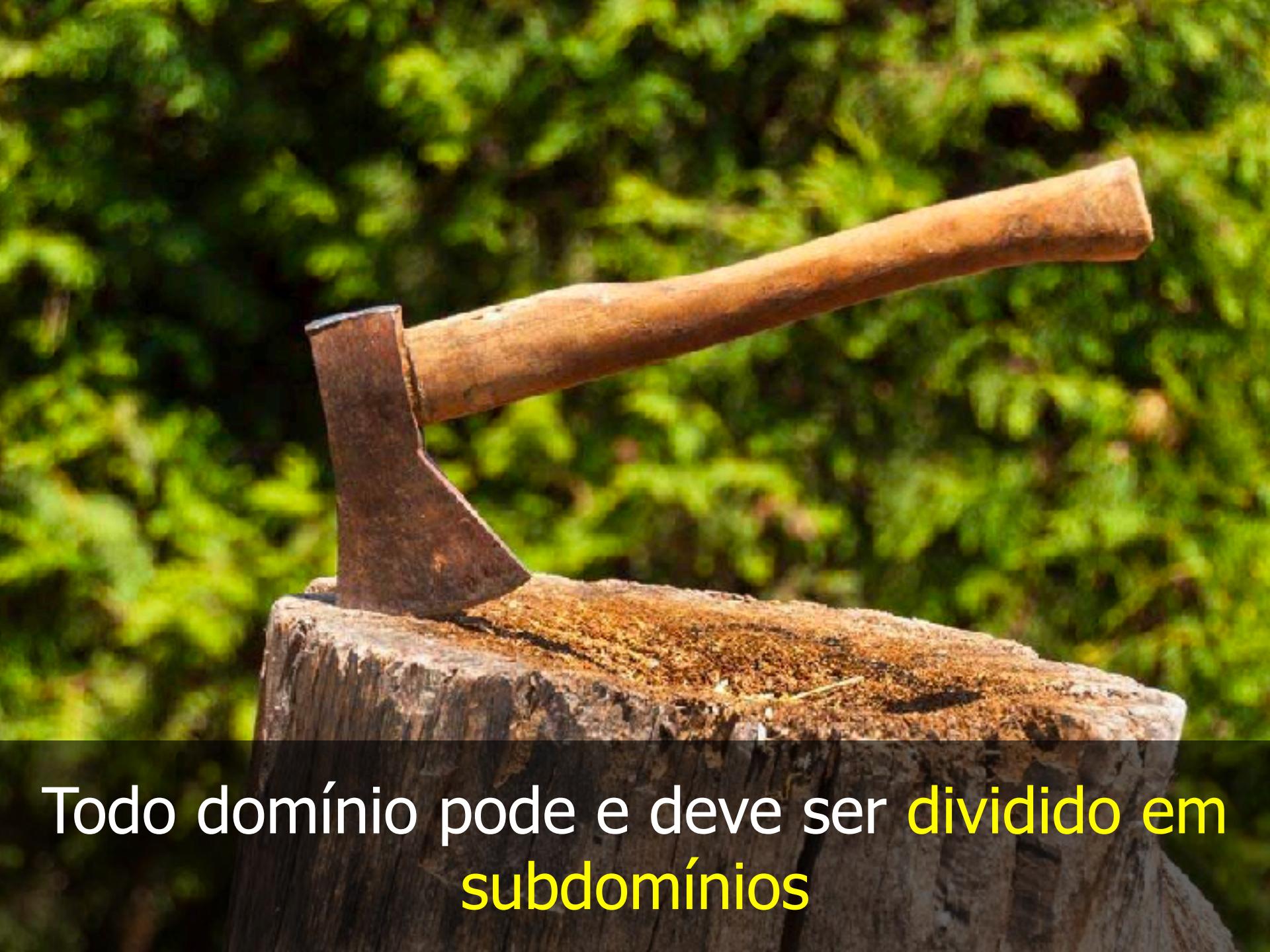
Strategic DDD:
Context Map

Tactical DDD:
Domain Model
inside a Bounded
Context





A modelagem estratégica identifica e define
as fronteiras entre os bounded contexts

A close-up photograph of a wooden ax handle with a dark metal head, stuck diagonally into a weathered tree stump. The background is a blurred green forest. The text is overlaid on the bottom left of the image.

Todo domínio pode e deve ser dividido em
subdomínios

Tipos de subdomínio

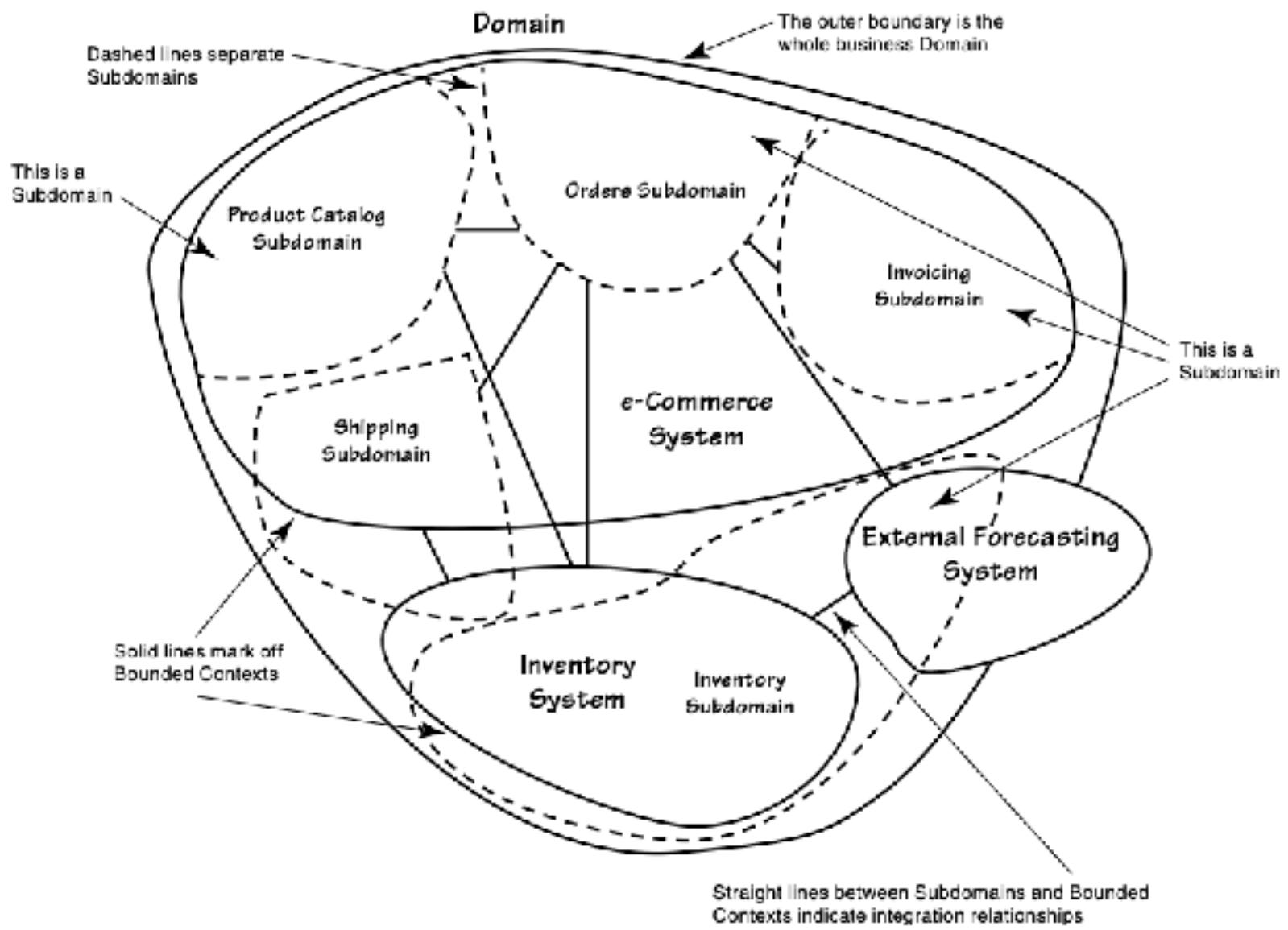
Core ou Basic: É o mais importante e que traz mais valor para a empresa, é onde você coloca seus maiores e melhores esforços

Support ou Auxiliary: Complementa o core domain, sem ele não é possível ter sucesso no negócio

Generic: É um subdomínio que pode ser delegado para outra empresa ou mesmo ser um produto de mercado



Em um sistema de **vendas de produtos online**,
quantos subdomínios diferentes temos?



Realização de uma venda online

- Recomendação de produtos
- Processamento do pagamento
- Emissão da nota fiscal
- Gestão do estoque
- Acompanhamento da entrega

Emissão de nota fiscal de serviço

- Cálculo dos impostos e do valor da nota fiscal de acordo com o regime contábil e considerando possíveis descontos aplicados
- Integração com as prefeituras
- Exportação das notas fiscais emitidas para o ERP

Processamento de pagamento recorrente no cartão de crédito

- Gestão da recorrência
- Integração com os adquirentes
- Análise antifraude

Concessão de um financiamento imobiliário

- Cálculo e provisionamento das parcelas
- Análise de documentos
- Emissão e gestão de assinatura do contrato



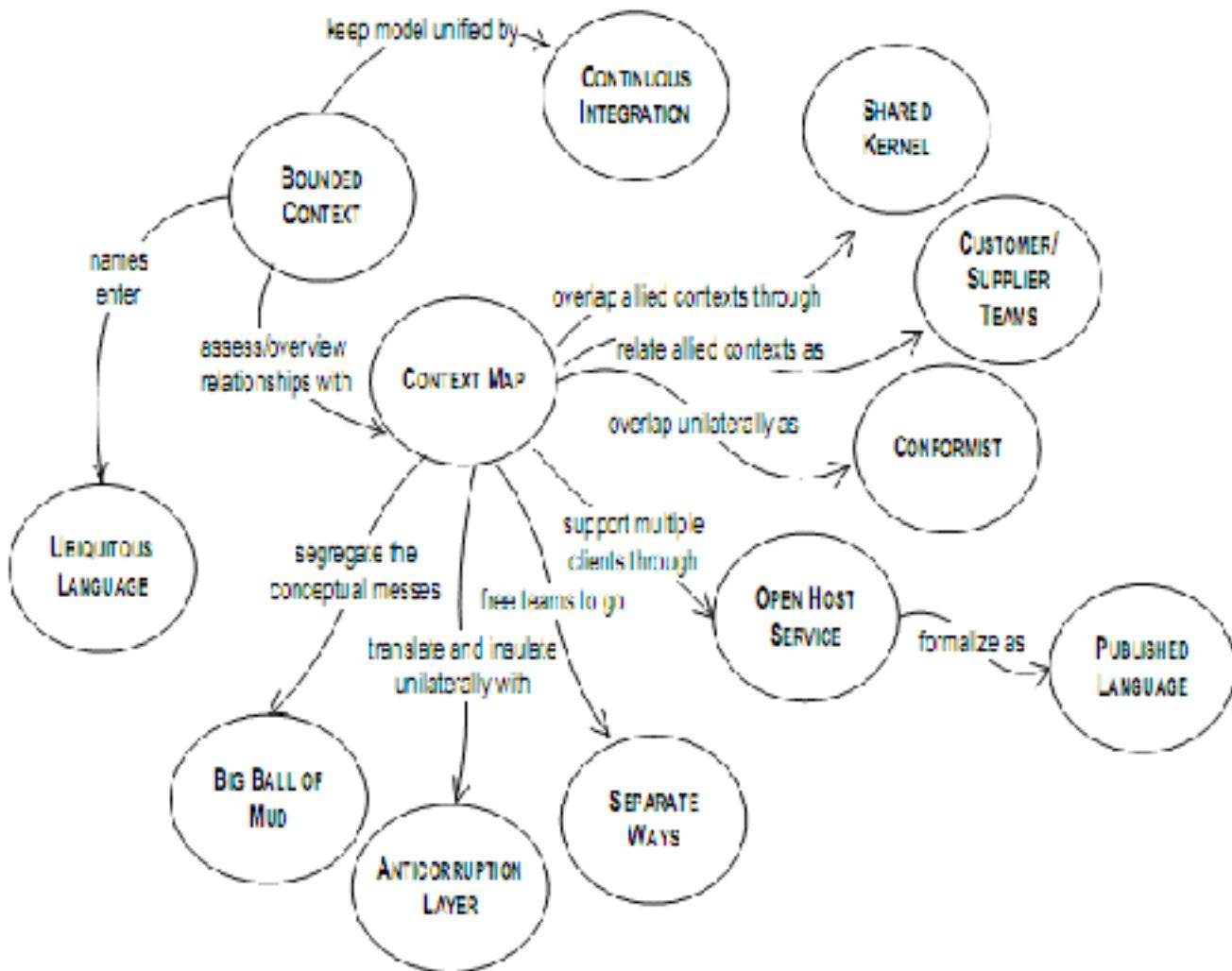
Posso implementar tudo junto?

Existe uma relação de um para um (1:1) entre um subdomínio e um bounded context

Imagine um bounded context como uma forma de modularização de negócio que tem como objetivo reduzir o acoplamento interno do código-fonte

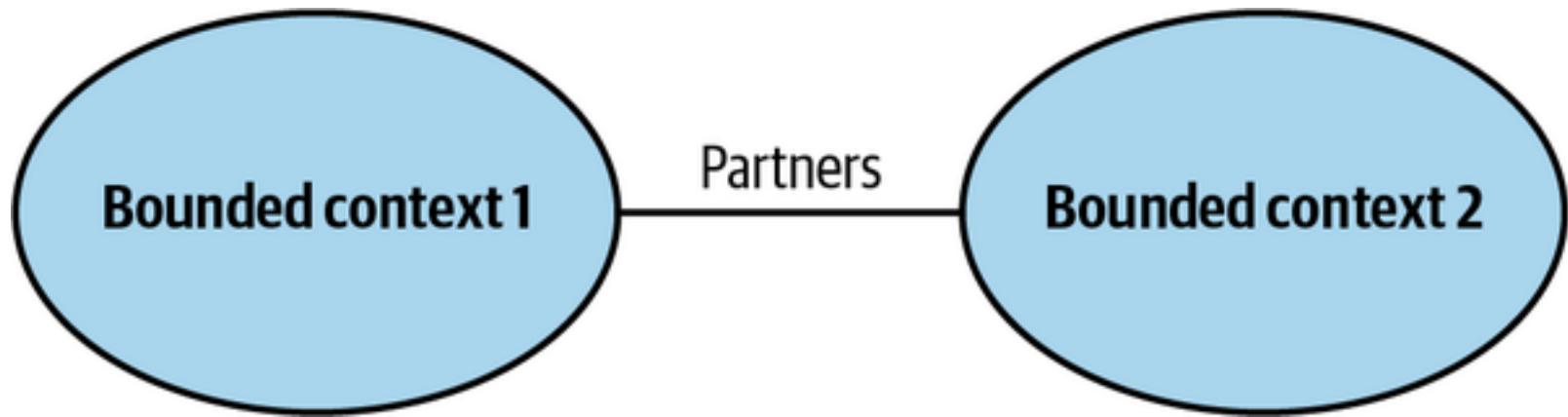


A forma de interação entre cada bounded context dá origem ao **context map**



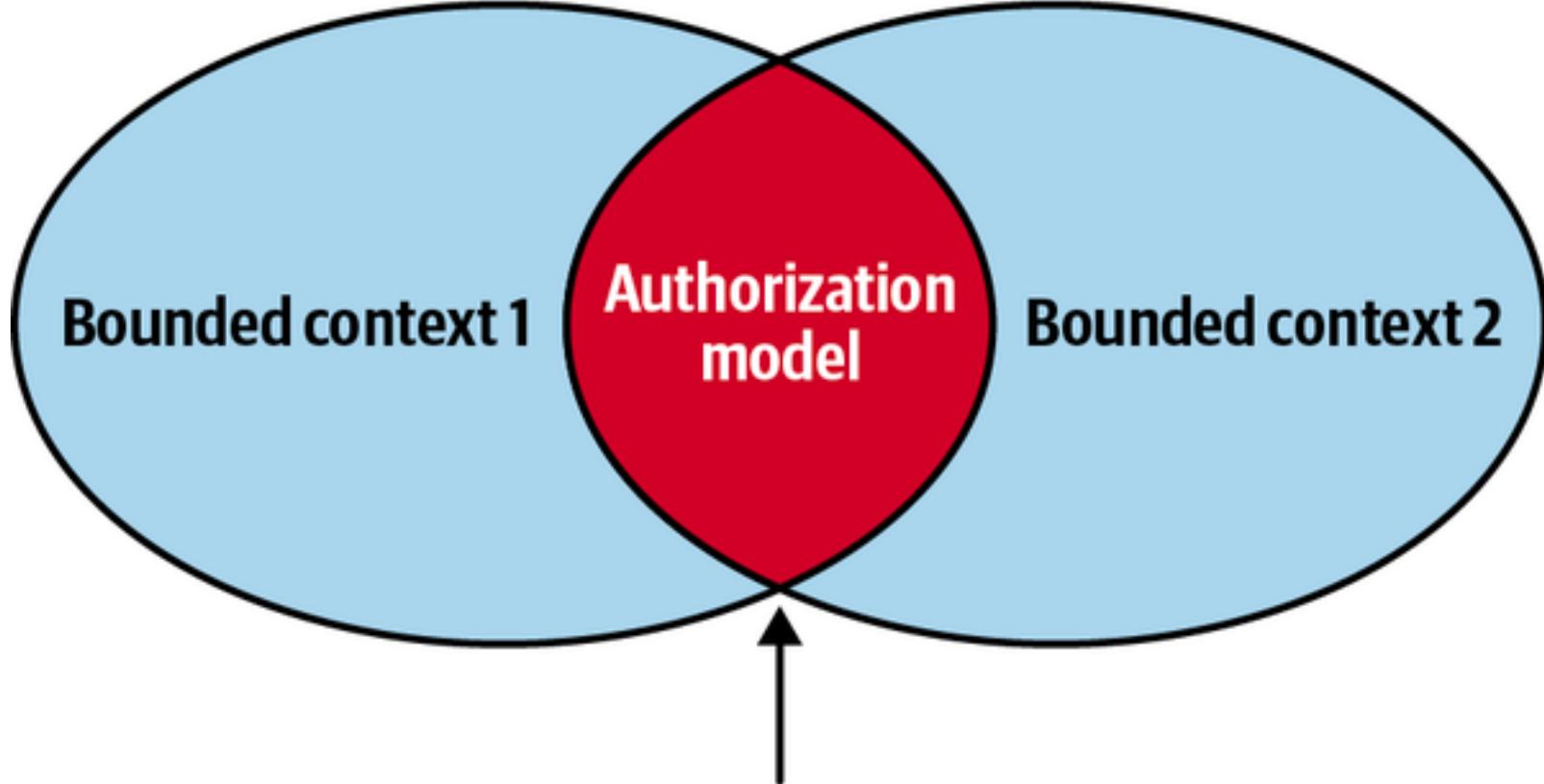
Integration Patterns

Os padrões de integração definem naturalmente o tipo de relacionamento entre cada bounded context



Partnership

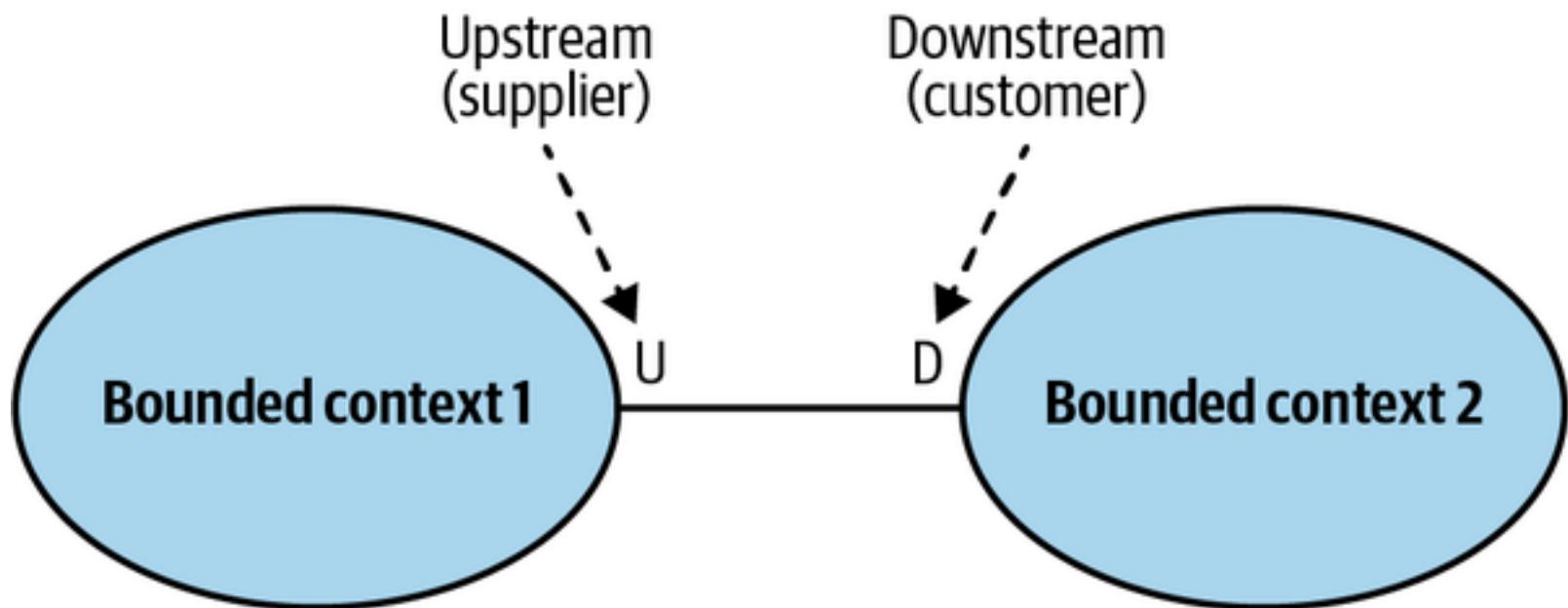
Duas ou mais equipes podem trabalhar de forma sincronizada numa entrega que envolve dois ou mais bounded contexts



Shared Kernel

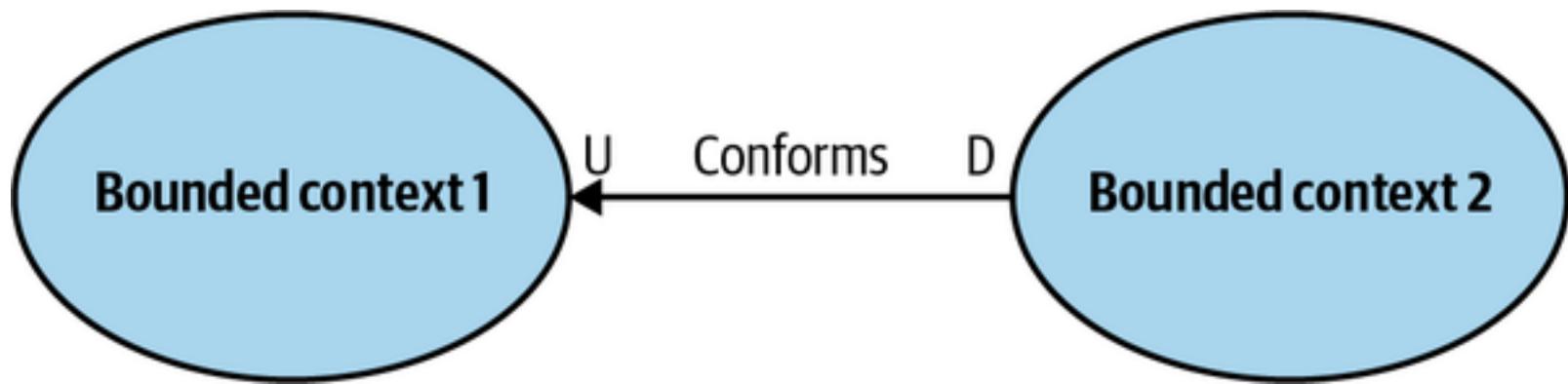
É relativamente normal compartilhar parte do código comum entre vários bounded contexts, principalmente por propósitos não relacionados diretamente ao negócio mas por infraestrutura

Em termos mais técnicos, o código pode ser compartilhado por meio do relacionamento direto em um monorepo ou algum tipo de biblioteca que deve ser versionada e publicada internamente para que possa ser importada pelos outros bounded contexts



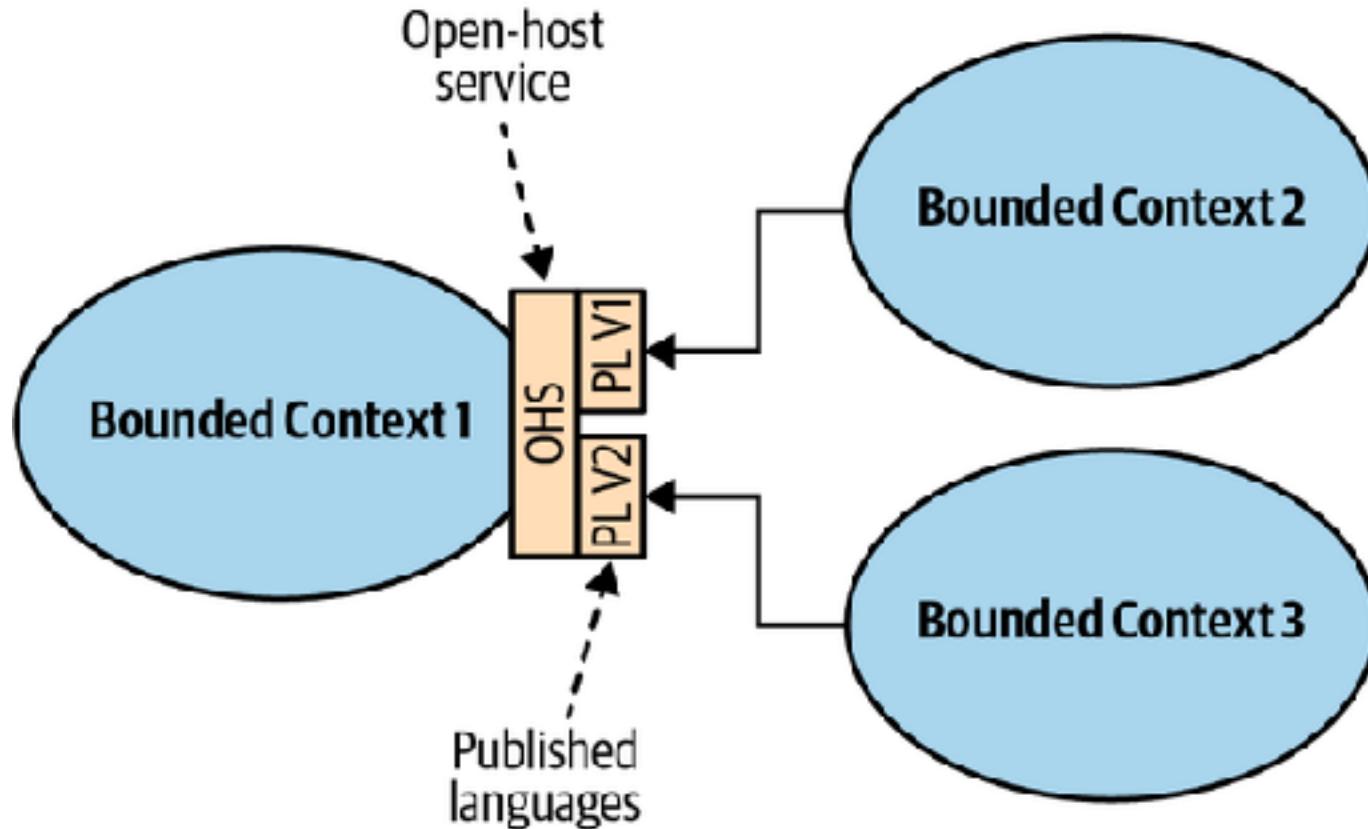
Customer/Supplier

Existe uma **relação de fornecimento** onde tanto o customer quanto o supplier podem determinar como deve ser o contrato entre eles



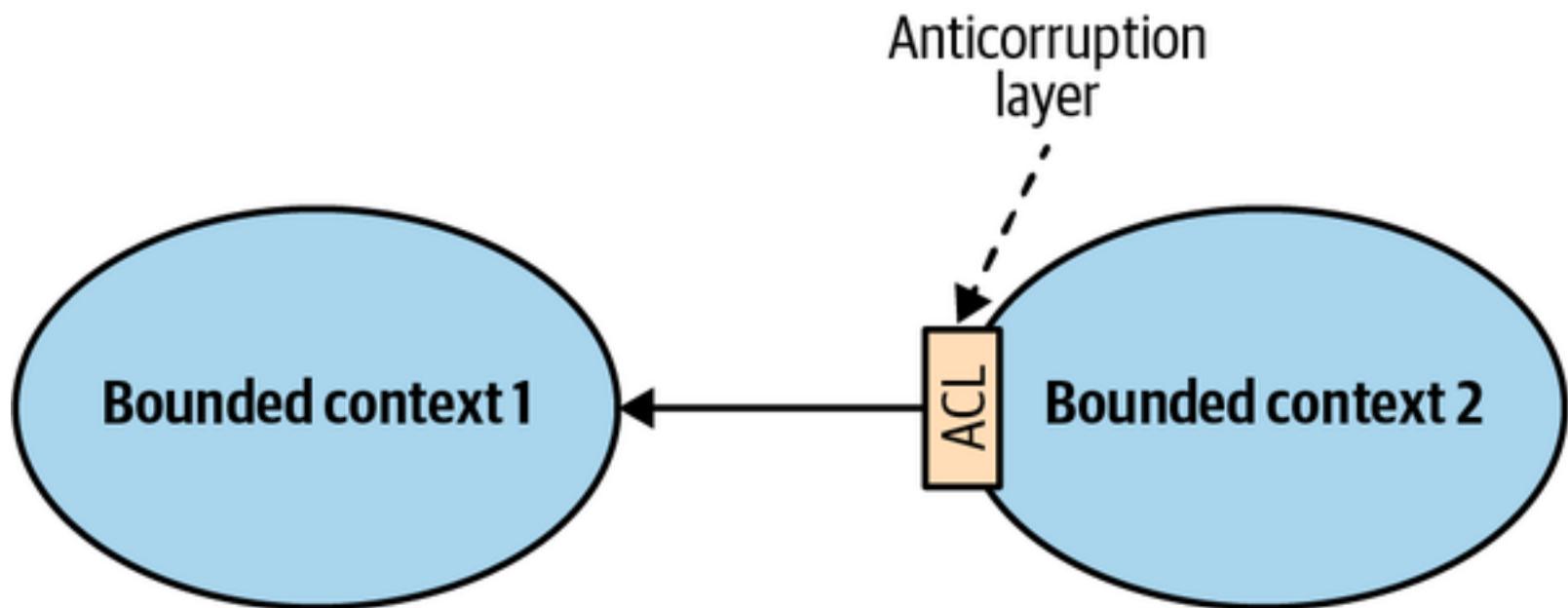
Conformist

Uma integração com uma API externa, contratada no modelo SaaS, acaba quase sempre sendo do tipo conformista já que temos que nos adequar a sua interface, nesses casos é normal oferecer um Open Host Service com uma Published Language



Open-Host Service

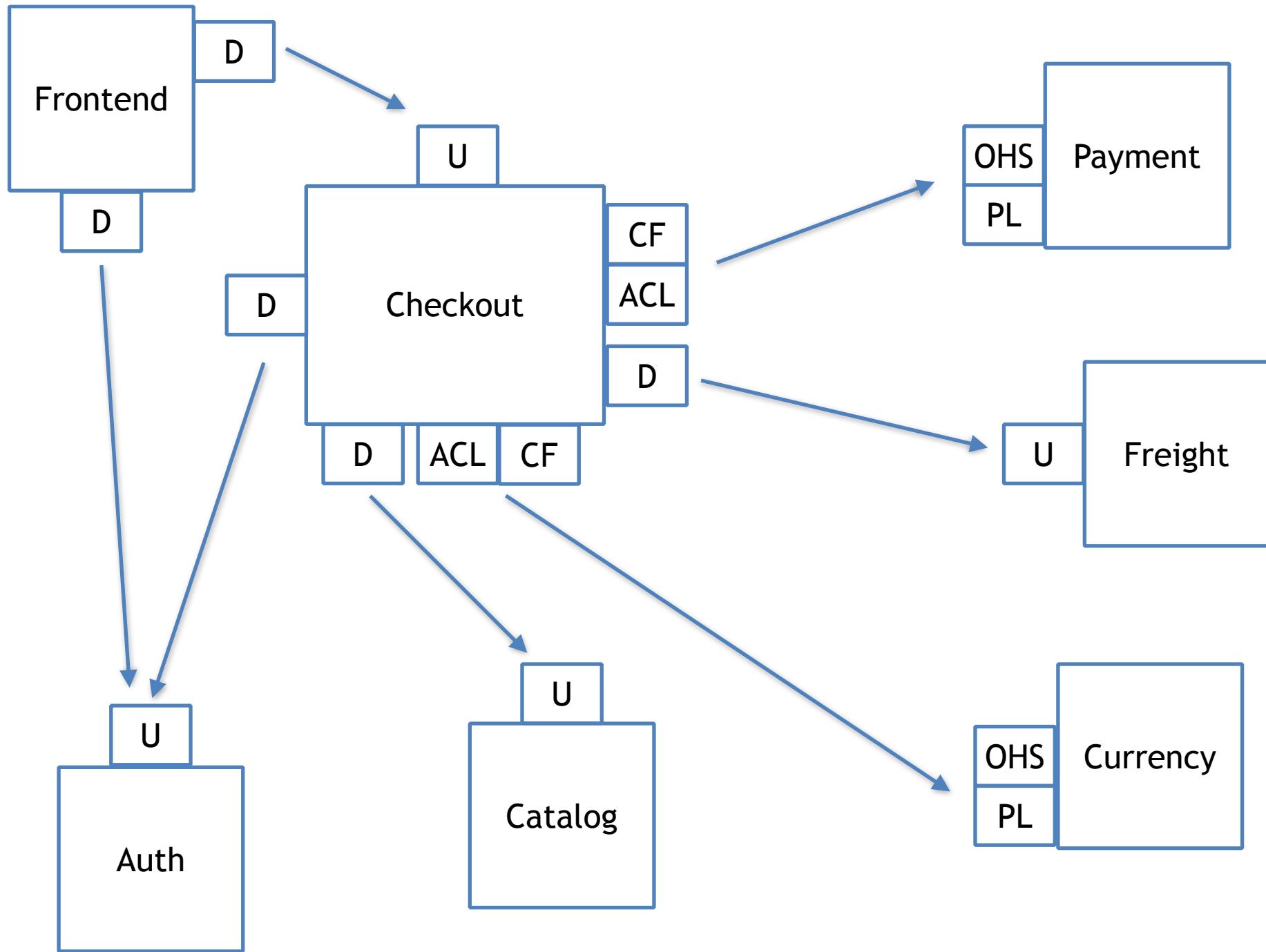
Um bounded context pode disponibilizar um conjunto de serviços utilizando um protocolo padrão e com uma documentação abrangente para quem tiver interesse em integrar

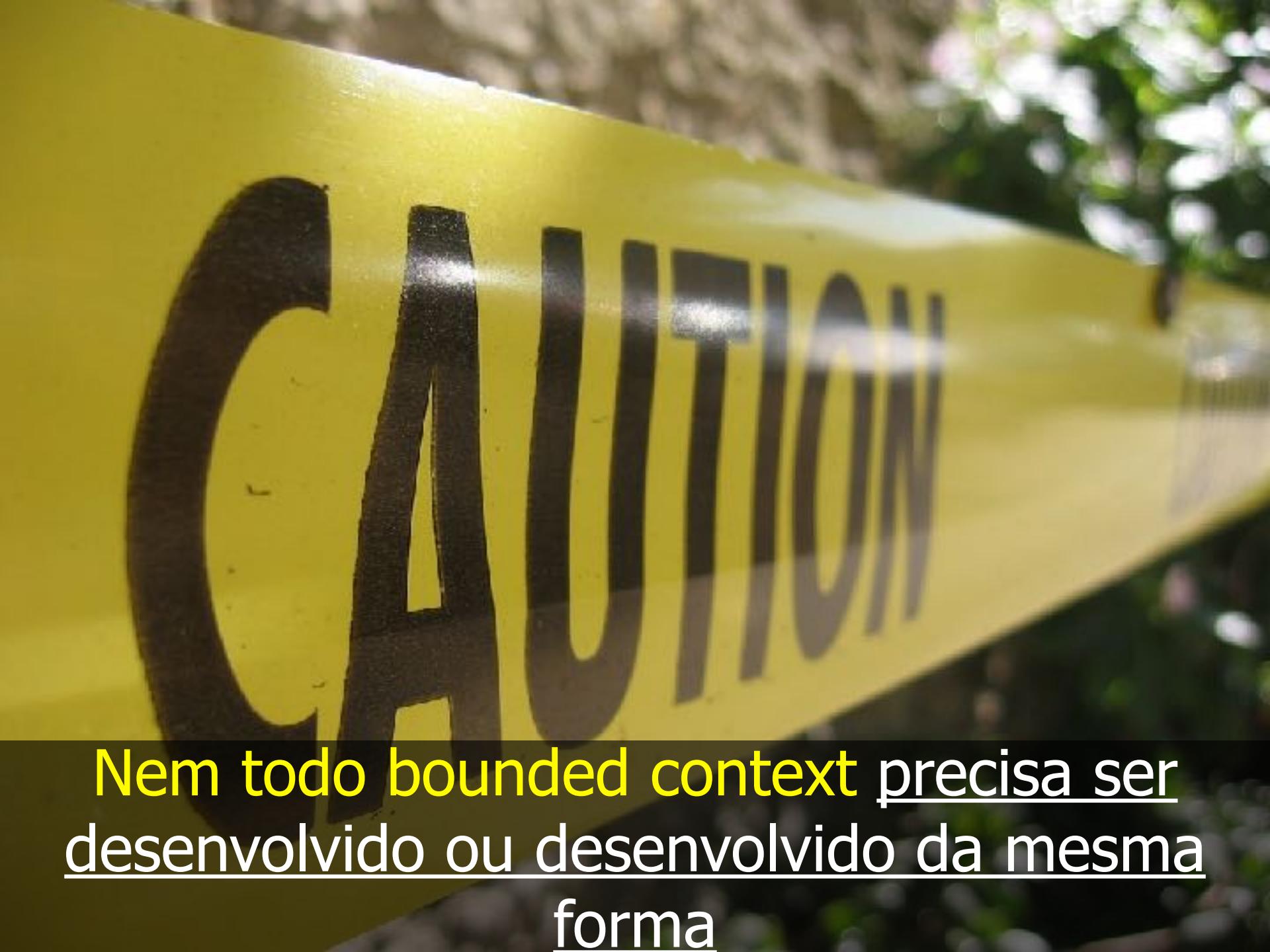


Anti-Corruption Layer

As relações conformistas geralmente exigem uma tradução para o domínio e isso pode ser feito por meio de adaptadores importantes para inclusive permitir a utilização de diferentes fornecedores

Eventualmente vale mais a pena ir por caminhos diferentes e não ter qualquer tipo de relação



A close-up photograph of a yellow caution tape. The word "CAUTION" is printed in large, bold, black capital letters. The tape is slightly curved, and the background is blurred green foliage.

CAUTION

Nem todo bounded context precisa ser
desenvolvido ou desenvolvido da mesma
forma



A fronteira do bounded context é
excelente para **definir um microservice**





Quais são as vantagens e desvantagens em ter uma arquitetura de **microservices**?

Vantagens

- Diversidade tecnológica
- Melhor controle sobre o débito técnico
- Facilidade em acompanhar a evolução tecnológica

Desafios

- Transações distribuídas
- Dificuldade em tratar e diagnosticar erros
- Complexidade técnica mais alta

Fazendo uma boa modelagem estratégica

- Divisão da complexidade
- Equipes menores
- Reuso

Comunicação assíncrona, Event-Driven Architecture, CQRS

- Escalabilidade
- Independência entre os serviços
- Tolerância à falhas
- Resiliência



Uma arquitetura monolítica nem sempre é
ruim, muito pelo contrário!

Para projetos menores com equipes pequenas, principalmente no início da construção de um produto, é a arquitetura que dá mais resultado com o menor esforço e custo de infraestrutura

MonolithFirst

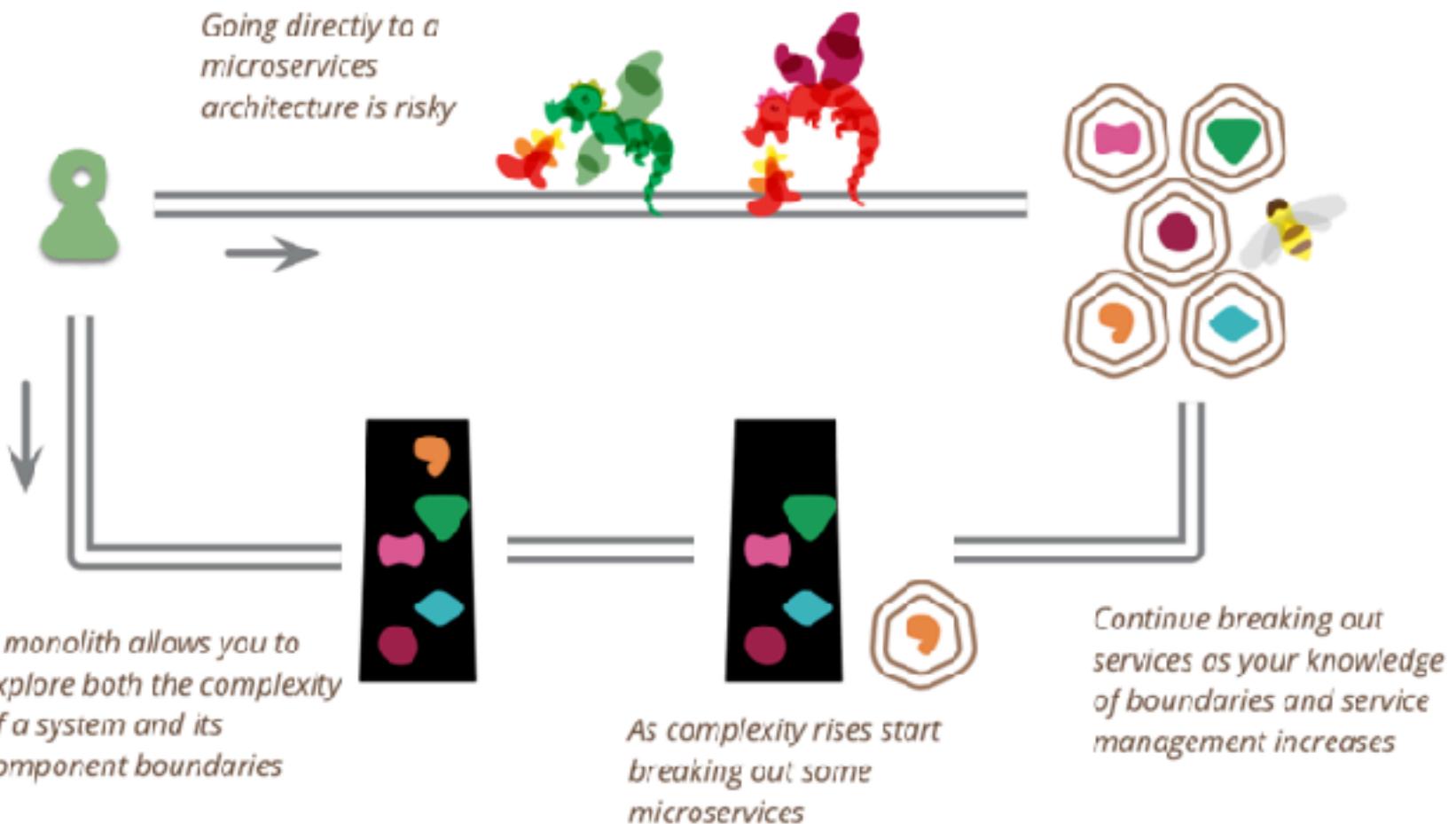


Martin Fowler

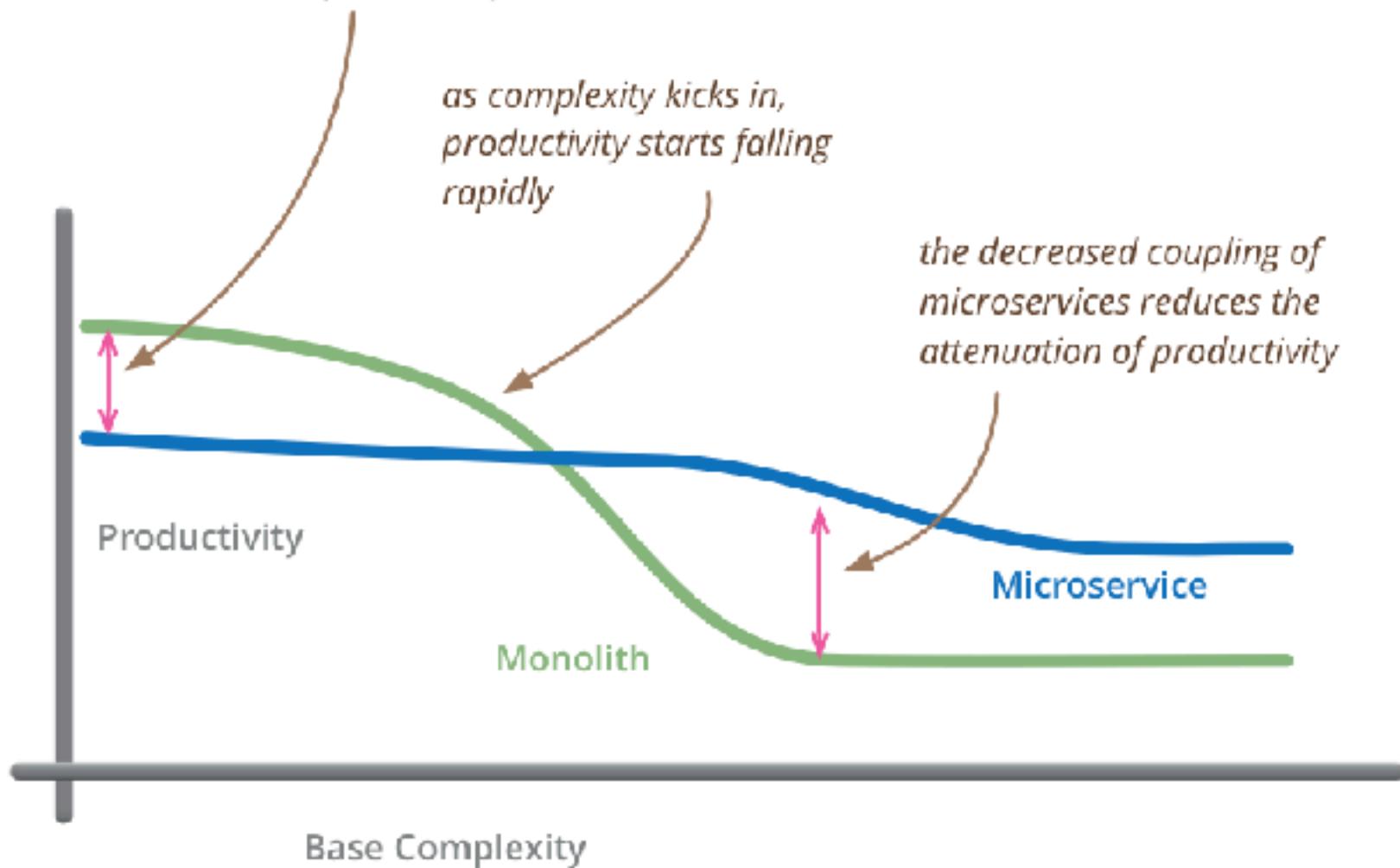
3 June 2015

As I hear stories about teams using a [microservices architecture](#), I've noticed a common pattern.

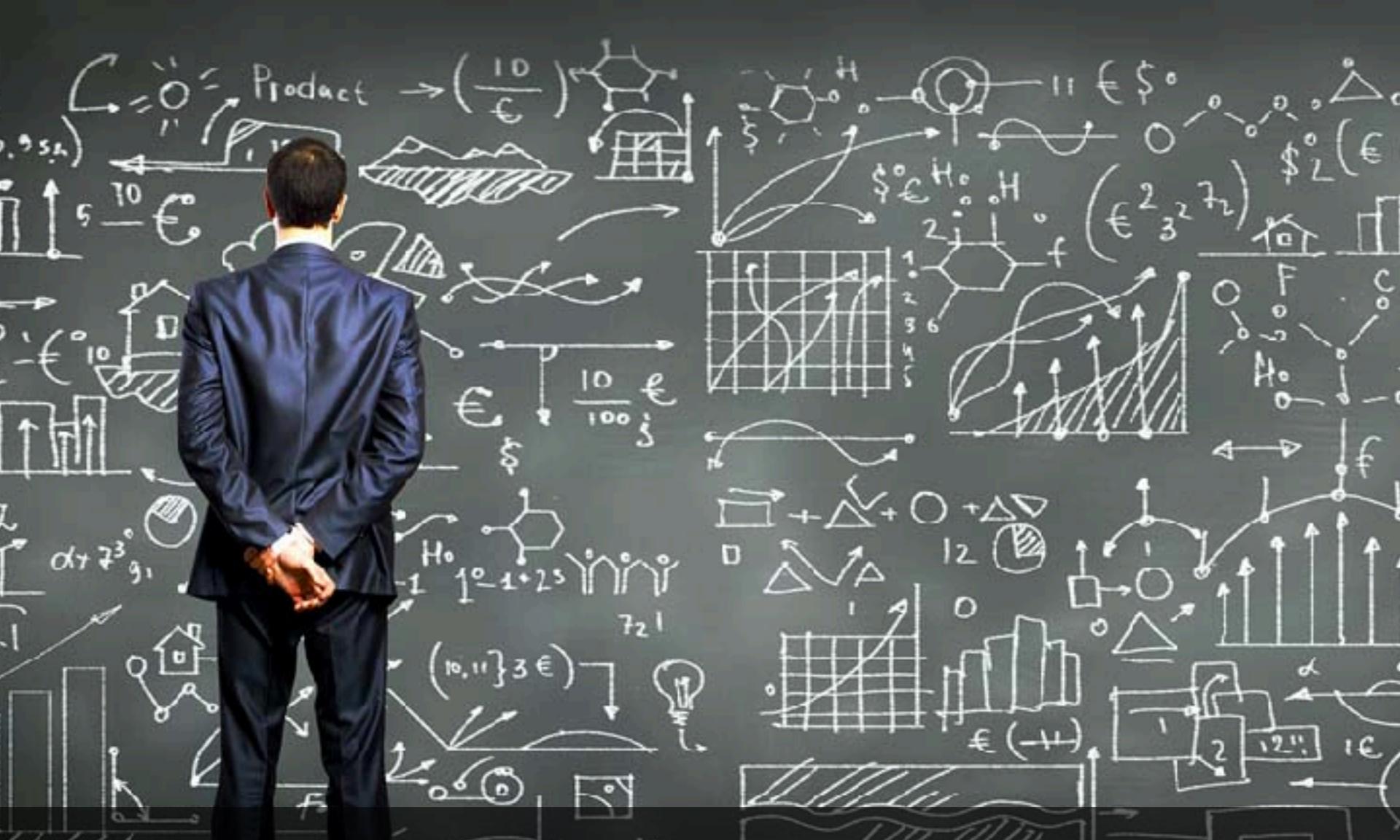
1. Almost all the successful microservice stories have started with a monolith that got too big and was broken up
2. Almost all the cases where I've heard of a system that was built as a microservice system from scratch, it has ended up in serious trouble.



for less-complex systems, the extra baggage required to manage microservices reduces productivity



but remember the skill of the team will outweigh any monolith/microservice choice



Leva **tempo** até entender qual é a melhor forma de dividir o domínio