

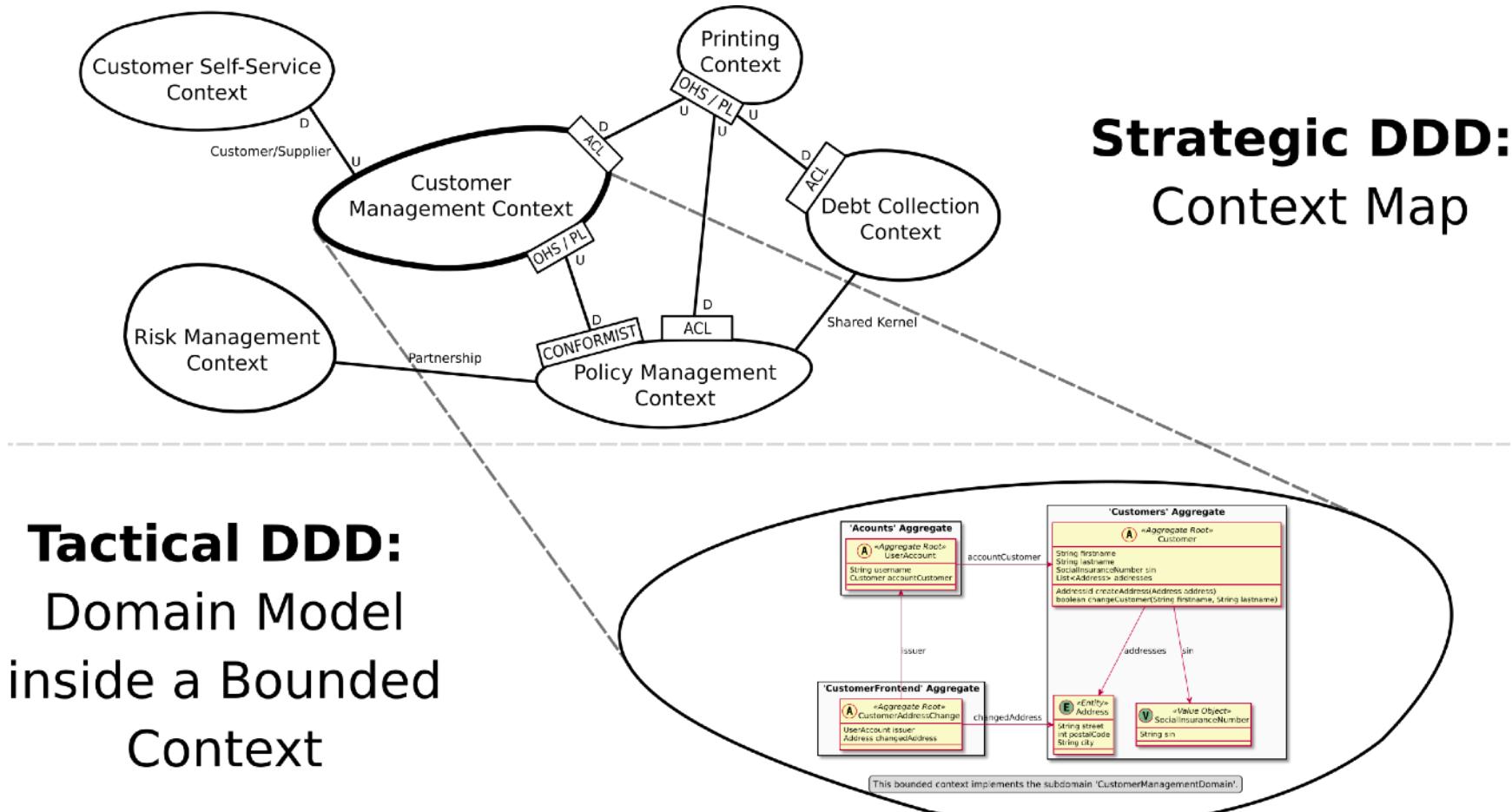
Open/Closed

Os componentes da arquitetura devem estar
abertos para extensão e fechados para
modificação

The background of the slide features a vibrant, abstract geometric painting in the style of Wassily Kandinsky. It is filled with bold, colorful shapes like circles, triangles, and rectangles in shades of red, blue, yellow, and black, set against a light beige background with dynamic black lines.

Domain-Driven Design Modelagem Estratégica

O Domain-Driven Design se divide em duas partes:





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**

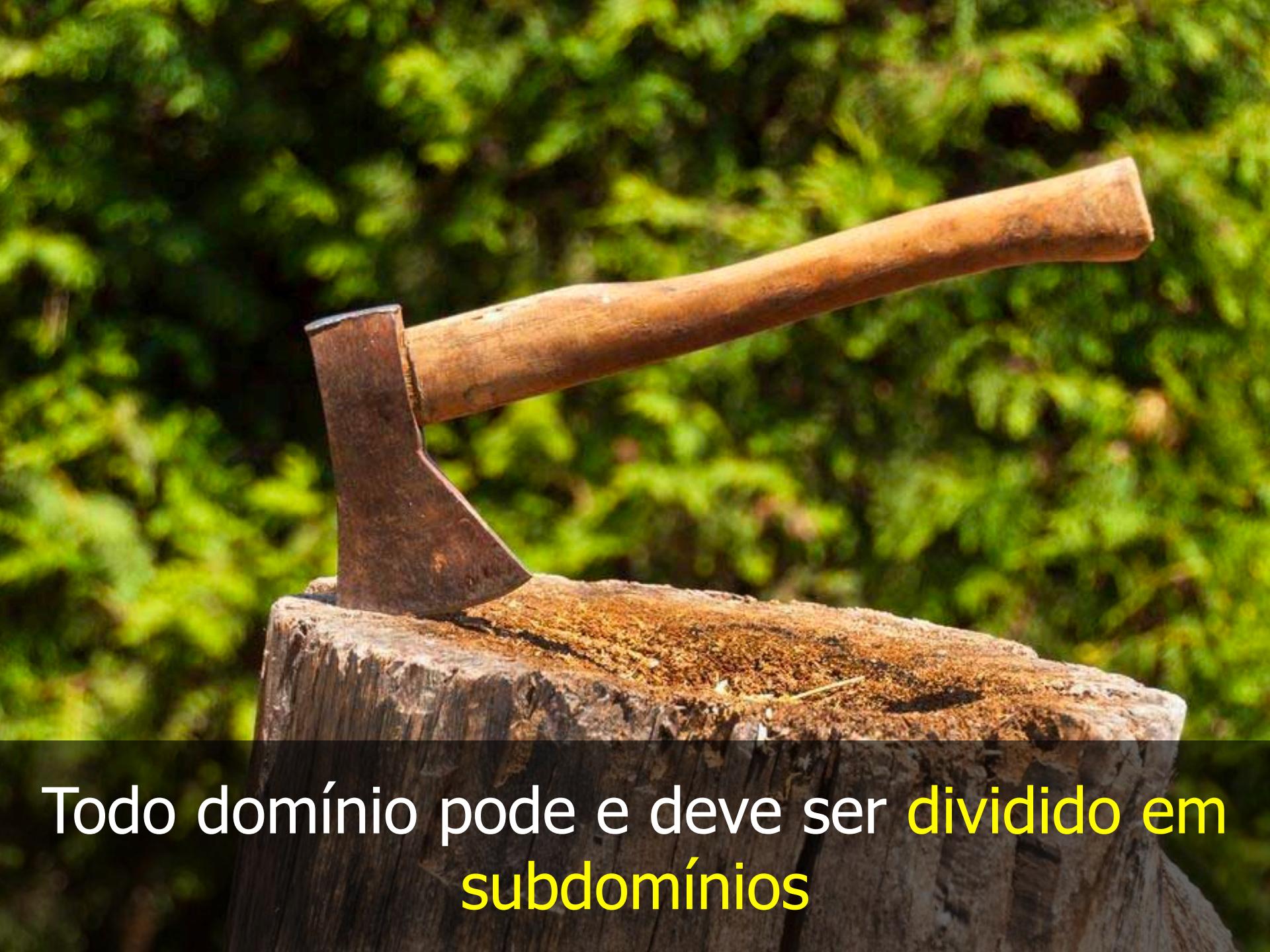


CAUTION

Excesso de separação leva ao aumento da
complexidade de integração



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



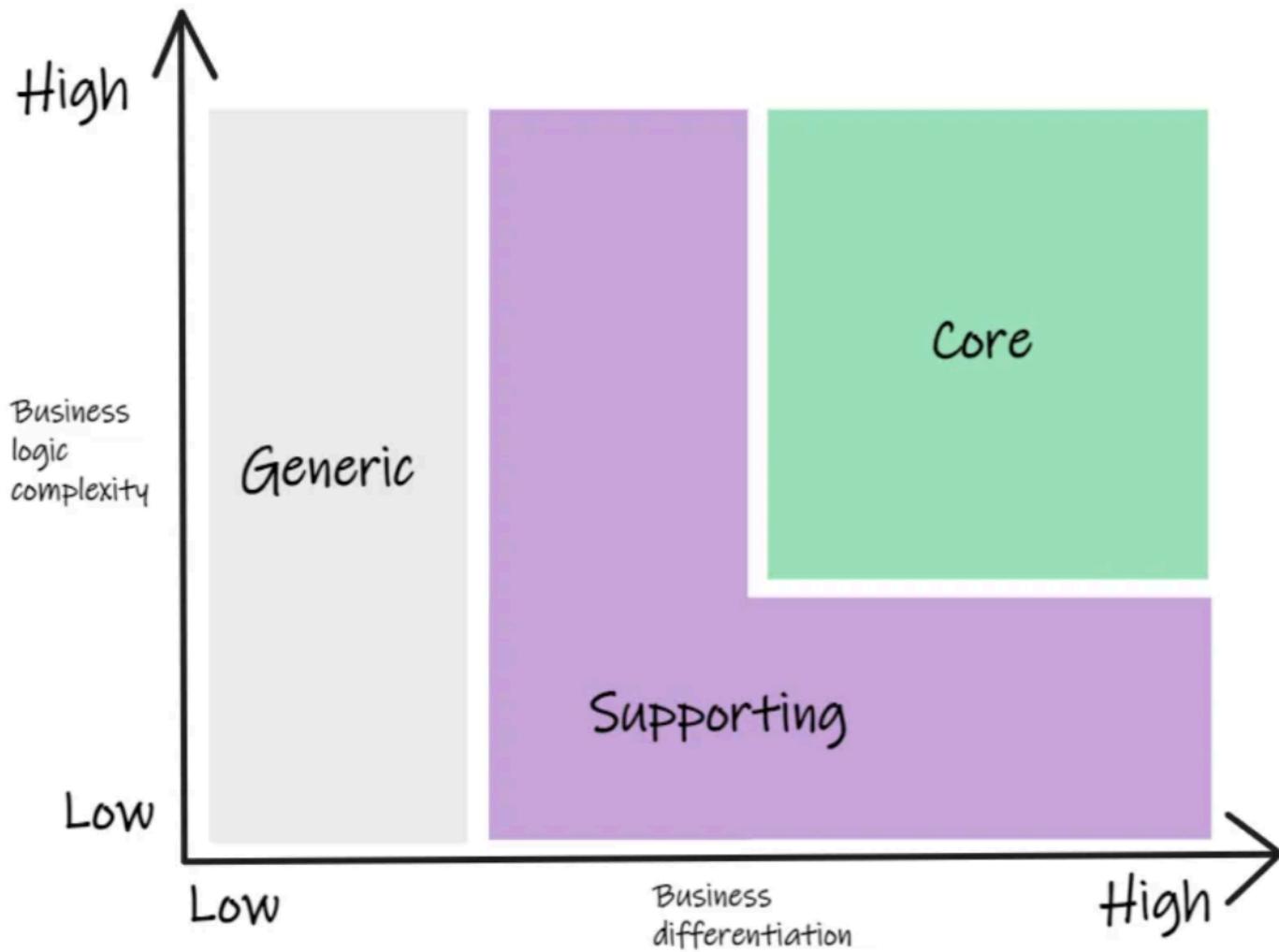
Todo domínio pode e deve ser dividido em
subdomínios

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 eles talvez seja difícil ter sucesso no negócio, mas você não precisa aplicar seus melhores esforços nele

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





CAUTION

O melhor software é aquele que você **não**
precisa desenvolver

E-Commerce

- Catálogo de produtos (Core)
- Carrinho de compras (Core)
- Checkout (Core)
- Avaliação dos produtos (Suporte)
- Produtos favoritos (Suporte)
- Processamento do pagamento (Genérico)
- Emissão de nota fiscal (Genérico)
- Gestão do estoque (Genérico)

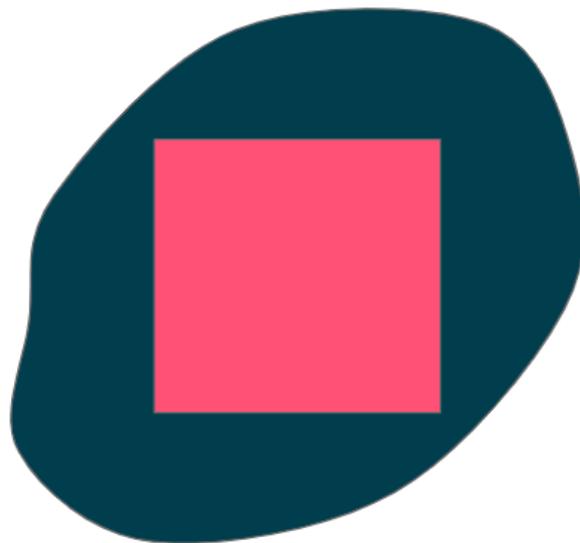
Gateway de Pagamento

- Gestão da recorrência (Core)
- Integração com os adquirentes e bancos (Core)
- Análise antifraude (Genérico)
- Réguas de cobrança para avisar do vencimento dos boletos (Suporte)

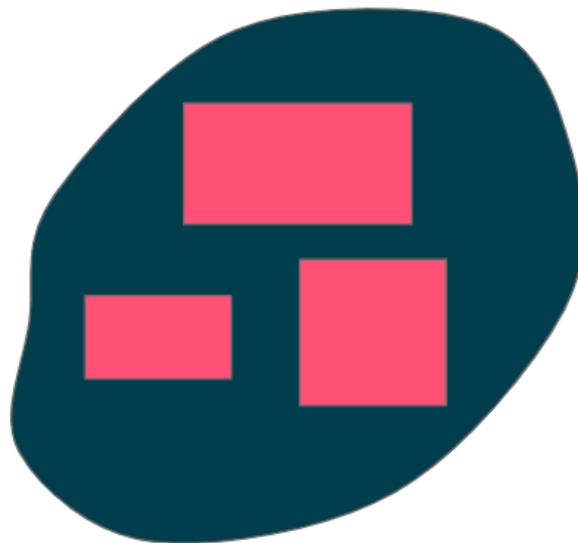
Gestão de financiamento imobiliário

- Cálculo e provisionamento das parcelas (Core)
- Análise de documentos (Core)
- Simulação do financiamento pelo site (Suporte)
- Assinatura digital dos contratos (Genérico)
- Processamento do pagamento (Genérico)

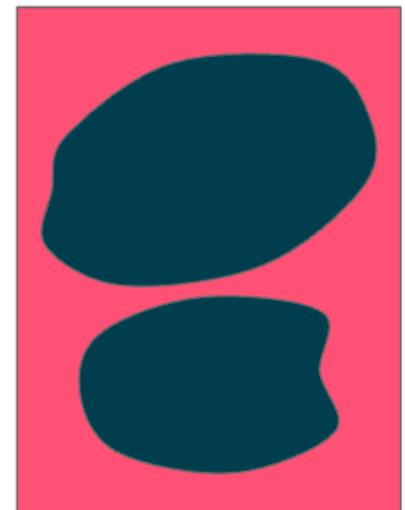
Which decomposition is right?



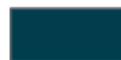
A



B



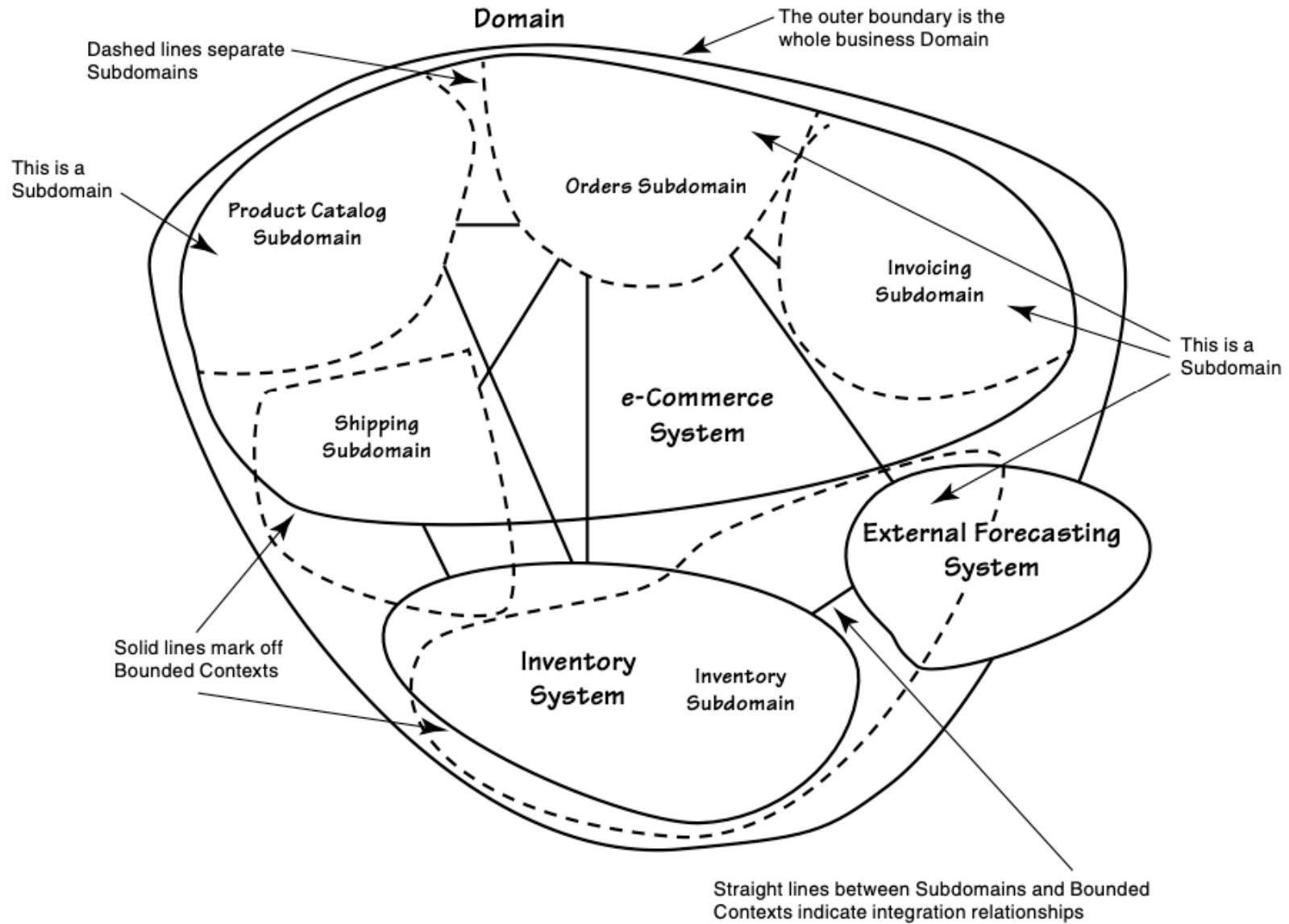
C



Subdomain



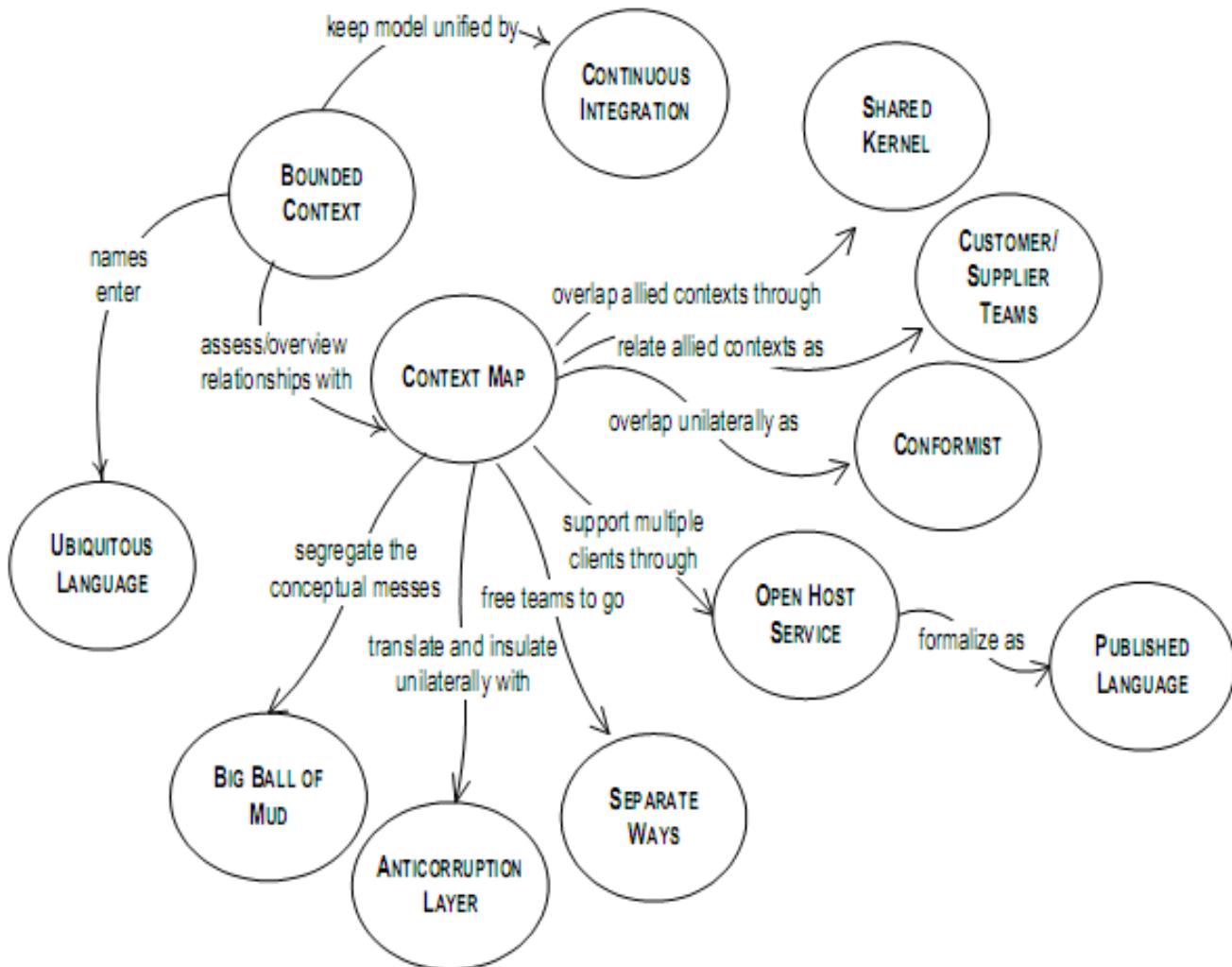
Bounded Context



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

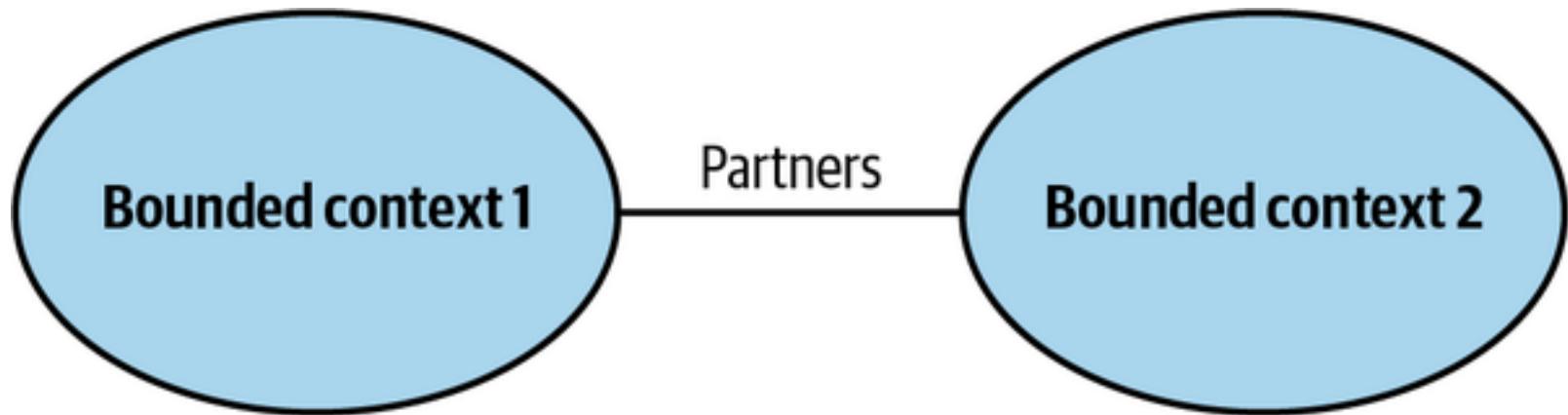


A forma de interação entre cada bounded context dá origem ao **context map**



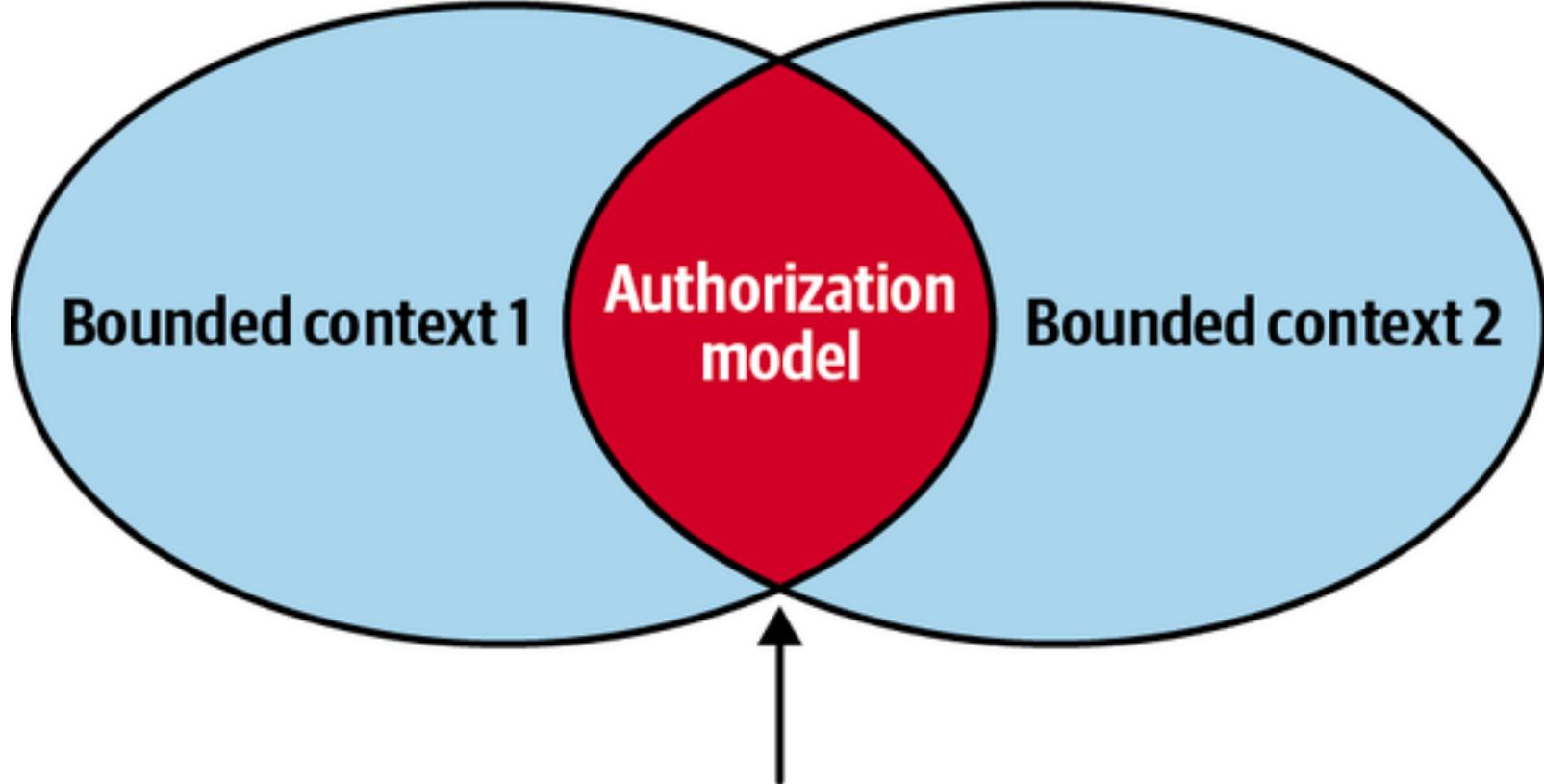
Integration Patterns

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



Partnership

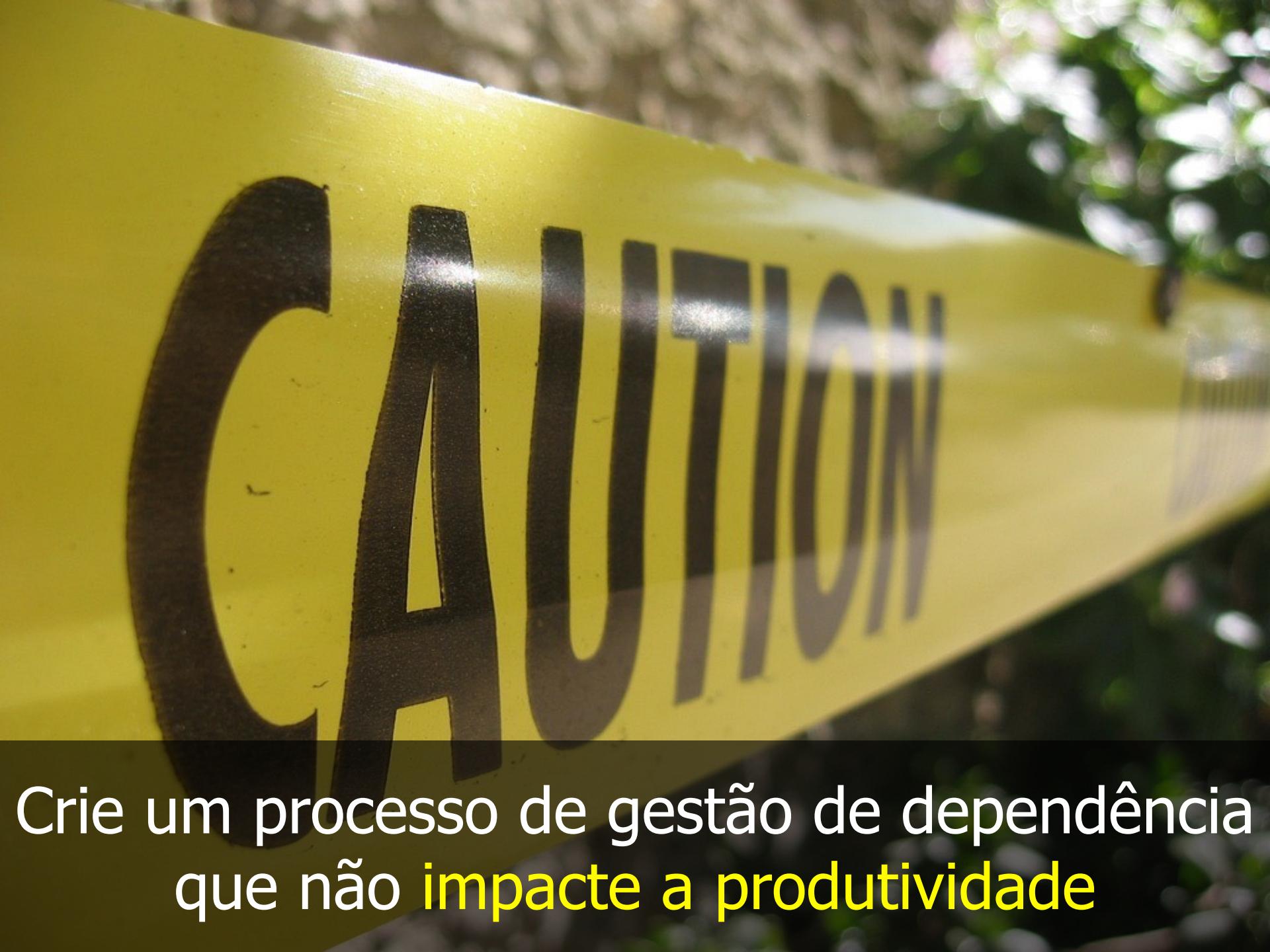
Duas ou mais equipes podem **trabalhar de forma sincronizada** numa entrega que envolve dois ou mais bounded contexts



Shared Kernel

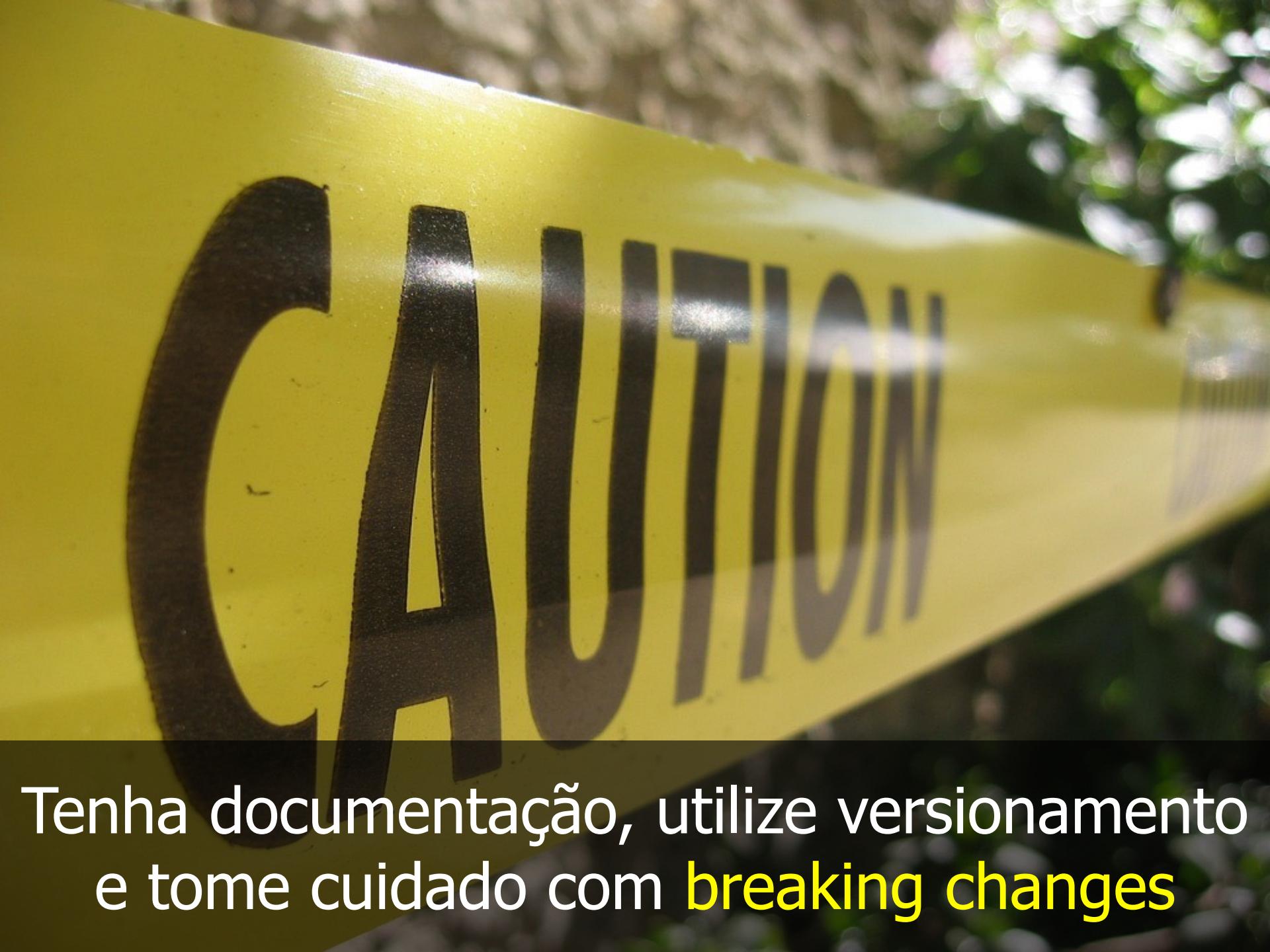
É relativamente normal compartilhar parte do código comum entre vários bounded contexts, principalmente por propósitos não relacionados diretamente ao negócio mas por infraestrutura

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



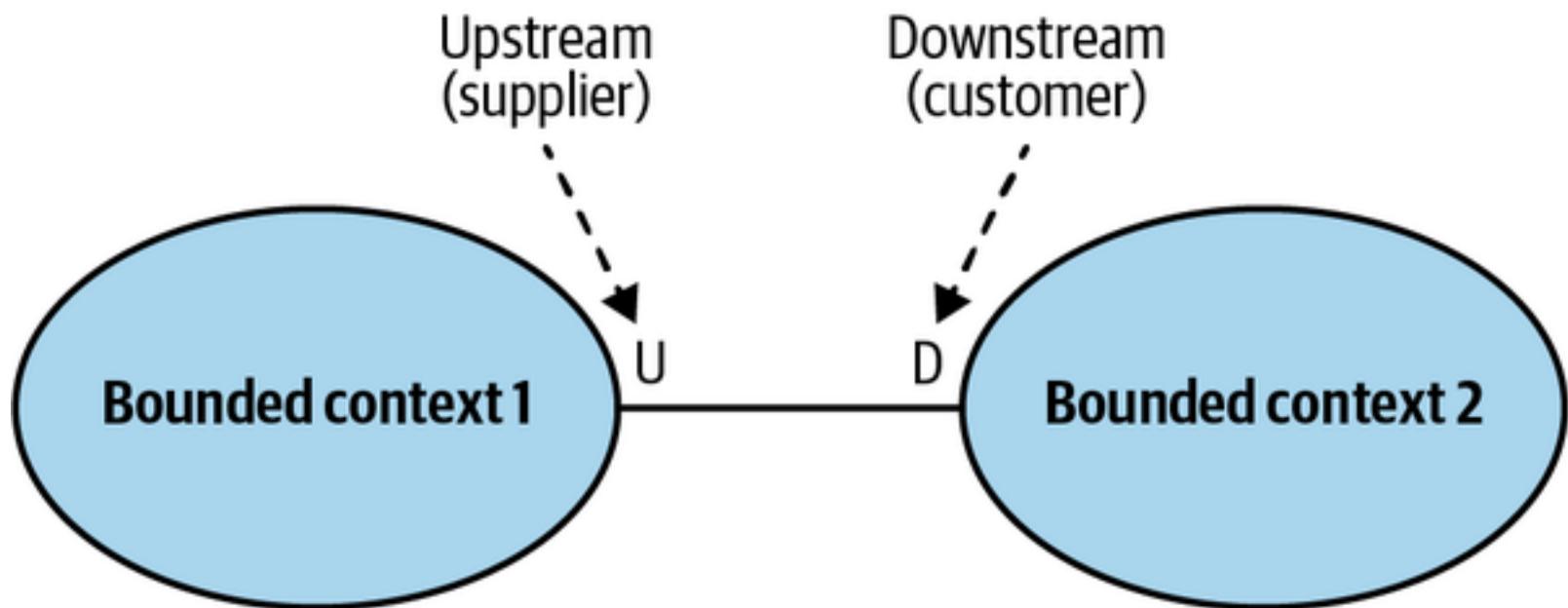
CAUTION

Crie um processo de gestão de dependência
que não impacte a produtividade



CAUTION

Tenha documentação, utilize versionamento
e tome cuidado com **breaking changes**

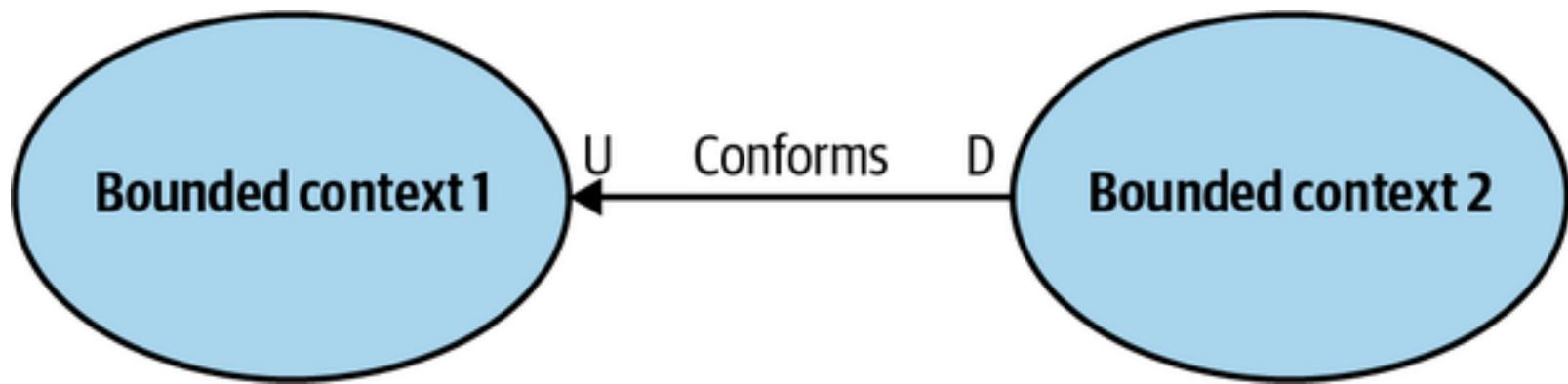


Customer/Supplier

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

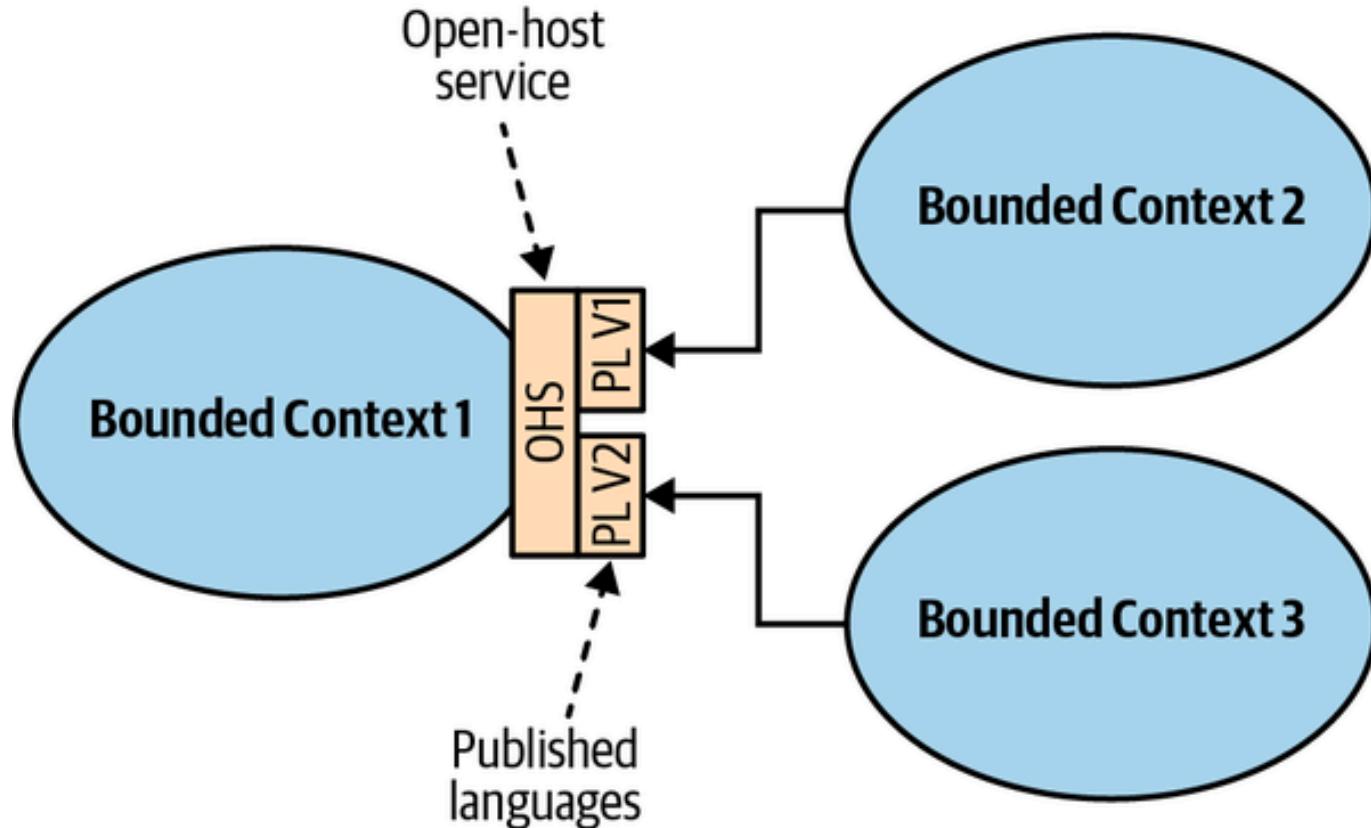
A close-up photograph of a yellow caution tape. The word "CAUTION" is printed in large, bold, black capital letters. The tape is slightly curved, and the background shows some green foliage.

Pode ser interessante ter testes de contrato
que garantam o funcionamento da API



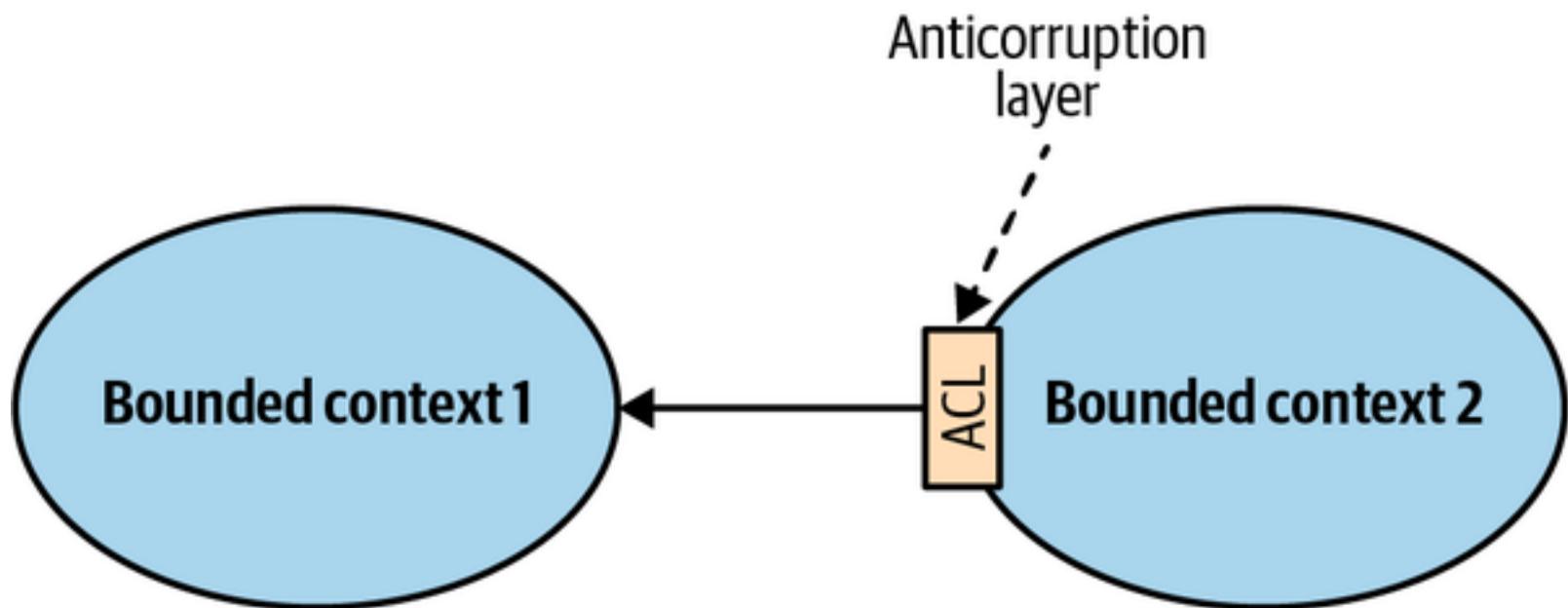
Conformist

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

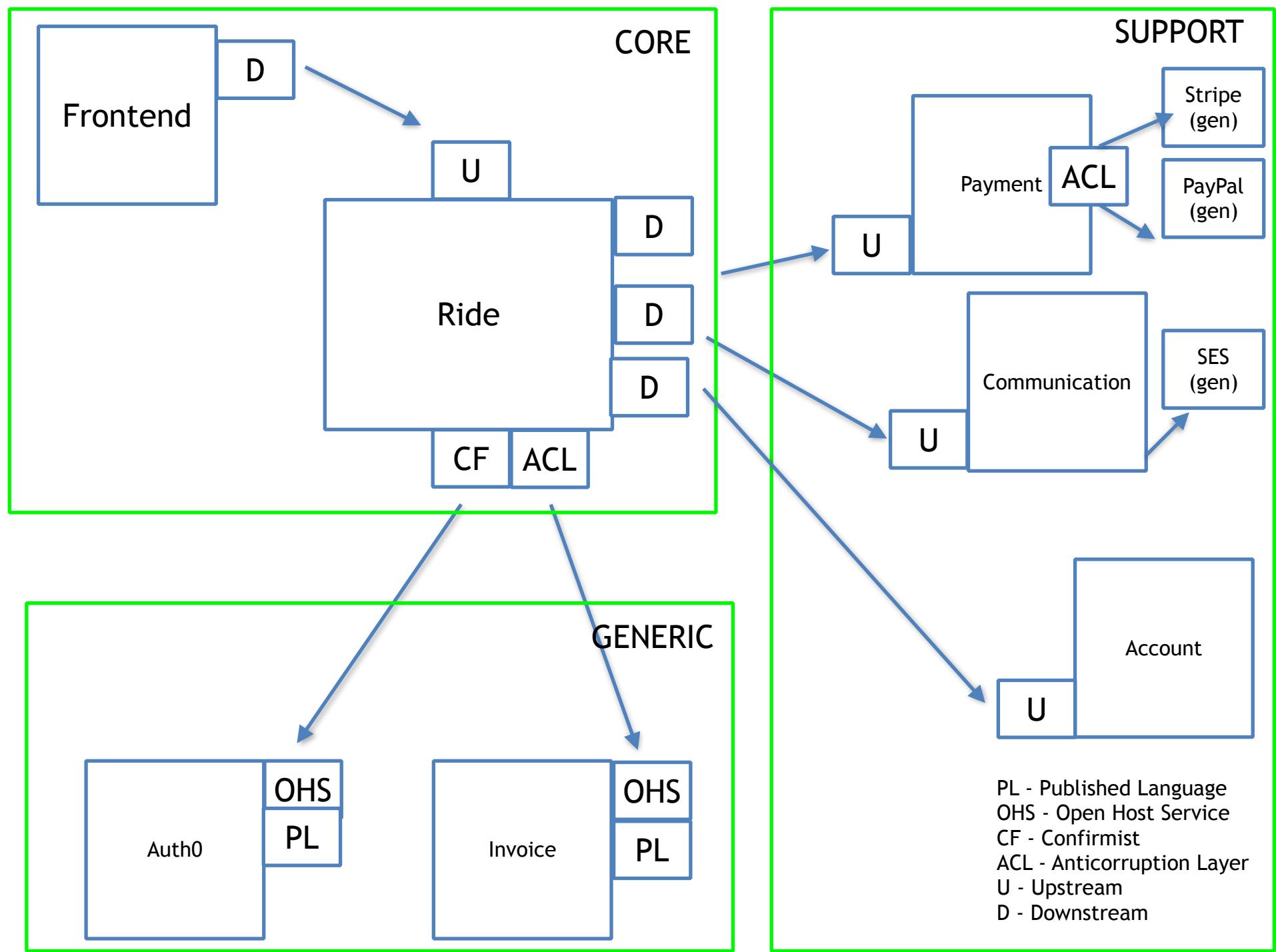
Um bounded context pode disponibilizar um conjunto de serviços utilizando um protocolo padrão e com uma documentação abrangente para quem tiver interesse em integrar



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





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 (por conta de uma base de código menor)

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

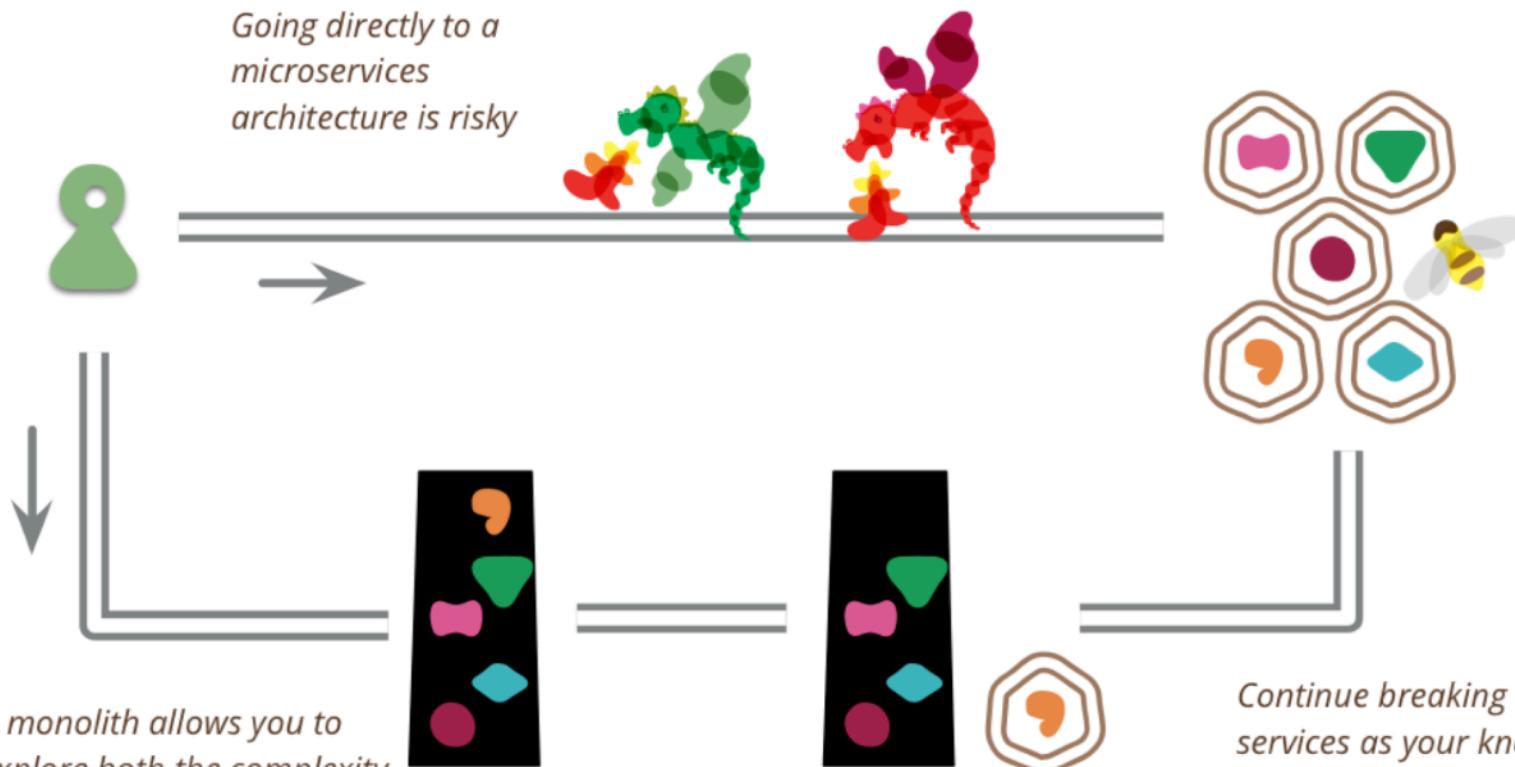


Martin Fowler

3 June 2015

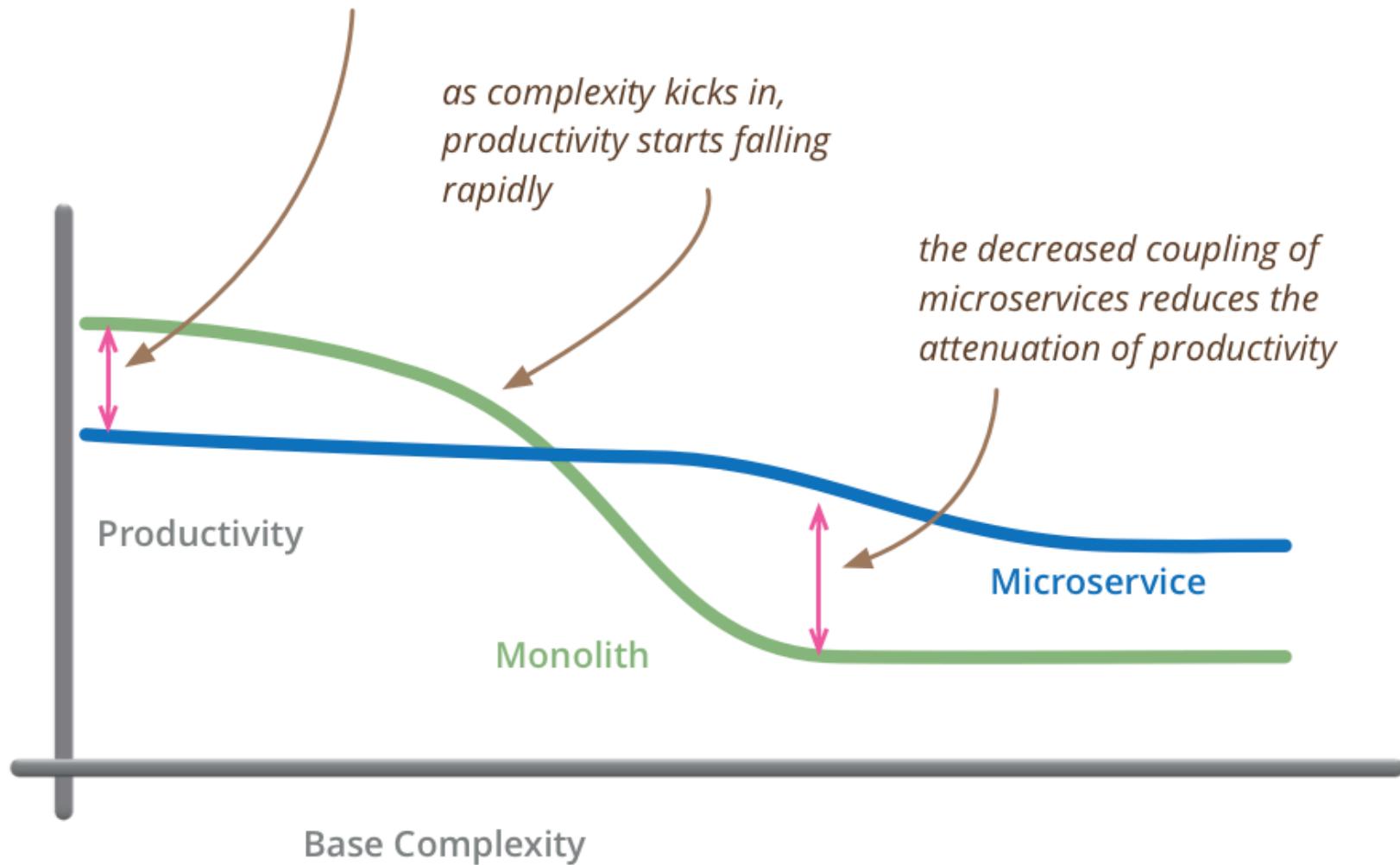
As I hear stories about teams using a [microservices architecture](#), I've noticed a common pattern.

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.

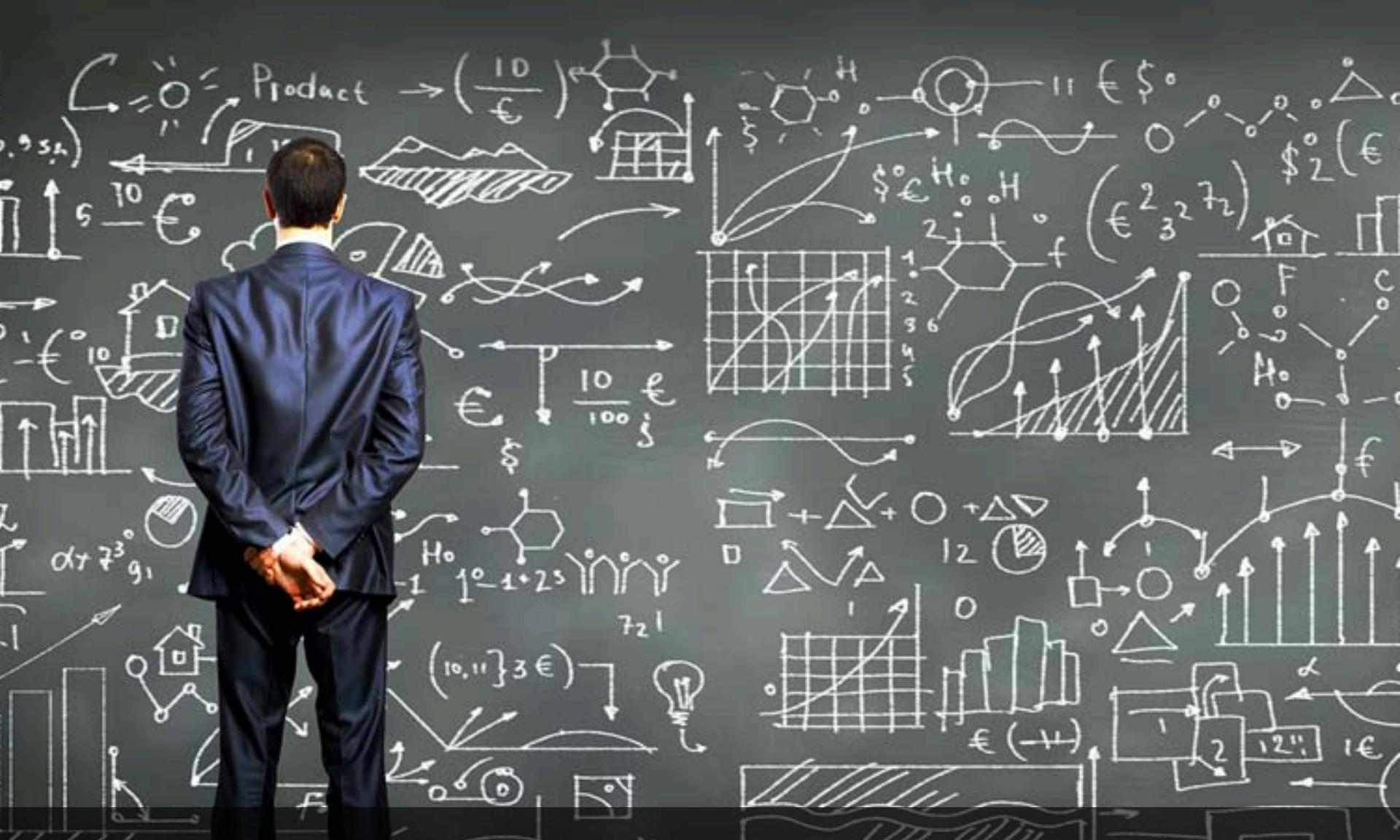


Continue breaking out services as your knowledge of boundaries and service management increases

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 os bounded contexts