



# Event-Driven Architecture



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



Muitas vezes somos levados a acreditar  
que microservices são sempre perfeitos...

A close-up photograph of a yellow caution tape. The word "CAUTION" is printed in large, bold, black capital letters. The tape is positioned diagonally across the frame, with a blurred background of green foliage and trees.

CAUTION

Cuidado, somente separar os serviços  
está longe de ser o único objetivo



A pior coisa que podemos ter é uma  
**arquitetura monolítica distribuída**



Alto acoplamento entre os serviços



Muitos pontos de falha

A close-up photograph of a house of cards built from a standard deck of playing cards. The cards are arranged in a triangular pattern, with each card's suit and value visible. The house is partially collapsed, with several cards leaning or fallen over, particularly in the upper and middle sections. The background is dark, making the white cards stand out.

Falta de resiliência



E se o serviço receber um grande volume  
de requisições de uma hora pra outra?



Baixa escalabilidade

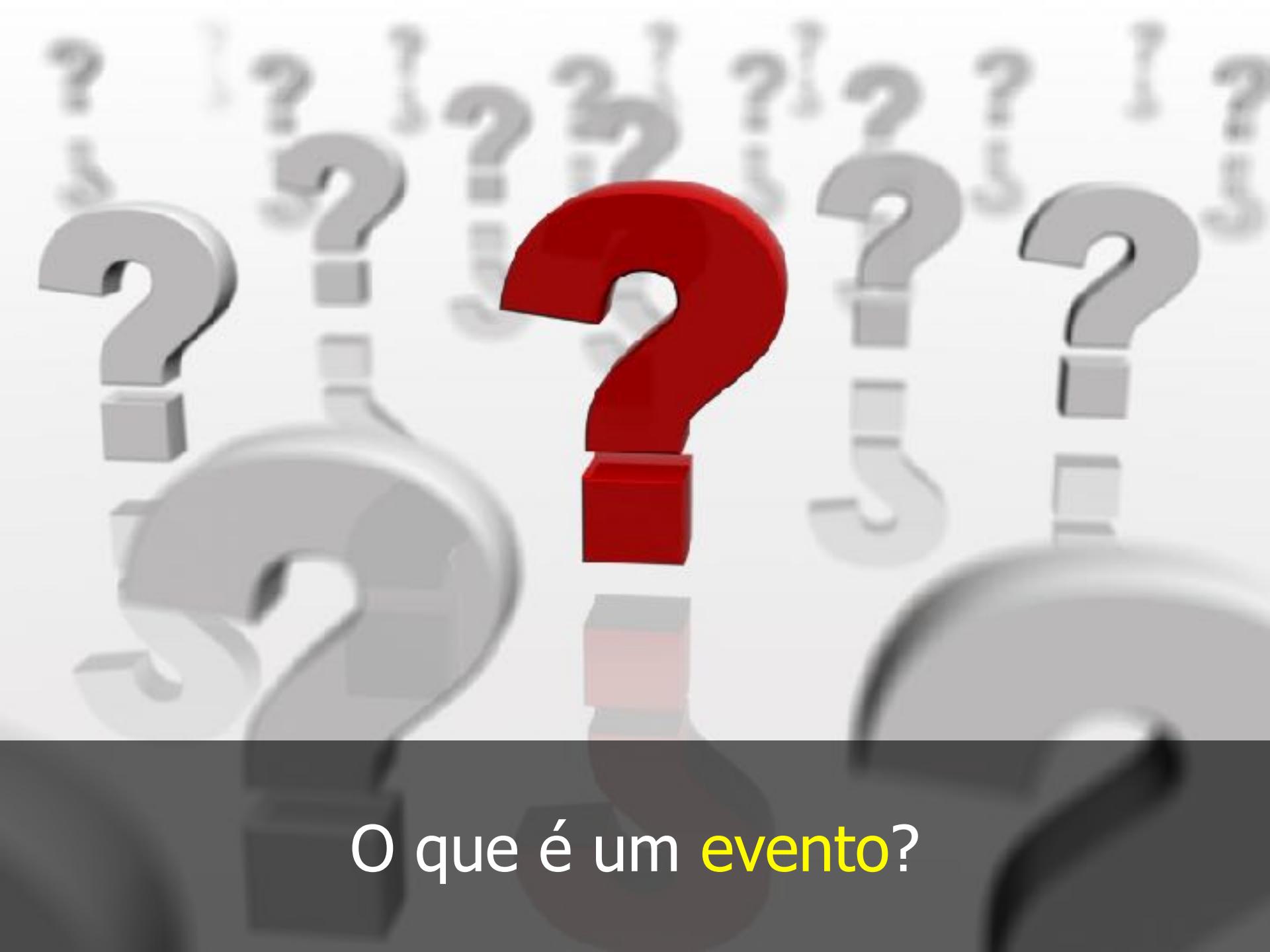


Possível desperdício de infraestrutura



Qual é a solução?

Uma **arquitetura orientada a eventos**, ou Event-Driven Architecture, utiliza eventos para disparar e se comunicar com serviços desacoplados e é cada vez mais comum, principalmente em um ambiente de microservices

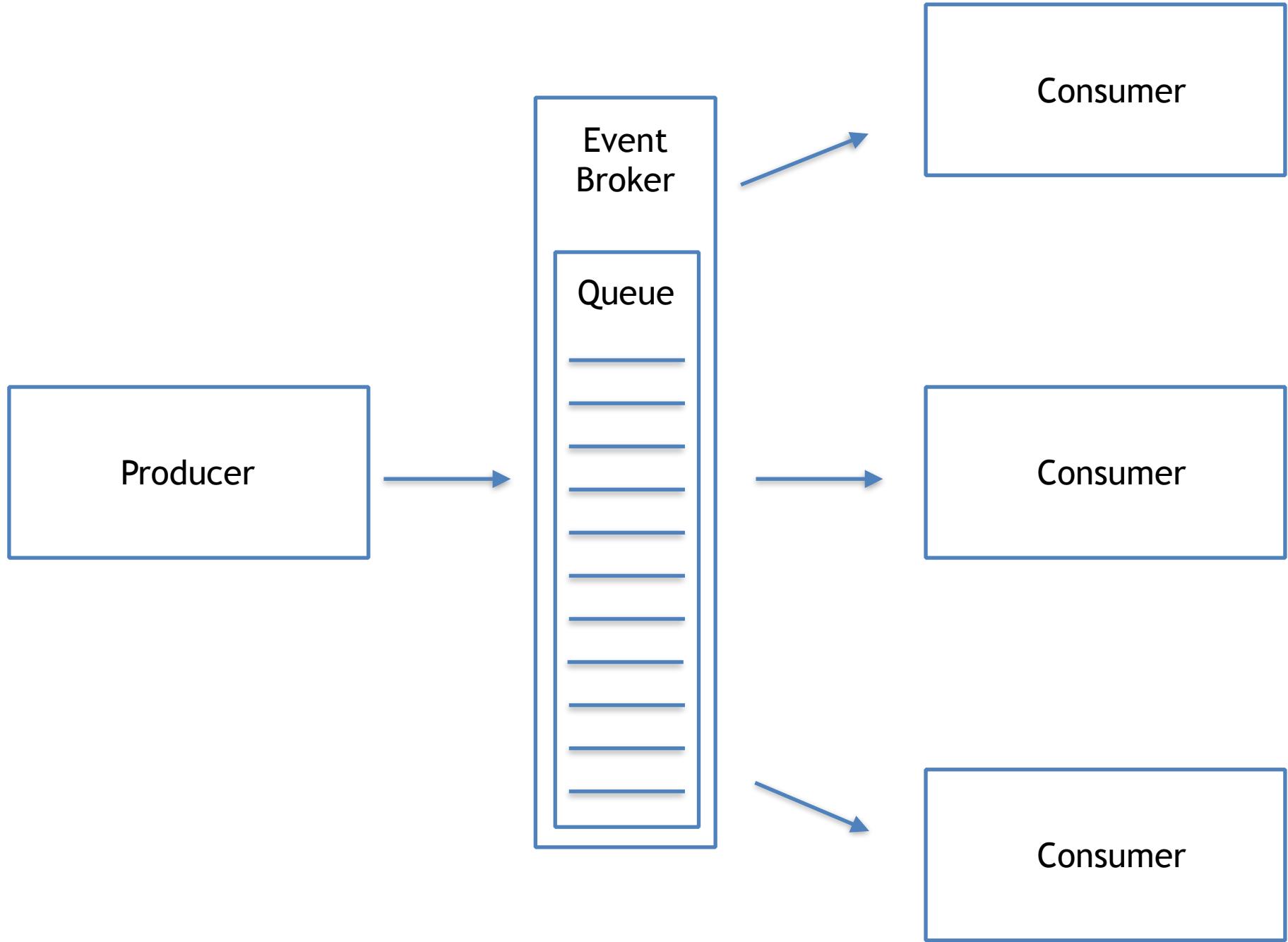


O que é um **evento**?

Os eventos são **fatos** que aconteceram no domínio e que podem ser um gatilho para a execução de regras de negócio



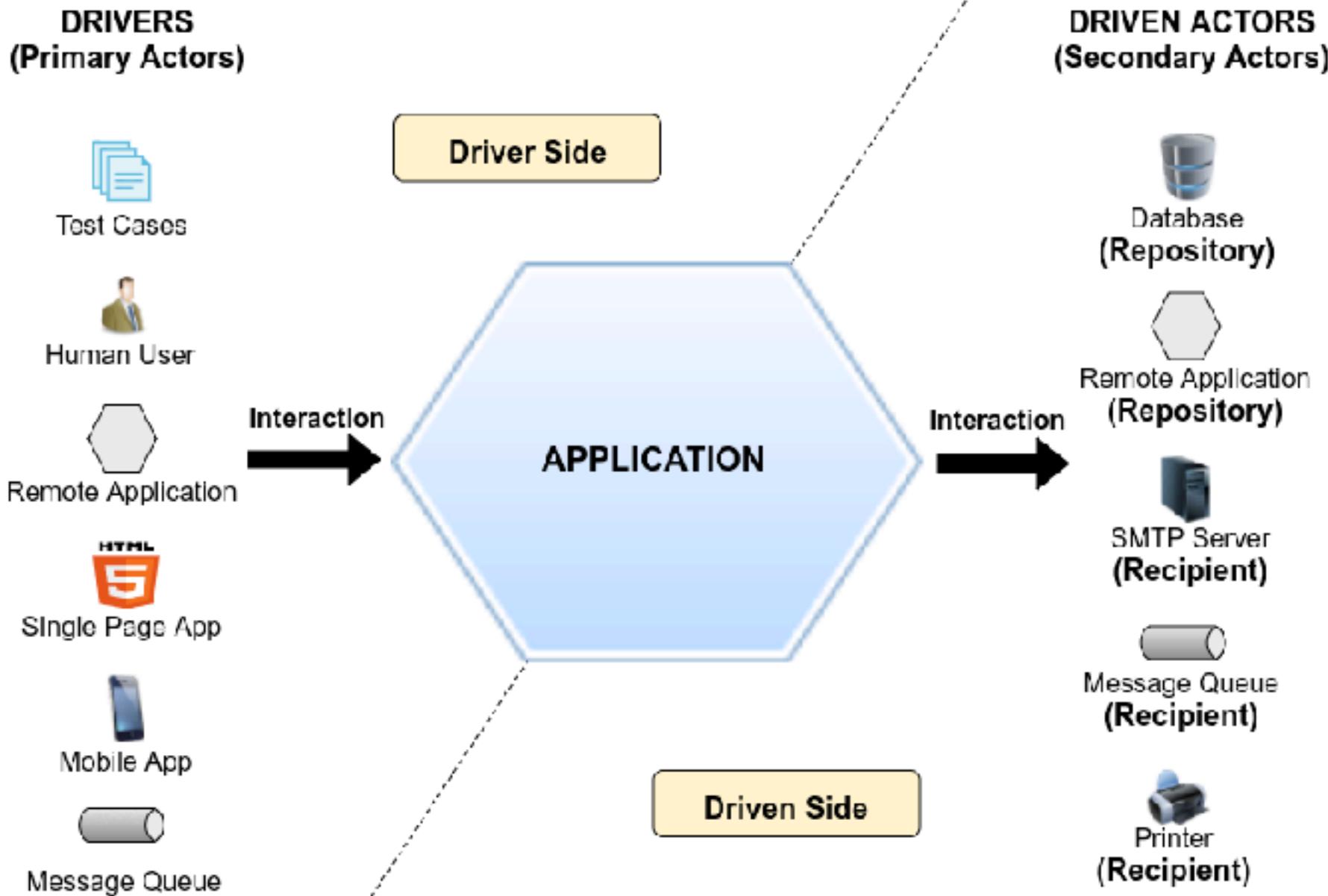
Eventos tem **baixo acoplamento**, são intercambiáveis e assíncronos

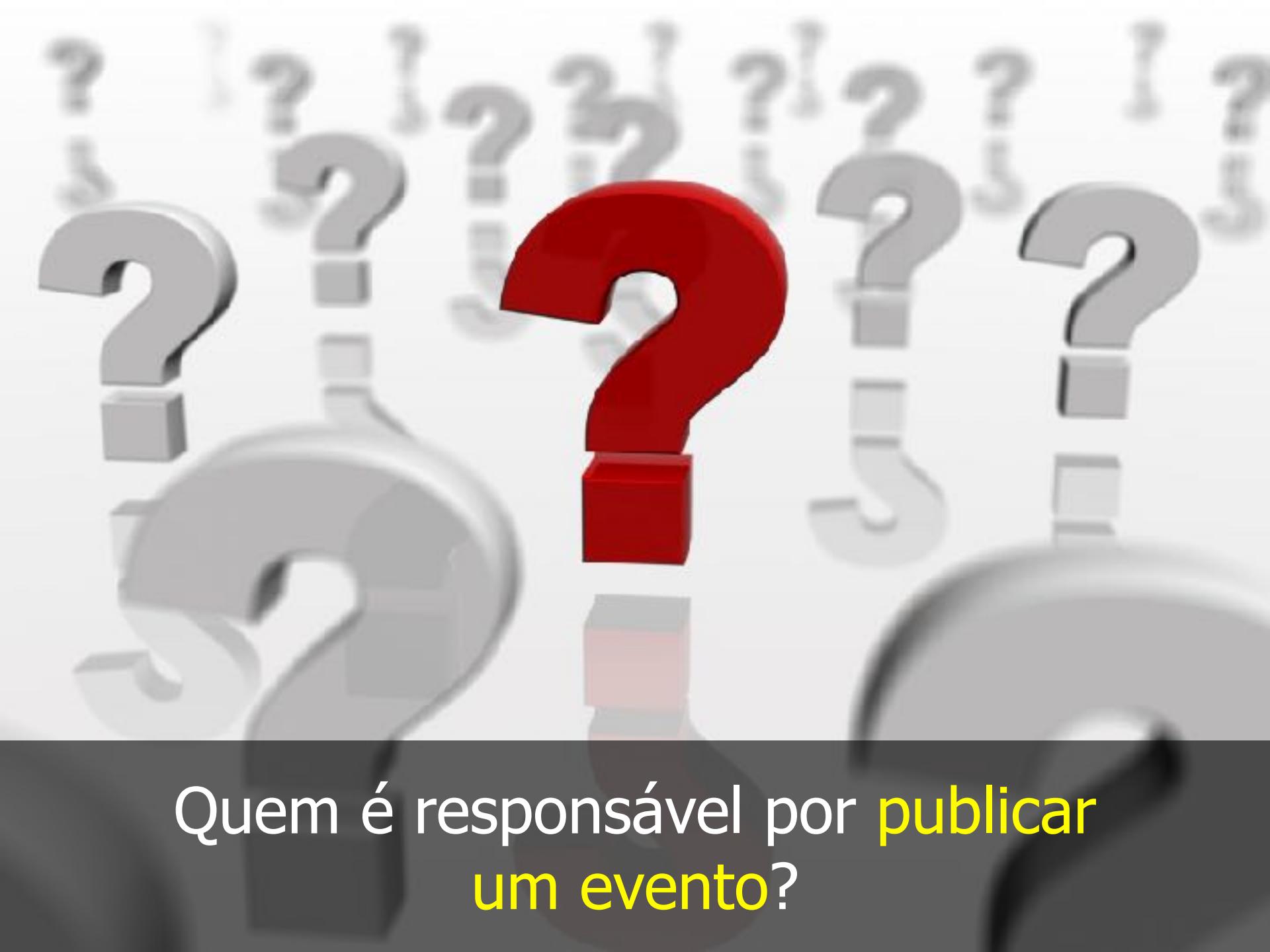


É a base para **desacoplar** aggregates  
dentro e fora de um bounded context

# Exemplos

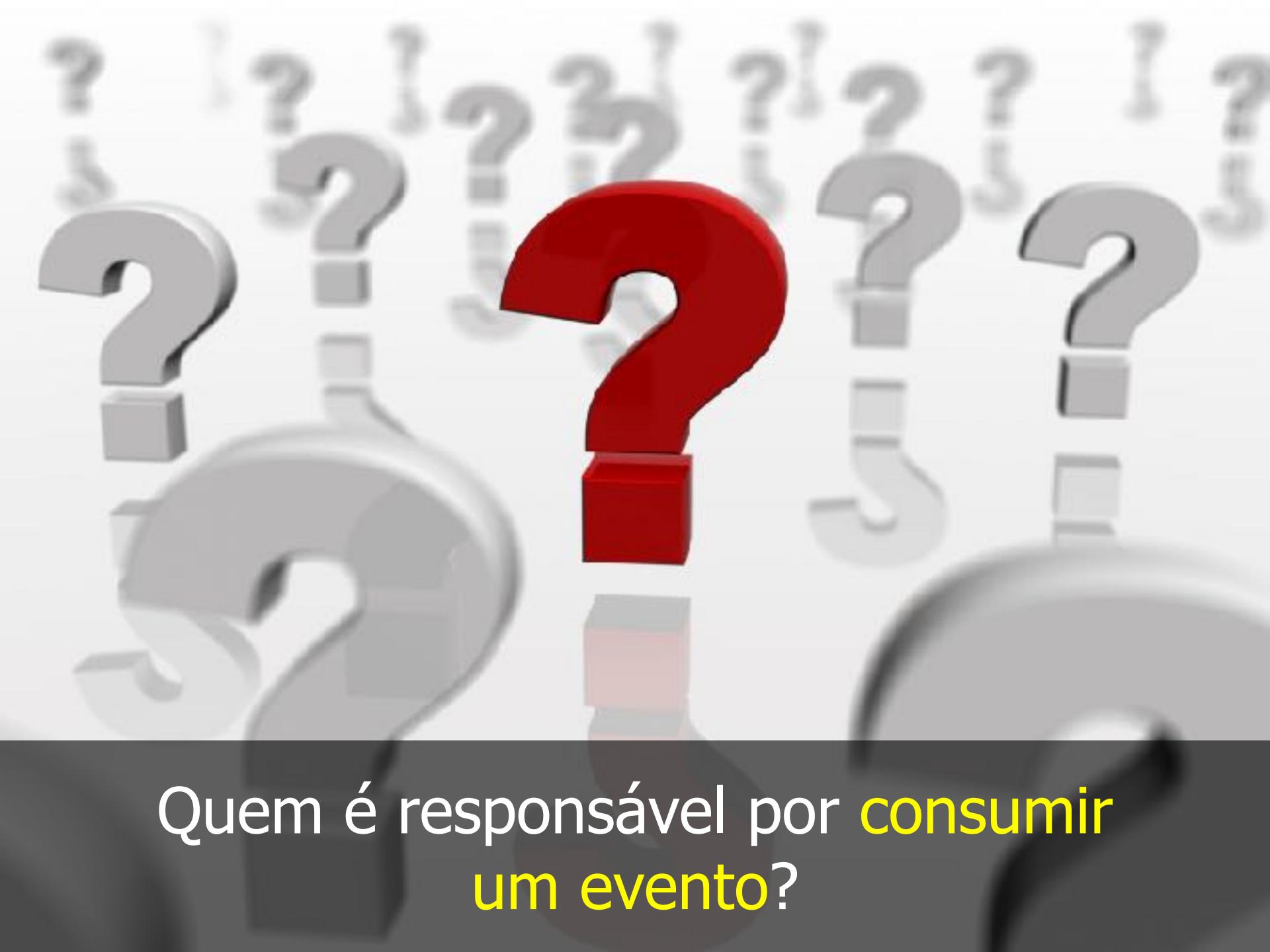
- Compra realizada (OrderPlaced)
- Pagamento aprovado (PaymentApproved)
- Nota fiscal emitida (InvoiceReady)
- Entrega confirmada (DeliveryConfirmed)
- Devolução solicitada (RefundRequested)
- Produto avaliado (ProductReviewed)





Quem é responsável por **publicar**  
**um evento?**

A publicação de um evento é uma ação realizada por um use case ou application service, que interage com uma fila por meio de uma **porta de saída**



Quem é responsável por **consumir**  
**um evento?**

Os use cases ou application services agem como handlers, e são os responsáveis por consumir os eventos que chegam por meio de uma porta de entrada



Utilizar eventos significa ter **microservices**?

Os eventos podem ser publicados e consumidos dentro da mesma aplicação, ainda que seja monolítica

Adotar uma arquitetura orientada a eventos tem os seguintes benefícios:

- Baixo acoplamento entre os serviços
- Tolerância a falha
- Melhor controle sobre o débito técnico
- Disponibilidade/Escalabilidade mais alta
- Menos custos com infraestrutura
- Melhor entendimento sobre o que aconteceu, inclusive com a possibilidade de PITR
- Diversidade tecnológica

Adotar uma arquitetura orientada a eventos tem os seguintes desafios:

- Complexidade técnica mais alta
- Duplicação de eventos
- Falta de clareza no workflow
- Transações distribuídas
- Dificuldade em tratar e diagnosticar erros
- Diversidade tecnológica



Qual é a diferença entre comando e evento?

Os comandos representam a intenção de um determinado usuário, podendo ser aceitos ou rejeitados dependendo da situação

# Comando

Os nomes dos comandos são **sempre no imperativo**

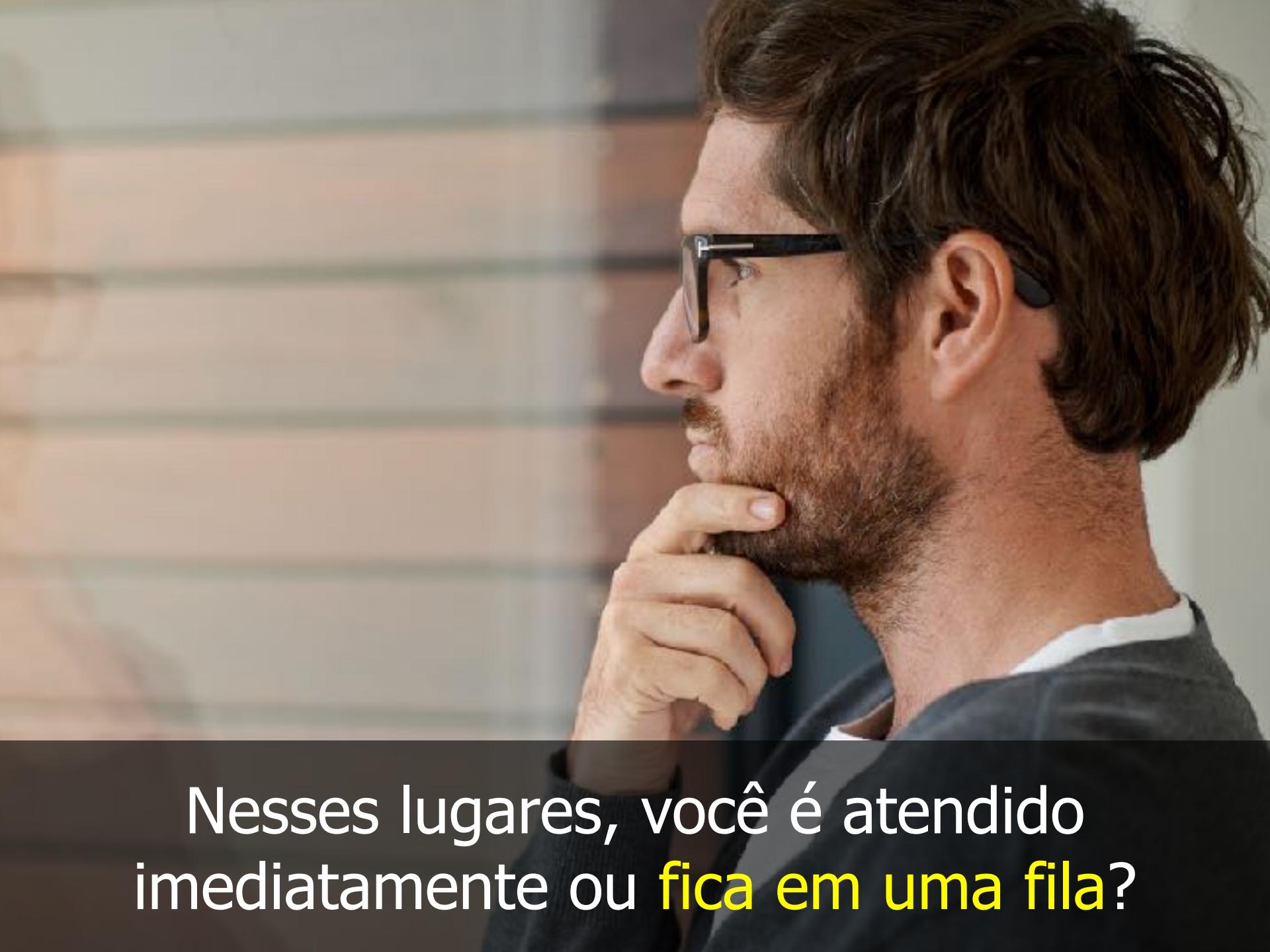
- PlaceOrder
- PayInvoice
- GenerateReport
- EnrollStudent
- UpdateCustomer
- UploadFile

# Similar ao padrão Command

- Promove o desacoplamento entre quem quer executar e quem é executado
- Um objeto que contém todos os dados necessário para abstrair uma determinada execução
- Pode ser executado de forma síncrona ou assíncrona
- É possível armazená-lo para executar futuramente
- Se for armazenado ele pode servir de base para refazer ou desfazer uma execução



O uso de **filas** é obrigatório?



Nesses lugares, você é atendido  
imediatamente ou **fica em uma fila?**



Restaurante



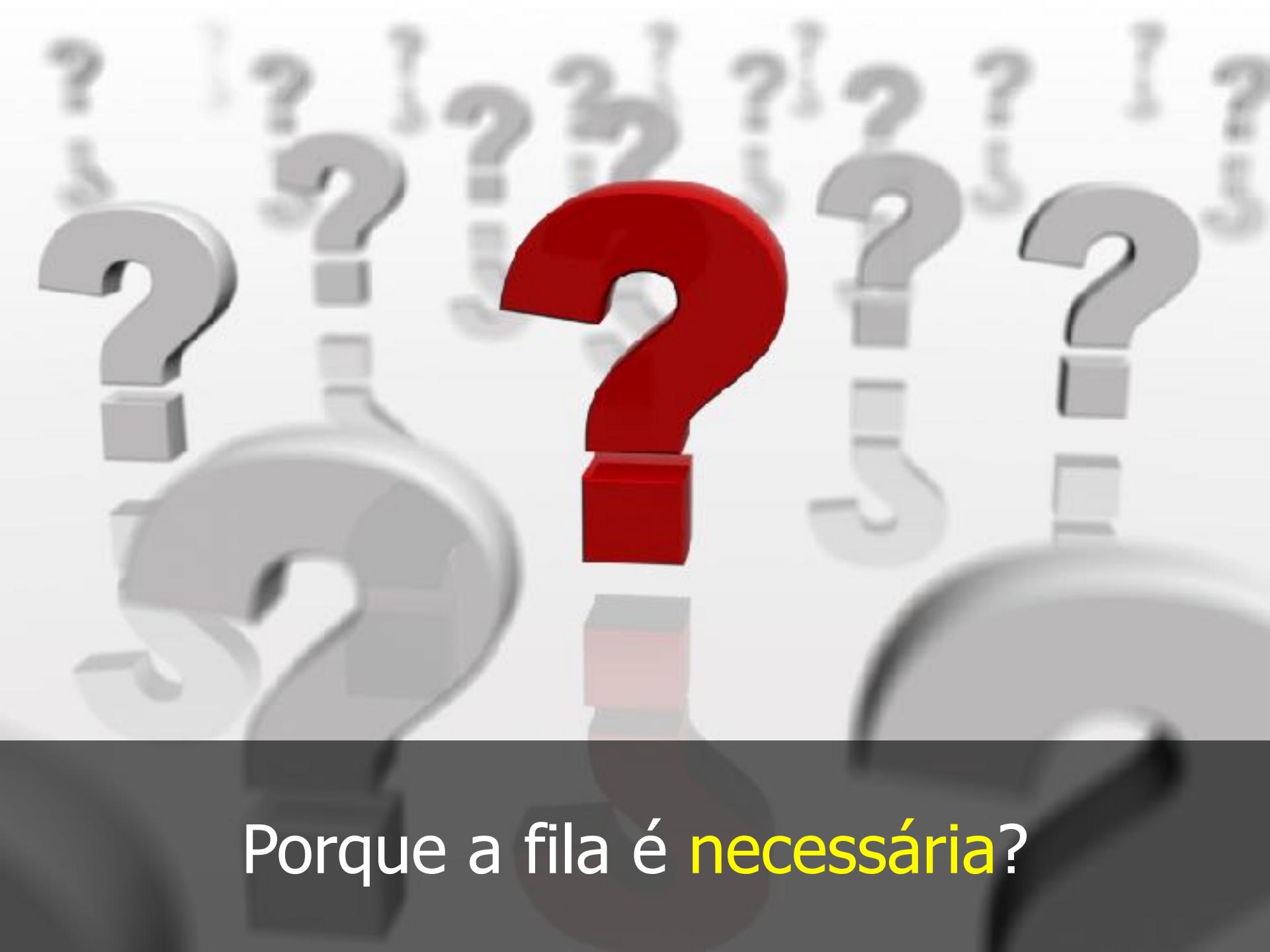
Cabelereiro



Médico



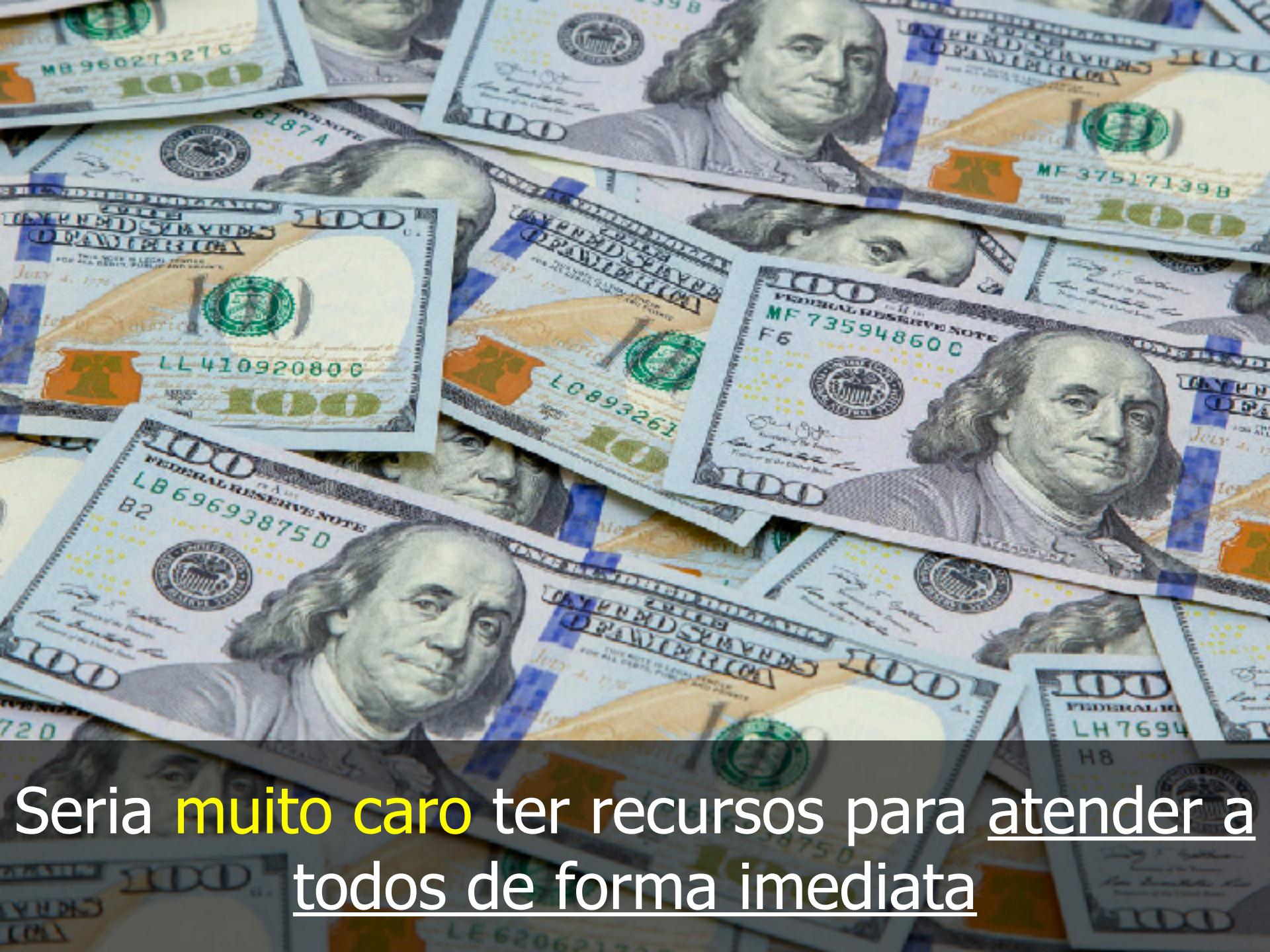
Pedir cancelamento de qualquer tipo  
de serviço por assinatura



Porque a fila é necessária?



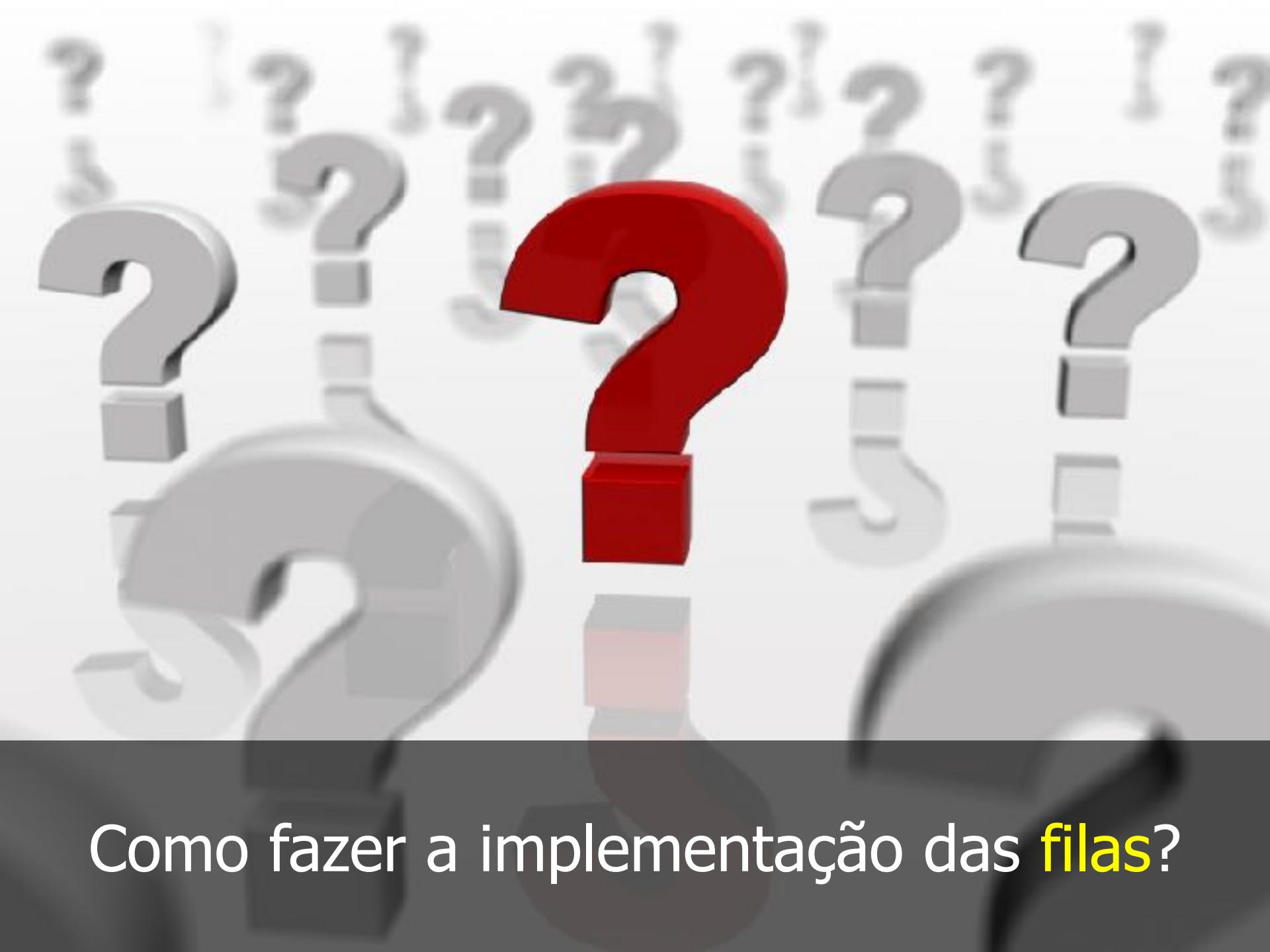
Não existem recursos suficientes disponíveis



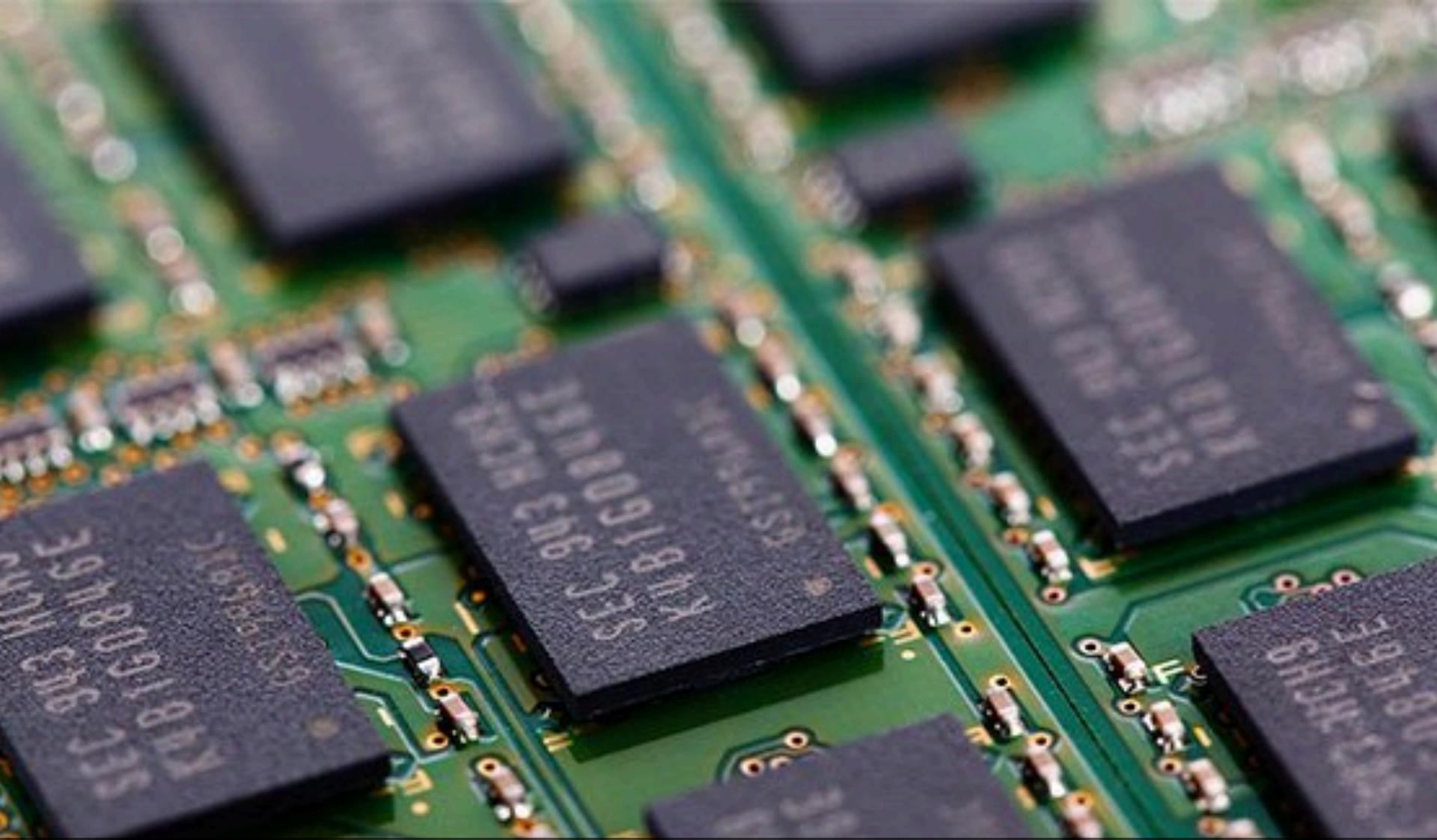
Seria **muito caro** ter recursos para atender a todos de forma imediata



Em diversos momentos, por conta da  
ociosidade, eles seriam desperdiçados



Como fazer a implementação das **filas**?



Localmente por meio de um **intermediário** que  
implementa um mecanismo de notificação



Os algoritmos geralmente são baseados  
nos padrões **Observer** e **Mediator**



Pela rede por meio de uma plataforma de  
mensageria

# Alguns tipos de plataformas de mensageria

- RabbitMQ
- Kafka
- AWS SQS
- ActiveMQ
- Google Pub/Sub
- ZeroMQ
- Pulsar



É obrigatório o uso de uma plataforma de  
**mensageria?**

Mensageria tem relação com **resiliência**, entre microservices com absoluta certeza é necessário mas dentro da mesma aplicação aumenta a complexidade técnica, uma questão que fica é: uma chamada a um método é resiliente?