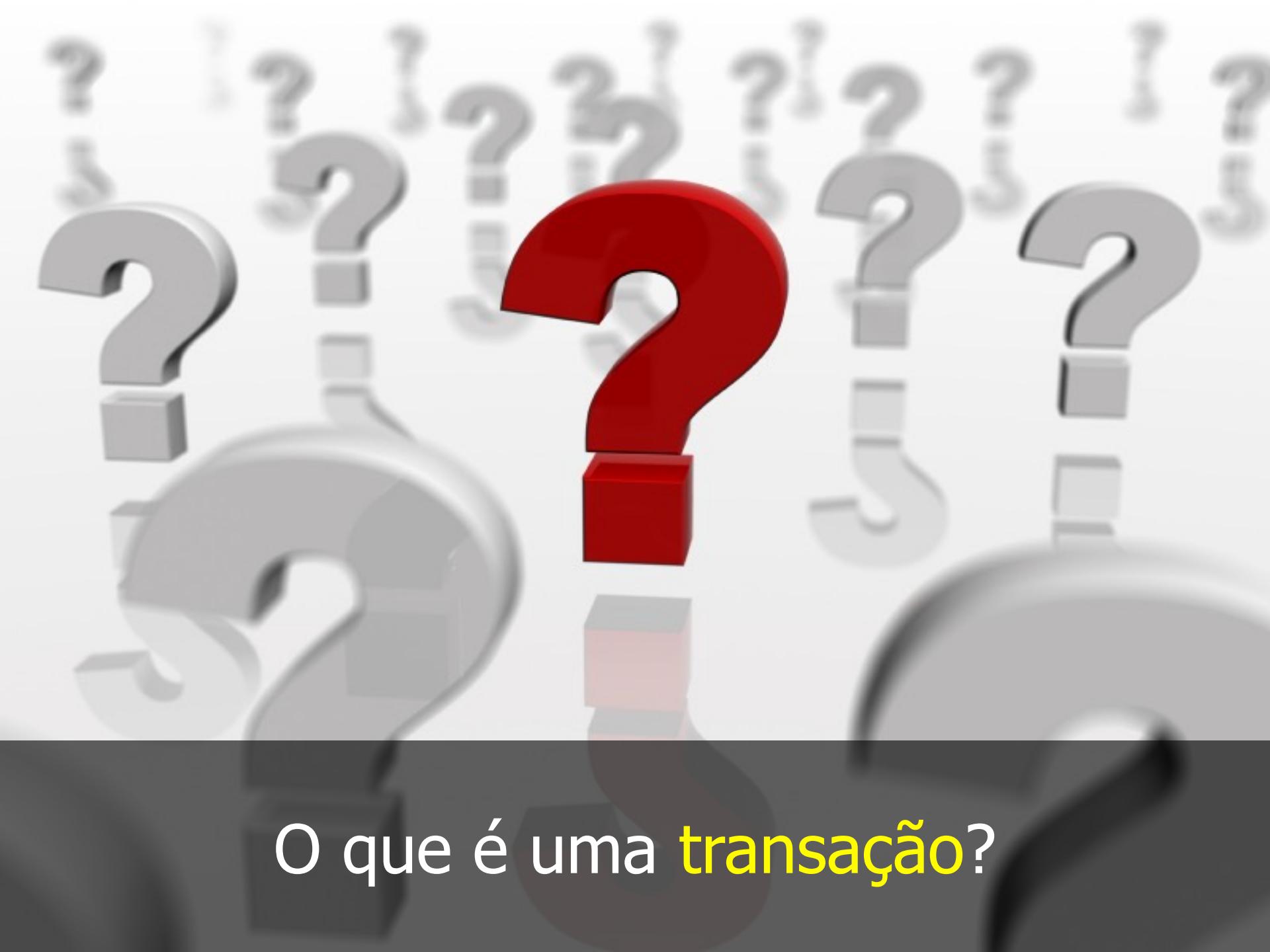


Event-Driven Architecture





O que é uma transação?

Transação é a abstração de um conjunto de operações que devem ser tratada como uma única unidade lógica, onde para ter sucesso, todas as suas operações devem ser bem sucedidas ou serem desfeitas

Uma forma comum de pensar em uma transação é pelo conceito de **ACID** ou Atomicity, Consistency, Isolation e Durability, relacionado a comandos executados em um banco de dados relacional

ACID

Atomicity: Todos os comandos da transação são tratados como uma unidade, todos executam ou nenhum executa

Consistency: Respeitam as regras de consistência estabelecidas no modelo de dados por meio de constraints, cascades, triggers e a sua combinação

Isolation: Faz com que a transação não tenha interferência de outros comandos que estejam sendo executados em paralelo por outras transações

Durability: Garante que uma vez comitada o resultado da transação seja persistido de forma definitiva

Por exemplo, imagine uma transação que faz uma transferência de fundos entre duas contas bancárias:

```
begin
```

```
insert into bank.transaction (id, type, amount) values (1, 'debit', 100);
```

```
insert into bank.transaction (id, type, amount) values (2, 'credit', 100);
```

```
commit
```



O que acontece se o **primeiro insert** tiver
sucesso e o **segundo falhar**?



CAUTION

Nem todas as operações de uma transação
são realizadas dentro do banco de dados

Upload de um vídeo em um canal do YouTube

- Verificação de direitos autorais
- Conversão para diferentes formatos
- Transcrição das legendas
- Notificação dos inscritos
- Atualização do algoritmo de busca
- Atualização dos algoritmos de recomendação



O que acontece se **faltar memória** na hora
de converter o vídeo?

Compra de um produto em uma e-commerce

- Processamento do pagamento
- Emissão da nota fiscal
- Expedição do estoque
- Solicitação de coleta
- Crédito de pontos de fidelização
- Geração de cupom de desconto para a próxima compra



O que acontece se a nota fiscal não puder ser emitida por o **certificado digital expirou**?

Finalização de uma corrida

- Cálculo da distância
- Cálculo da tarifa
- Processamento do pagamento
- Envio do comprovante da corrida
- Emissão da nota fiscal



O que acontece se o gateway de pagamento
estiver **fora do ar**?



A maior parte dos sistemas tem transações distribuídas e com muitos **pontos de falha**

Quanto mais complexa e distribuída for a arquitetura, maiores são as chances de alguma coisa dar errado e a **resiliência** é a capacidade de manter o funcionamento e se recuperar de falhas



Como lidar com transações de forma resiliente?

É possível adotar padrões como Retry, Fallback
ou até mesmo SAGA

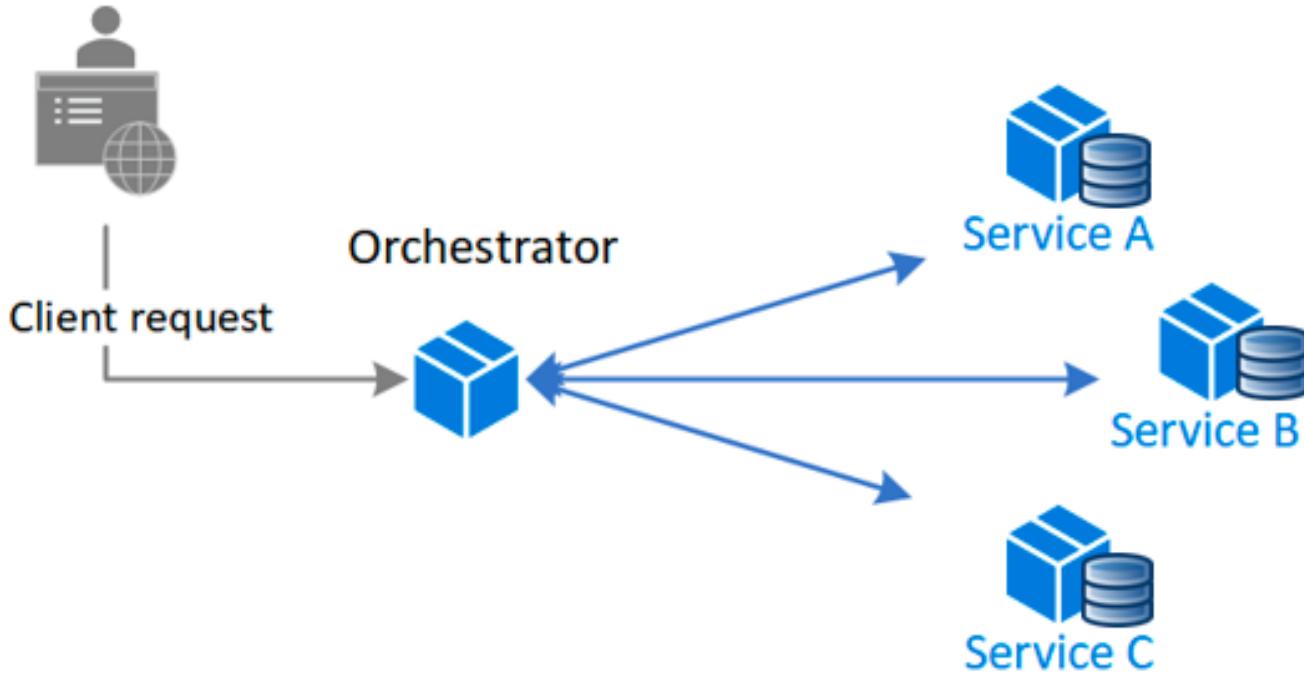
O padrão Retry simplesmente **realiza uma ou mais retentativas** em um pequeno intervalo de tempo, elas podem resolver problemas simples como perda de pacotes, oscilações na rede e até mesmo um deploy fora de hora

O padrão Fallback ao se deparar com uma indisponibilidade **faz a tentativa em outro serviço**, por exemplo, um grande e-commerce deve trabalhar com diversos adquirentes de cartão de crédito para evitar indisponibilidades e até mesmo bloqueios

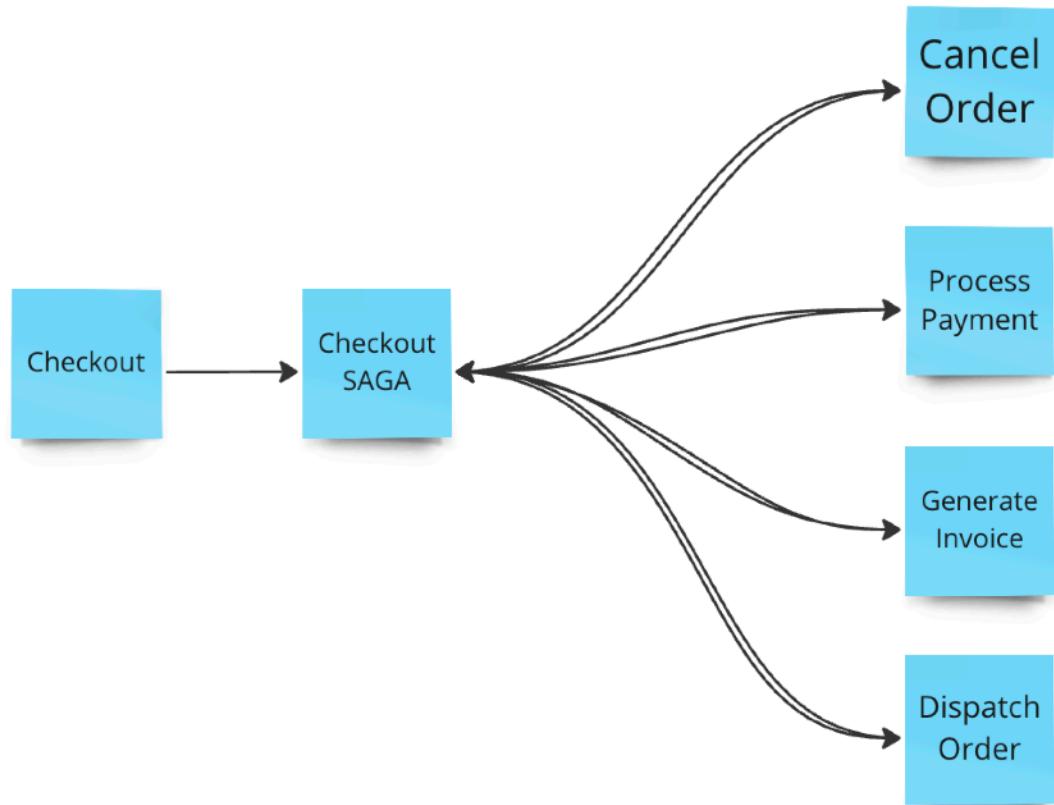
O padrão SAGA é responsável pelo gerenciamento de uma transação de longa duração por meio de uma sequência de transações locais

Tipos de transação locais

- **Pivot Transaction:** São transações go/no go, ou seja, a partir delas é decidido se o fluxo de execução segue em frente ou é abortado
- **Compensable Transaction:** São desfeitas caso a transação toda seja abortada
- **Retriable Transaction:** Tem uma garantia de execução e podem se recuperar de uma possível falha ou indisponibilidade



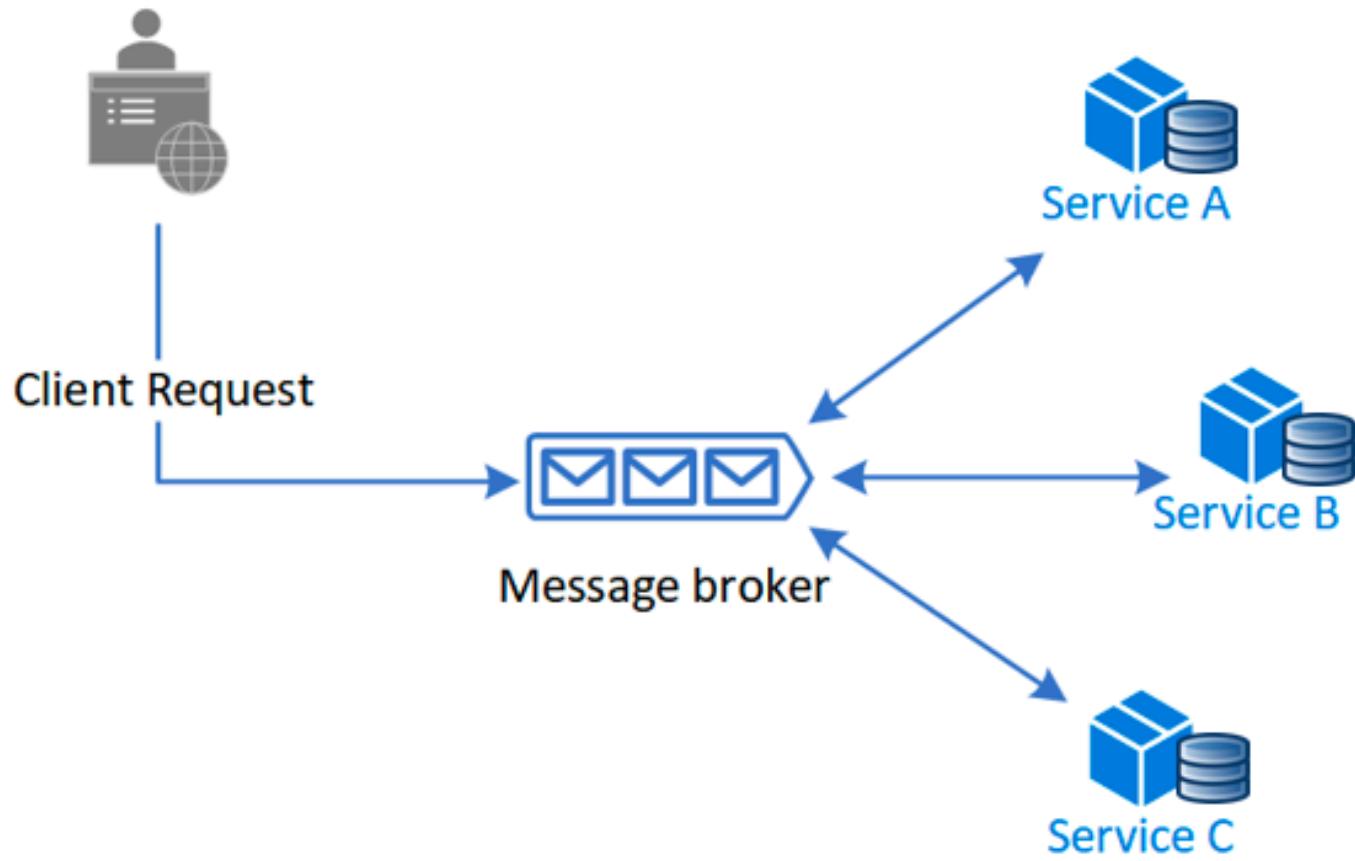
Orquestrado: existe uma lógica centralizada que faz a coordenação de cada um dos passos



Compensable Transaction

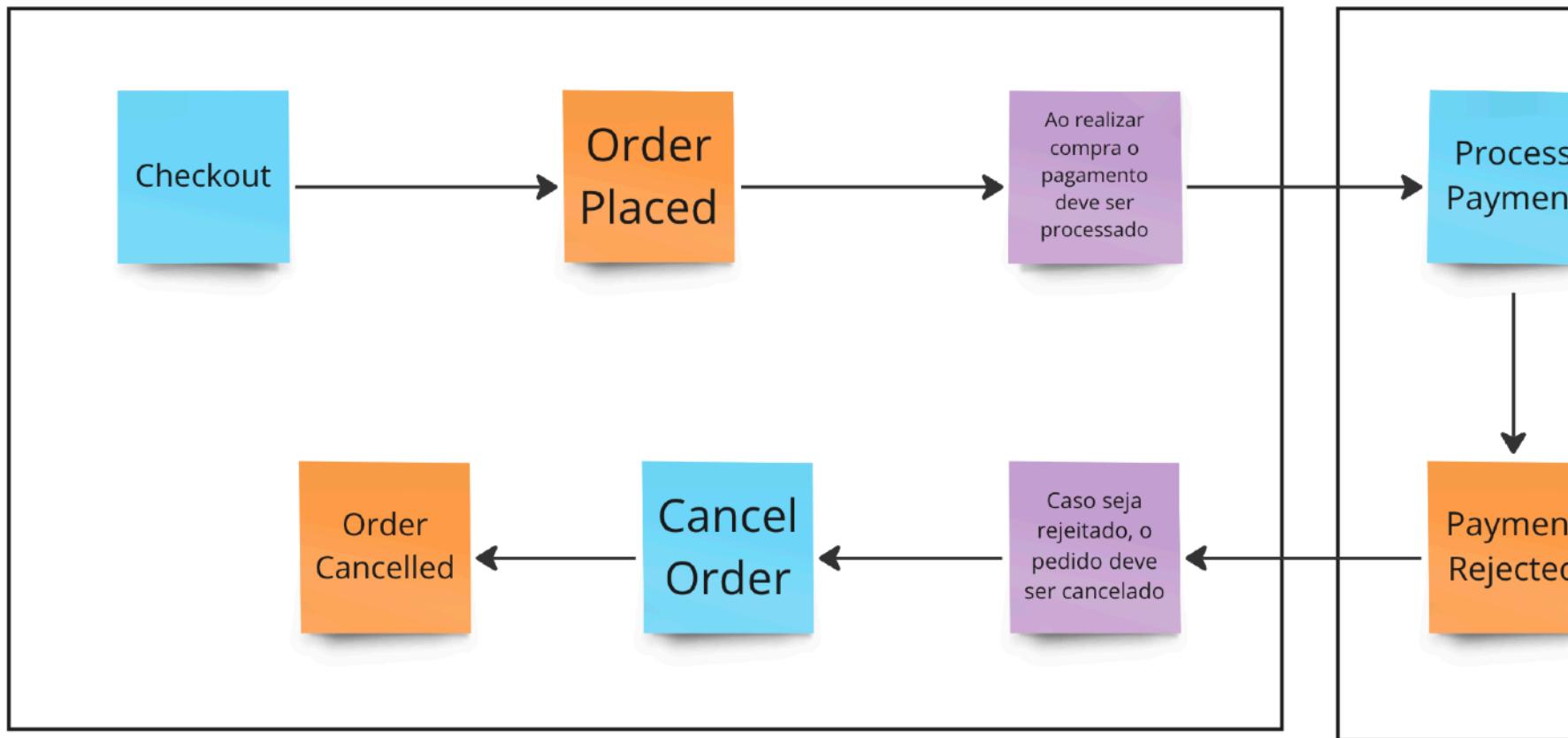
Pivot Transaction

Retriable Transaction

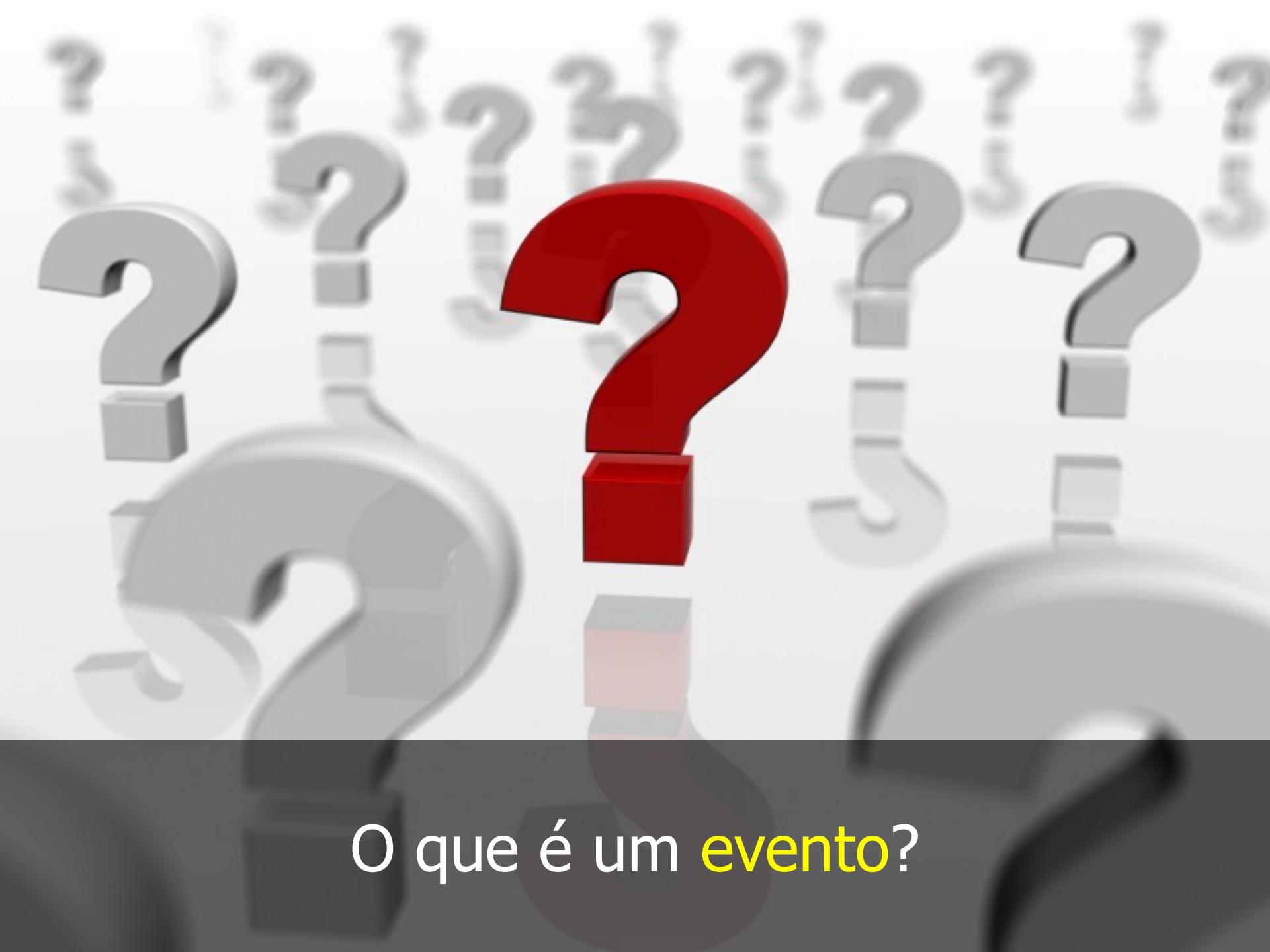


Coreografado: cada participante publica e trata eventos de forma independente, decidindo como realizar a sua parte

Compensable Transaction



Uma **arquitetura orientada a eventos**, ou Event-Driven Architecture, é uma solução para transações distribuídas em um ambiente de microservices tendo baixo acoplamento, de forma assíncrona e sem a necessidade de um orquestrador

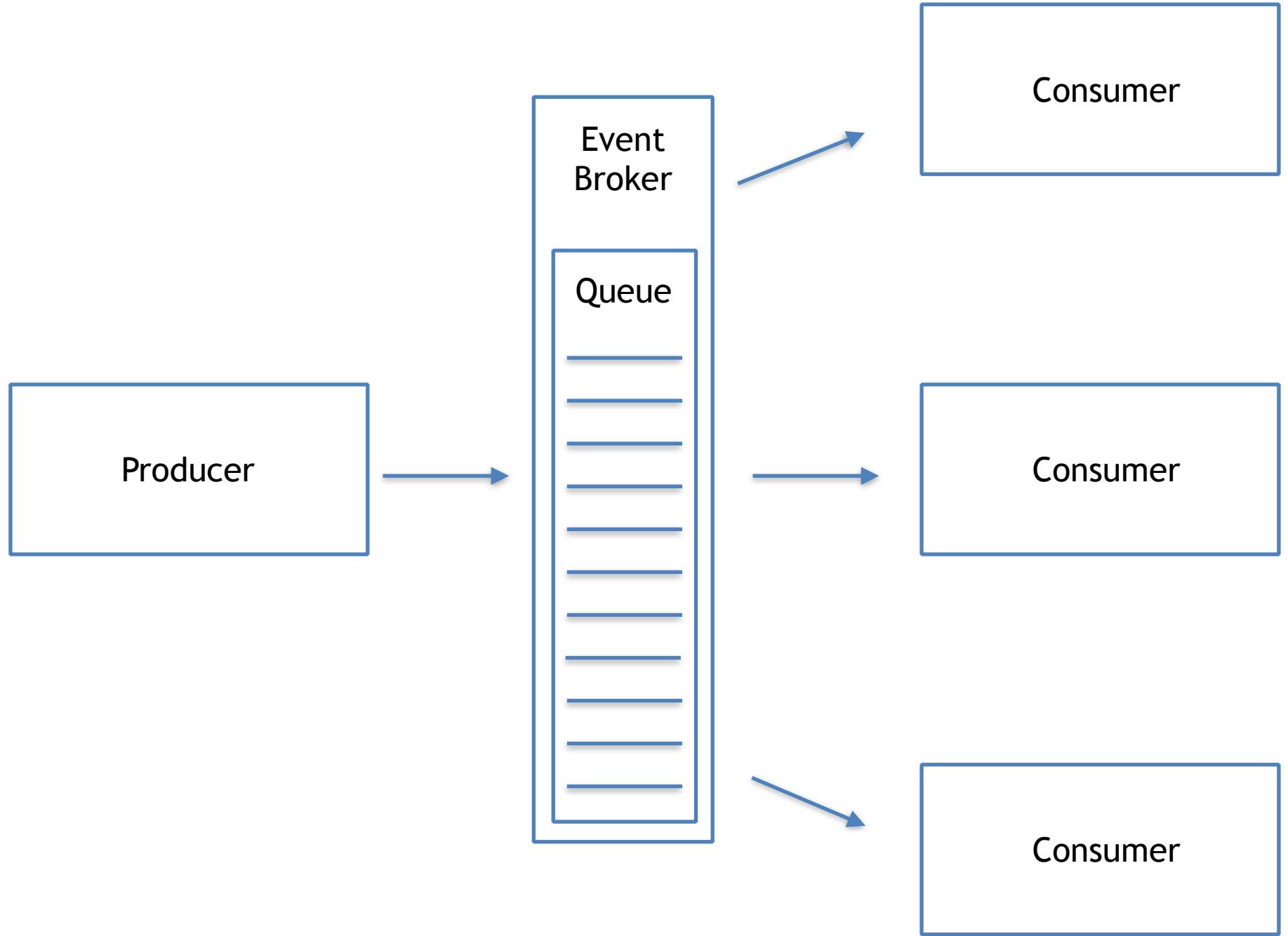


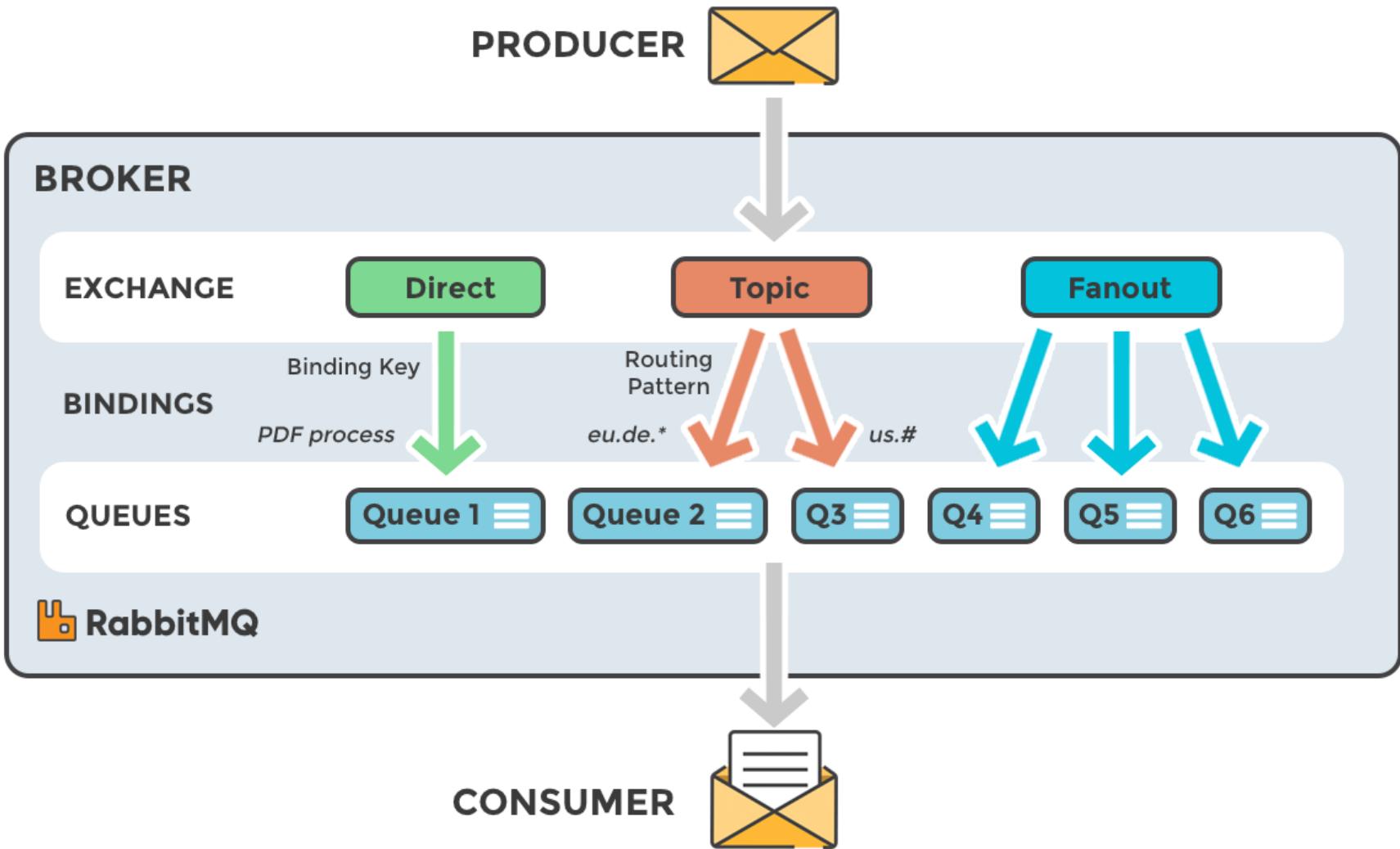
O que é um **evento**?

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

Exemplos

- OrderPlaced
- PaymentApproved
- InvoiceGenerated
- RideRequested
- RideEnded
- PositionUpdated







Porque a **fila** é necessária?



Restaurante

A photograph of a modern hair salon interior. The room is spacious with a polished floor and warm lighting. Along the left wall, there are several styling stations, each featuring a large rectangular mirror with a thin, glowing white border. The mirrors reflect the interior of the salon, showing other stations and a reception area. Each station has a black, ergonomic styling chair with a chrome base. The ceiling is white with recessed lights and a long, linear light fixture running along the edge. In the background, there's a seating area with a sofa and a small table, and a row of chairs along a wall. The overall atmosphere is clean, professional, and contemporary.

Cabelereiro



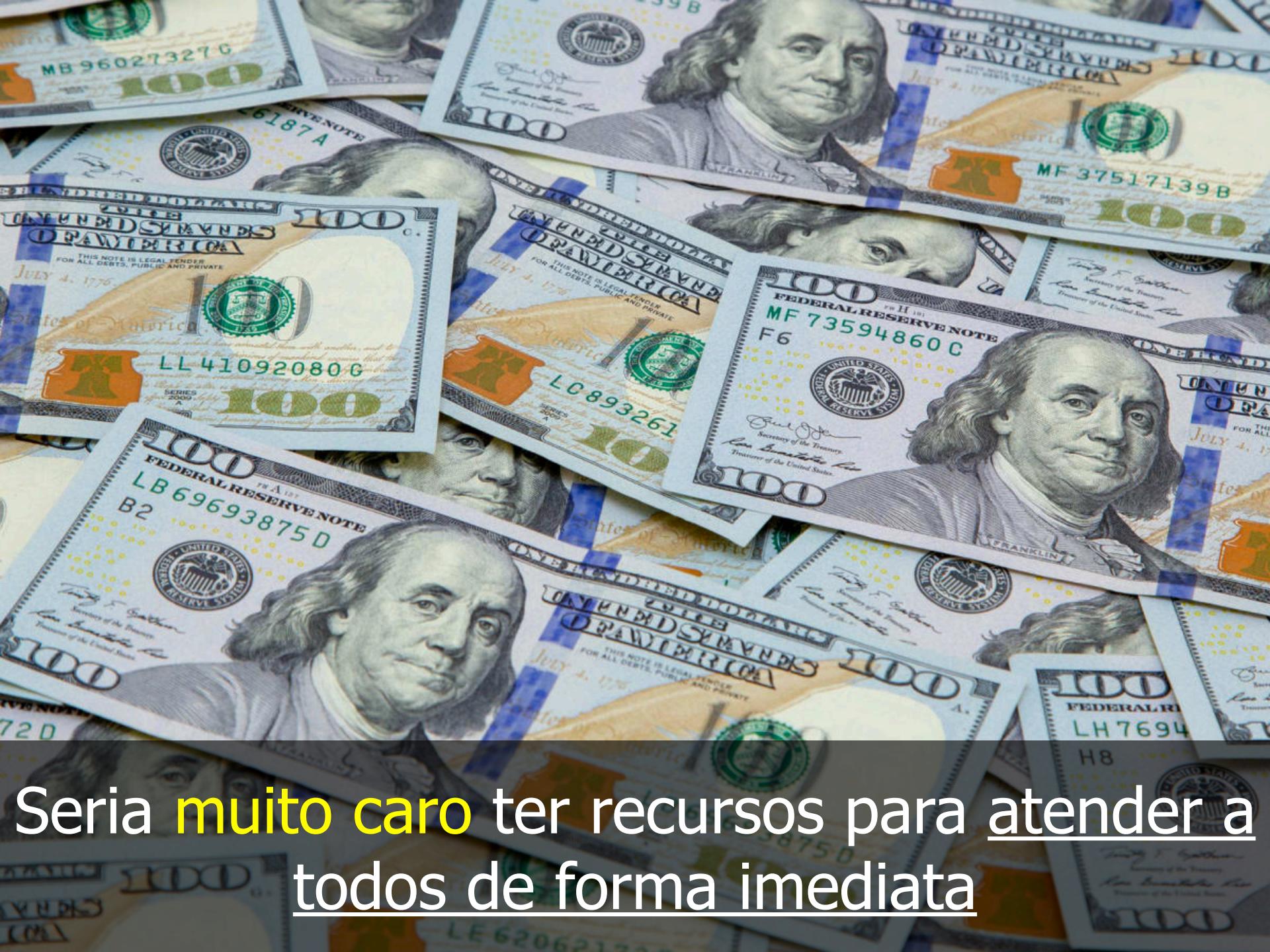
Médico



Pedir cancelamento de qualquer tipo
de serviço por assinatura



Não existem recursos suficientes disponíveis



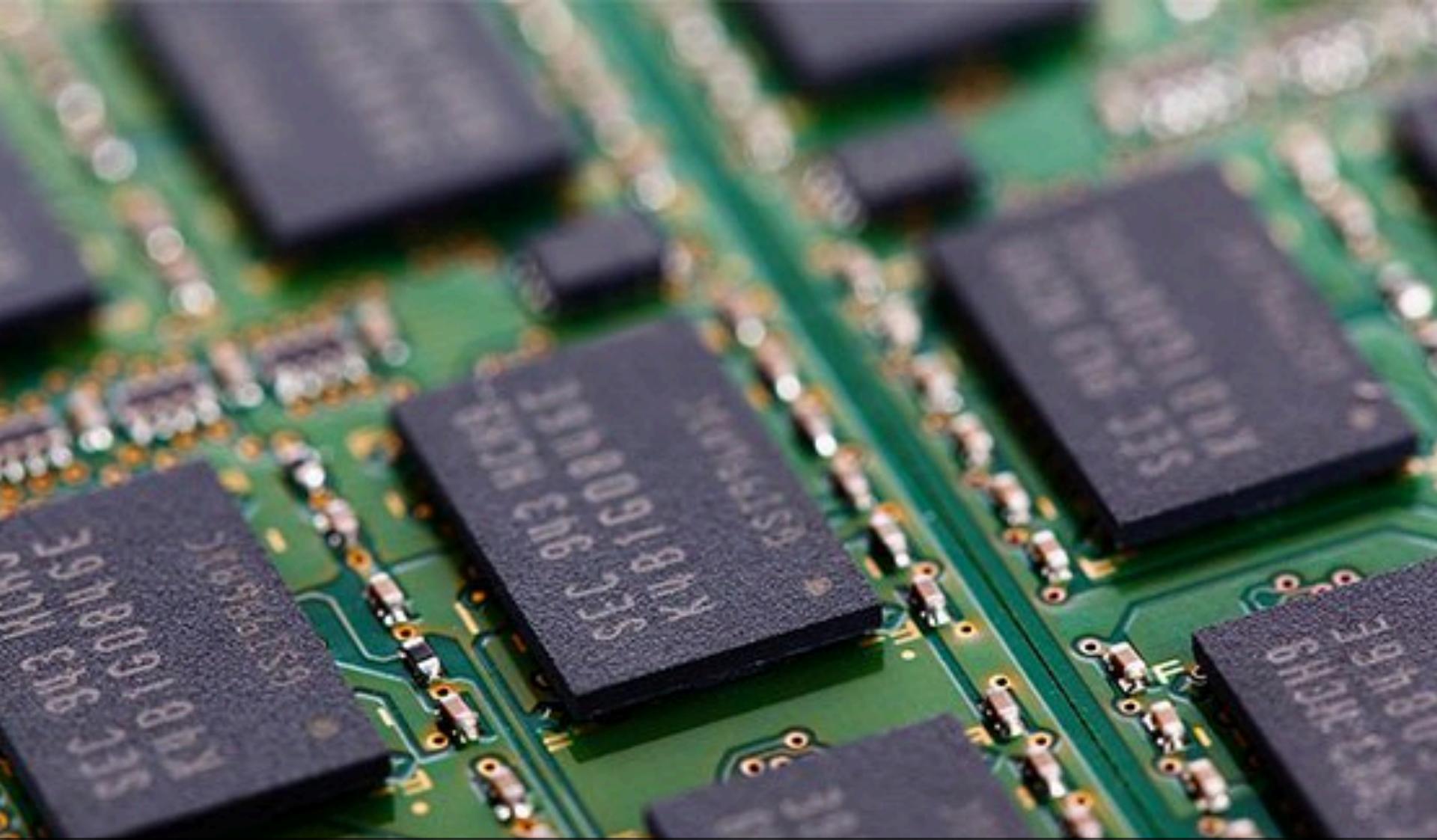
Seria **muito caro** ter recursos para atender a todos de forma imediata



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

Adotar uma arquitetura orientada a eventos tem os seguintes benefícios:

- Baixo acoplamento entre os use cases dentro e fora de um bounded context
- Tolerância a falha com capacidade para retomar o processamento do ponto onde parou
- Disponibilidade e escalabilidade mais alta
- Menos custos com infraestrutura*

Adotar uma arquitetura orientada a eventos tem os seguintes desafios:

- Complexidade técnica mais alta
- Lidar com a **duplicação** de eventos
- Falta de clareza no workflow
- Dificuldade em **tratar e diagnosticar erros**