



MICROSERVIÇOS

Vinicius Soares - Aula 02

Professores

VINICIUS SOARES

Professor Convidado

Entusiasta da Computação Distribuída, Vinicius Soares é Head de Tecnologia em uma das principais empresas do sul do país, ajudando clientes e empresas a alcançarem seus resultados de forma rápida e assertiva. Apaixonado por Java, Arquitetura de Sistemas e Computação em Nuvem, Vinicius possui sólida experiência liderando equipes de Arquitetura usando SOA e Microserviços com tecnologias Open-sources. Compartilha suas experiências através de conteúdo online e eventos nacionais e internacionais como Devoxx, TDC e Campus Party. Como empreendedor, já ajudou mais de 1000 pessoas a se qualificarem para o mercado de TI e atuarem de forma representativa na área.

LUIS FERNANDO PLANELLA GONZALEZ

Professor PUCRS

Doutor Ciências da Computação (PUCRS, 2018). Desenvolvedor e arquiteto Java com experiência profissional desde 1999, certificado pela Sun como programador e desenvolvedor de componentes web na plataforma Java. Entusiasta de software livre.

Ementa da disciplina

Estudo sobre a arquitetura de microserviços. Estudo sobre os conceitos de particionamento de serviços, replicação e distribuição, comunicação assíncrona via filas e Soluções serveless..



Microserviços

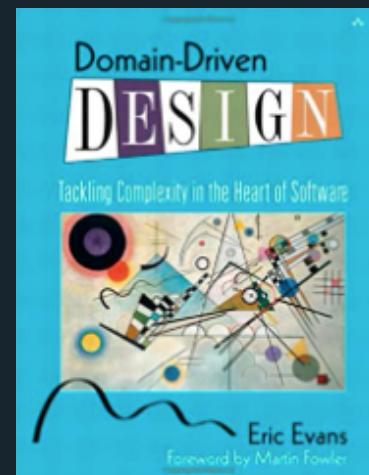
Vinicius Soares

PARTICIONAMENTO DE SERVIÇOS



Domain Drive Design

- Fundamentado na experiência de mais de 20 anos de Eric Evans no desenvolvimento de sistemas, o DDD é uma abordagem que reúne um conjunto de boas práticas, padrões, ferramentas e recursos da orientação a objetos que têm como objetivo a construção e desenvolvimento de sistemas de acordo com o domínio e regras de negócio do cliente.
- Além disso, questões relacionadas ao processo de desenvolvimento, como a necessidade de um estreito relacionamento entre a equipe de programadores e os especialistas do domínio, também são tratadas pela abordagem.



Domain Drive Design

- O principal conceito do DDD é o modelo.
- O modelo expressa o domínio e negócio do cliente e pode ser criado utilizando desenhos, fluxogramas, diagramas, etc.
- O importante é que ele represente o negócio do cliente.
- Como principais componentes do DDD, podemos listar: a linguagem onipresente, a arquitetura em camadas e os padrões.

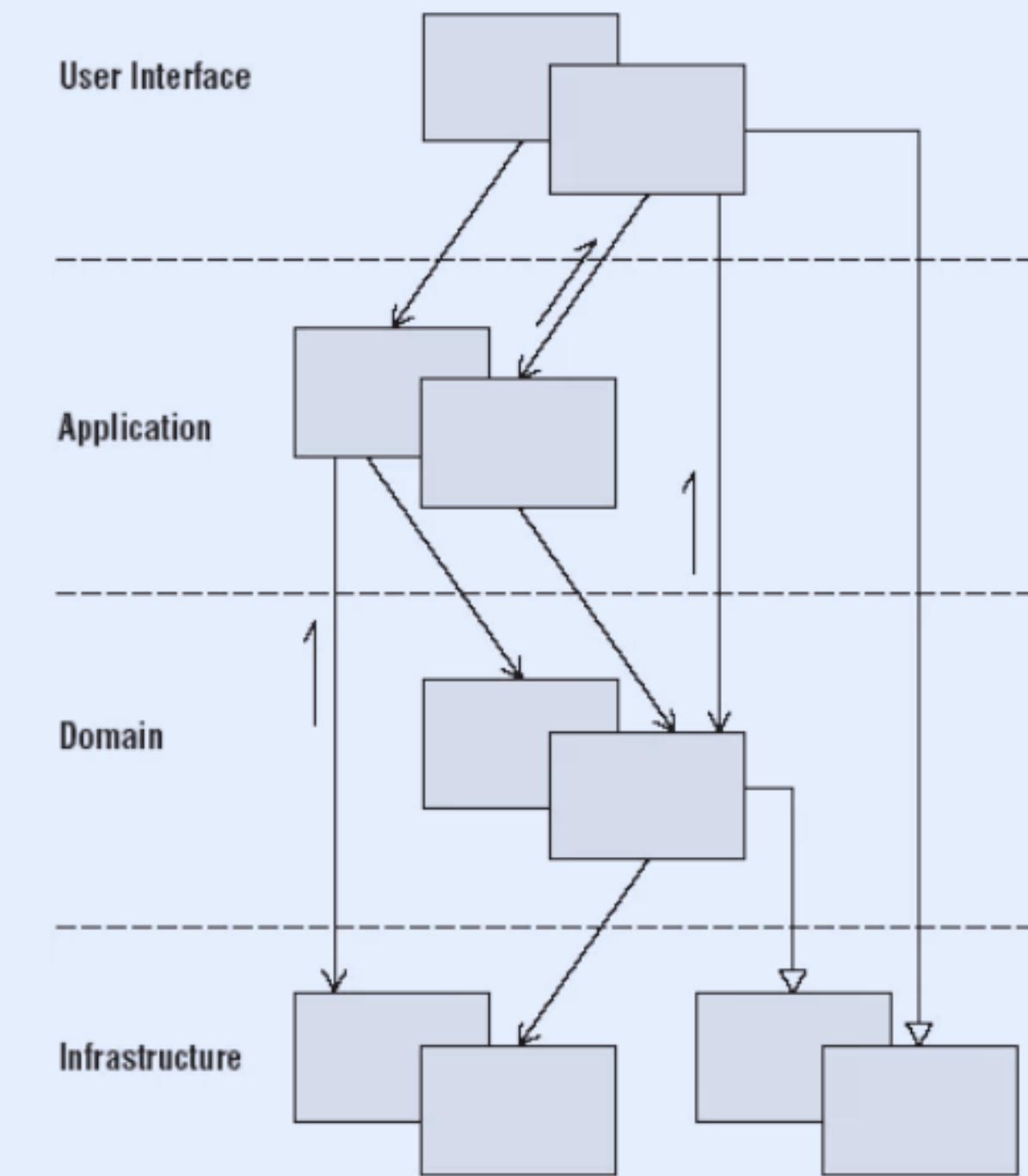
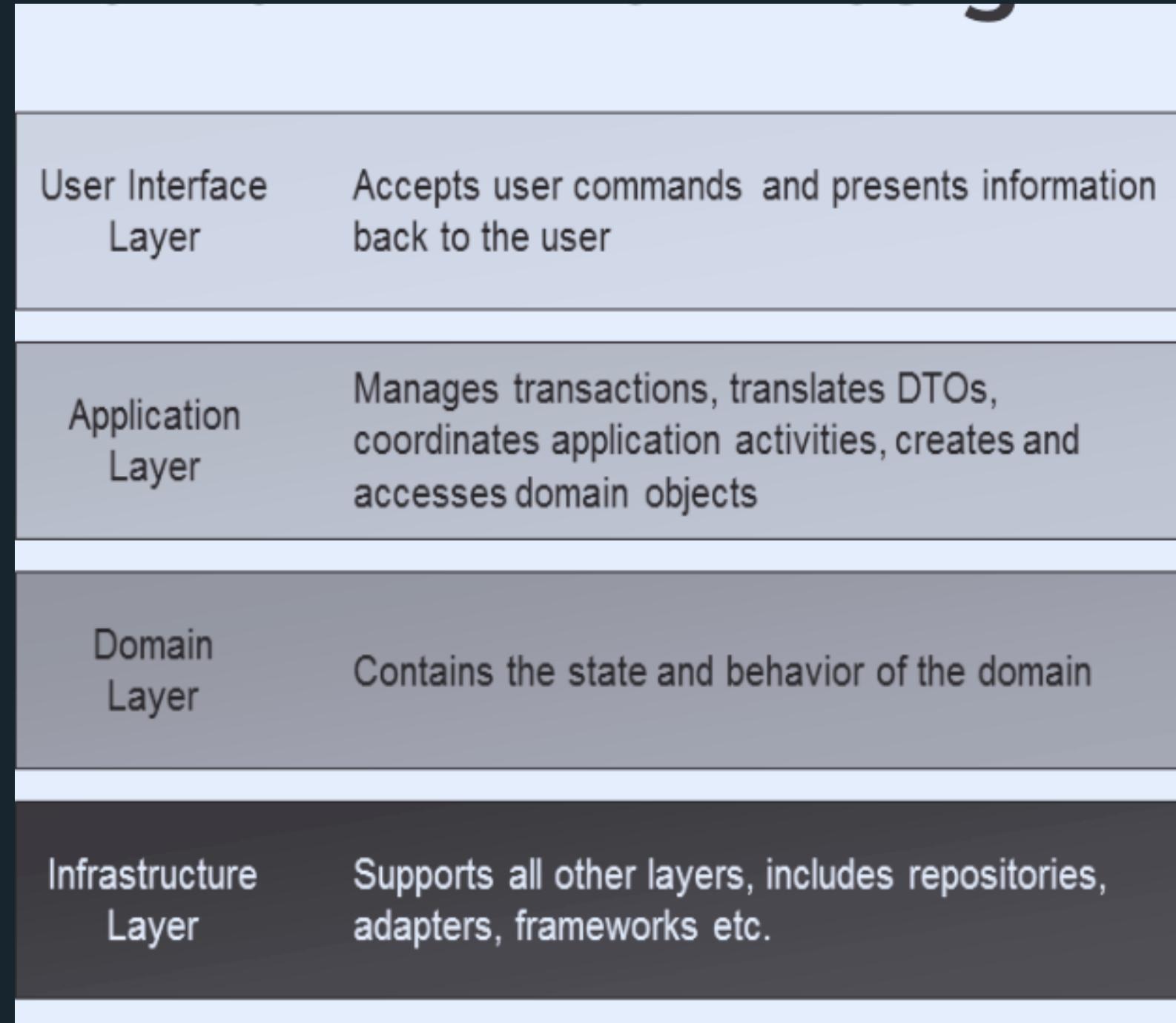
Domain Drive Design

- É uma abordagem de desenvolvimento de software que reúne um conjunto de conceitos, princípios e técnicas cujo foco está no domínio e na lógica do domínio com o objetivo de criar um Domain Model ou (modelo do domínio).
- Significa desenvolver software de acordo com o domínio relacionado ao problema que estamos propondo resolver.
- O foco da abordagem é criar um domínio que “fale a língua” do usuário usando o que é conhecido como linguagem Ubíqua(ubiquitous language ou linguagem Comum,Onipresente)

Domain Drive Design

- Linguagem Ubíqua (linguagem comum) entende-se que ao trabalhar com DDD devemos conversar usando uma mesma língua, em um único modelo, de forma que o mesmo seja compreendido pelo cliente, analista, projetista, desenhista, testador, gerente, etc. nesta linguagem, que seria a linguagem usada no dia a dia.

Domain Drive Design - Camadas



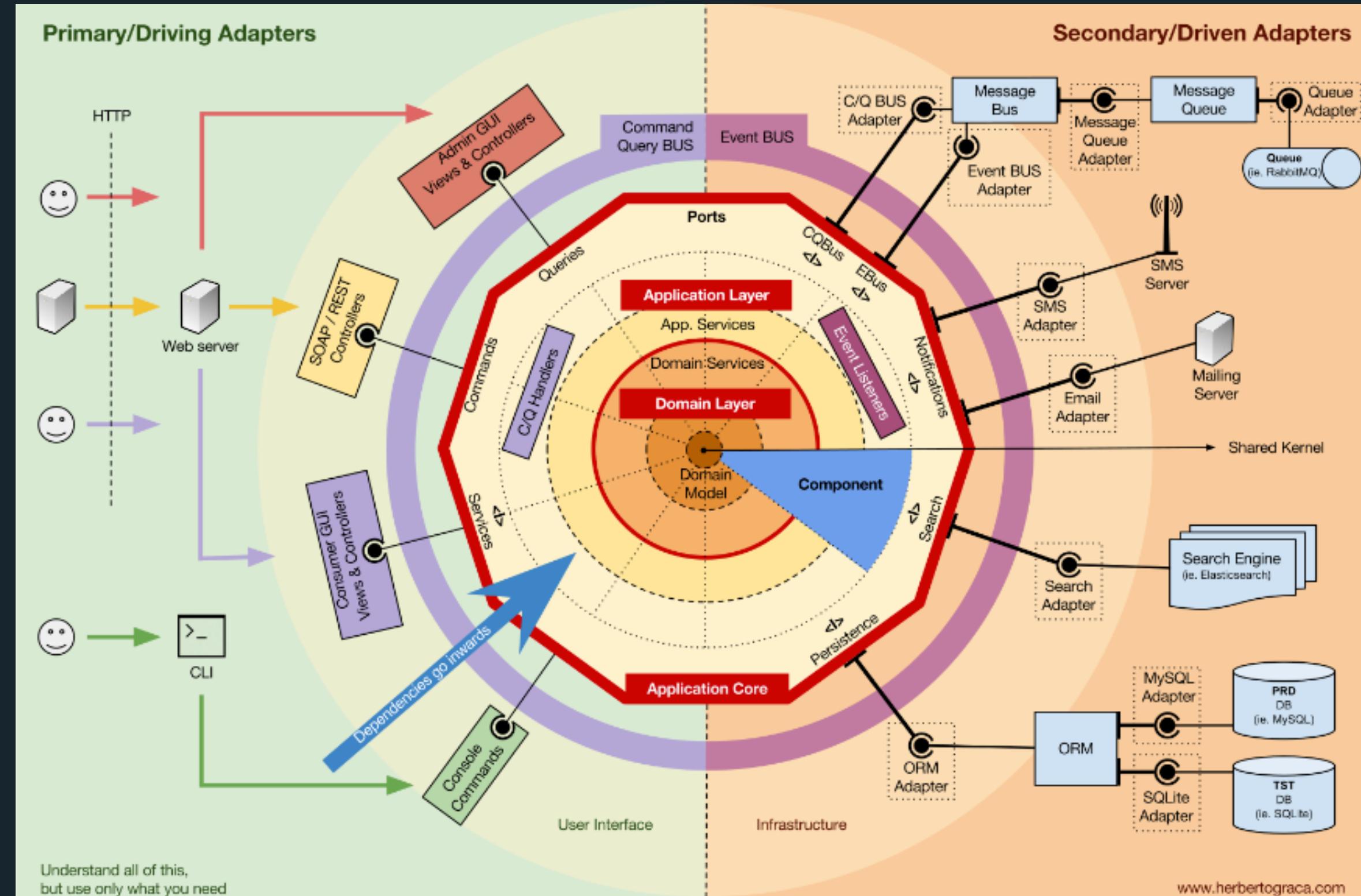
Domain Drive Design - Modelo

- O modelo é evolutivo: A cada iteração entre especialistas de domínio e a equipe técnica, o modelo se torna mais profundo e expressivo, mais rico, e os desenvolvedores transferem essa fonte de valor para o software.
- Assim, o modelo vai sendo gradualmente enriquecido com o expertise dos especialistas do domínio destilado pelos desenvolvedores, fazendo com que o time ganhe cada vez mais insight sobre o negócio e que esse conhecimento seja transferido para o modelo (para o código) através dos blocos de construção do DDD.

Domain Drive Design - Modelo

- Quando novas regras de negócio são adicionadas e/ou regras existentes são alteradas ou removidas, a implementação é refatorada para refletir essas alterações do modelo no código.
- No final, o modelo (que em última instância será o software) vai expressar com riqueza de conhecimento o negócio.

Modelo DDD com arquitetura Hexagonal e Layered



Vantagens

- O código fica menos acoplado e mais coeso.
- O negócio é melhor compreendido por todos da equipe o que facilita o desenvolvimento.
- Alinhamento do código com o negócio.
- Favorecer reutilização.
- Mínimo de acoplamento.
- Independência da Tecnologia.

DDD e Event Storming

- Event Storming é uma técnica de design rápido que engaja especialistas do domínio de negócios com desenvolvedores para que alcancem um ciclo rápido de aprendizagem (aprender o máximo possível no menor tempo possível)
- Segundo Martin Fowler - Bounded Context (Contexto limitado) é um padrão central no design orientado a domínio.
- É o foco da seção de design estratégico da DDD que trata de lidar com grandes modelos e equipes. O DDD lida com modelos grandes, dividindo-os em diferentes contextos limitados e sendo explícito sobre suas inter-relações.

DDD e Event Storming

- Para encontrar os comandos, agregações e boundaries usaremos o conceito de Evento Storming;
- Event Storming é uma técnica de design rápido que engaja especialistas do domínio de negócios com desenvolvedores para que alcancem um ciclo rápido de aprendizagem (aprender o máximo possível no menor tempo possível;



- Mapeando os Eventos
- Identificando os Comandos
- Associando os Aggregates
- Delimitando as Fronteiras do Modelo e
- Identificando Domínios de Negócio

EVENTOS DE DOMÍNIOS

EVENTOS DE DOMÍNIOS

- Use eventos de domínio para implementar explicitamente os efeitos colaterais de alterações em seu domínio. Em outras palavras, e usando terminologia DDD, use eventos de domínio para implementar explicitamente efeitos colaterais entre várias agregações.
- Opcionalmente, para melhor escalabilidade e menor impacto em bloqueios de banco de dados, use consistência eventual entre agregações dentro do mesmo domínio.

EVENTOS DE DOMÍNIOS

- Um evento é algo que ocorreu no passado. Um evento de domínio é algo que ocorreu no domínio que você deseja que outras partes do mesmo domínio (em processo) tenham conhecimento. As partes notificadas geralmente reagem de alguma forma aos eventos.
- Um benefício importante dos eventos de domínio é que os efeitos colaterais podem ser expressos explicitamente.

EVENTOS DE DOMÍNIOS

- Em resumo, eventos de domínio ajudam você a expressar, explicitamente, as regras de domínio, com base na linguagem ubíqua fornecida pelos especialistas do domínio. Os eventos de domínio também permitem uma melhor separação de interesses entre classes dentro do mesmo domínio.
- É importante garantir que, assim como uma transação de banco de dados, todas as operações relacionadas a um evento de domínio sejam concluídas com êxito ou nenhuma delas seja.

EVENTOS DE DOMÍNIOS

- Os eventos de domínio são parecidos com eventos do estilo de mensagens, com uma diferença importante. Com mensagens reais, enfileiramento de mensagens, agentes de mensagens ou um barramento de serviço que usa o AMQP, uma mensagem é sempre enviada de forma assíncrona e comunicação entre processos e computadores.
- Isso é útil para a integração de vários contextos delimitados, microsserviços ou até mesmo aplicativos diferentes. No entanto, com os eventos de domínio, ao acionar um evento na operação de domínio em execução no momento, você deseja que os efeitos colaterais ocorram dentro do mesmo domínio.

EVENTOS DE DOMÍNIOS

- A primeira etapa do Event Storming consiste em mapear os eventos que ocorrem no domínio que está sendo estudado.
- Um evento é qualquer coisa relevante que aconteceu no passado e tende a ser de simples compreensão para pessoas não técnicas.
- O padrão para descrever o evento é utilizar o verbo no passado e deve-se tentar mapear todos os eventos.



EVENTOS DE DOMÍNIOS

- Para essa etapa, utilizamos os post-its de cor laranja.
- Talvez possa existir um pouco de dificuldade inicial para explicar o que é um evento para os especialistas de negócios (não técnicos).
- O facilitador deve cuidar para que, sem que percebam, os participantes fujam do escopo da etapa e começem a citar o mapeamento das entidades ou regras de negócios.
- Alguns exemplos de eventos identificados: Produto Ativado, Produto Desativado, Licença Utilizada, Licença Liberada, Novo Ambiente Criado.



COMANDOS

COMANDOS DE DOMÍNIO

- Nesta etapa, você muda da análise do domínio para os primeiros estágios do design do sistema.
- Até esse momento, você está simplesmente tentando entender como os eventos no domínio se relacionam - é por isso que a participação de especialistas em domínios é tão crítica.
- No entanto, para criar um sistema que implemente o processo de negócios em que você está interessado, é necessário passar à questão de como esses eventos ocorrem.
- Os comandos são o mecanismo mais comum pelo qual os eventos são criados.
- A chave para encontrar comandos é fazer a pergunta: "Por que esse evento ocorreu?".

COMANDOS DE DOMÍNIO

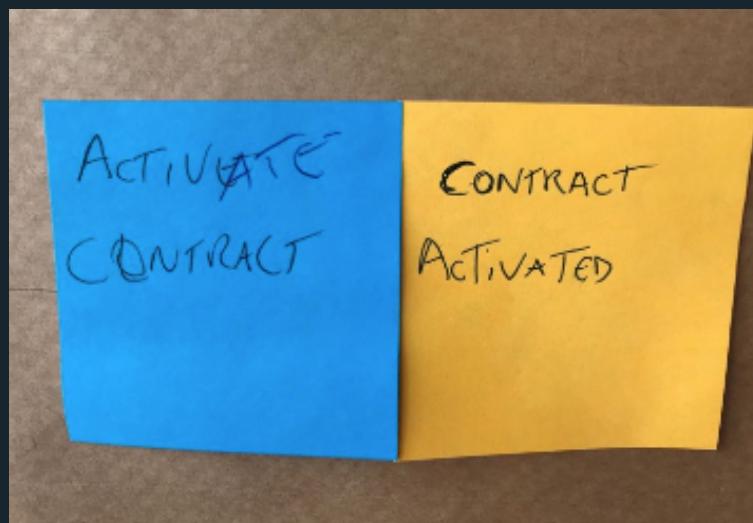
- Nesta etapa, o foco do processo passa para a sequência de ações que levam a eventos;
- Seu objetivo é encontrar as causas pelas quais os eventos registram os efeitos. Os tipos de gatilhos de eventos esperados são:
 - Um operador humano toma uma decisão e emite um comando;
 - Algum sistema ou sensor externo fornece um estímulo (SEDA, EDA, Cron, Event Sourcing);
 - Um evento resulta de alguma política - processamento tipicamente automatizado de um evento precursor;
 - A conclusão de algum período determinado de tempo decorrido;

COMANDOS DE DOMÍNIO

- Outra parte importante do processo que se torna mais detalhada nessa etapa é a descrição de políticas que podem acionar a geração de um evento a partir de um evento anterior (ou conjunto de eventos).
- Avalie se o elemento de dados é uma entidade comercial principal, identificada exclusivamente por uma chave, suportada por vários comandos.
- Tem uma vida útil ao longo do processo de negócios. Isso levará ao desenvolvimento de uma análise do ciclo de vida da entidade.
- Esse primeiro nível de definição de dados ajuda a avaliar o escopo e a responsabilidade do micro serviço à medida que você começa a ver pontos em comum emergindo dos dados usados entre vários eventos relacionados.

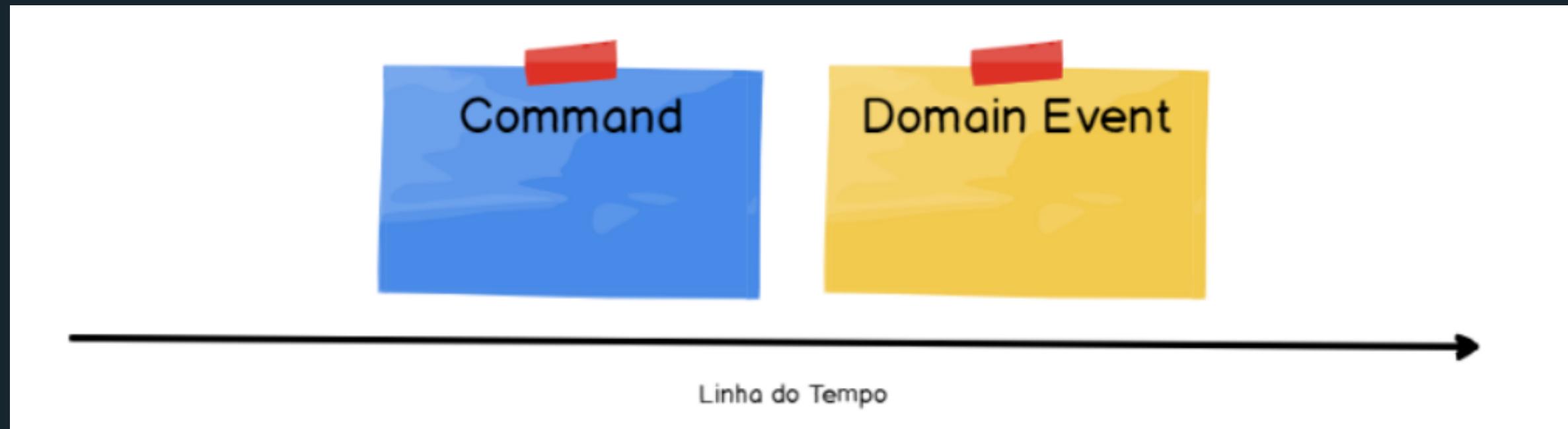
COMANDOS DE DOMÍNIO

- O comando de disparo é identificado em um post-it azul.
- O comando pode se tornar uma operação de micro serviço exposta via API.
- A pessoa humana que emite o comando é identificada e mostrada em uma nota laranja.
- Alguns eventos podem ser criados aplicando políticas de negócios.



COMANDOS DE DOMÍNIO

- Identificação dos comandos que geram os eventos;
- Geralmente os comandos estão associados à alguma ação do usuário, interação com sistema externo ou gerados por um temporizador/cron;
- Verbo na forma imperativa;
- Deve ser colocado no lado esquerdo do evento que ele gera;
- Durante o processo, é comum identificar que um comando pode gerar vários eventos;



AGREGAÇÕES

AGREGAÇÃO

- Agregação é um padrão no Domain Driven Design.
- Um agregado DDD é um cluster de objetos de domínio que podem ser tratados como uma única unidade.
- Um exemplo pode ser um pedido e seus itens de linha, esses serão objetos separados, mas é útil tratar o pedido (junto com seus itens de linha) como um único agregado.
- Um agregado terá um de seus objetos componentes como a raiz agregada. Quaisquer referências externas ao agregado devem apenas ir para a raiz agregada.
- A raiz pode assim garantir a integridade do agregado como um todo.

AGREGAÇÃO

- Agregados são o elemento básico da transferência de armazenamento de dados - você solicita carregar ou salvar agregados inteiros. As transações não devem cruzar fronteiras agregadas.
- Às vezes, os agregados DDD são confundidos com as classes de coleção (listas, mapas, etc.). Agregados DDD são conceitos de domínio (ordem, visita à clínica, lista de reprodução), enquanto as coleções são genéricas.
- Um agregado geralmente contém coleções múltiplas, junto com campos simples.
- O termo "agregado" é comum e é usado em vários contextos diferentes (por exemplo, UML), caso em que não se refere ao mesmo conceito que um agregado DDD.

AGREGAÇÃO

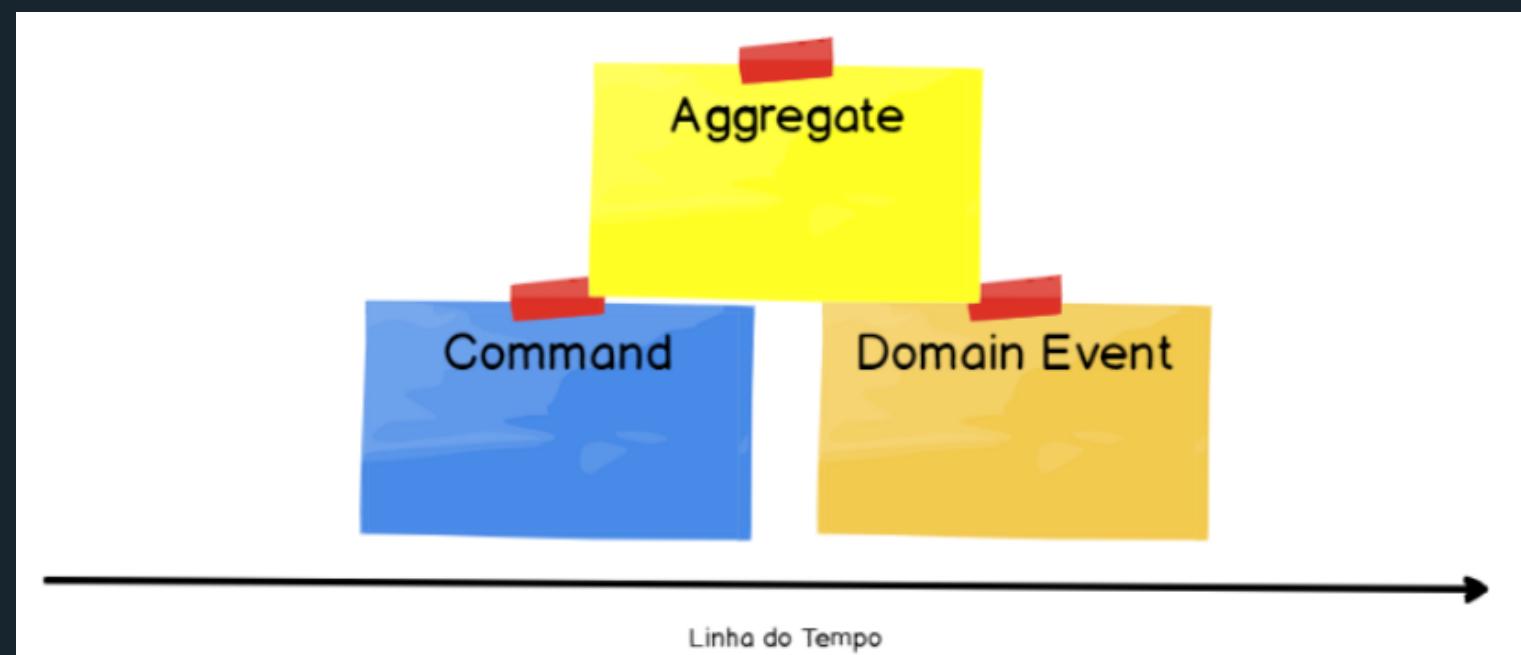
- Os aggregates são a parte do sistema que recebem os comandos e que geram os eventos, eles são os objetos que armazenam os dados e são modificados pelos comandos.
- Aplicativos tradicionais têm usado frequentemente transações de banco de dados para impor a consistência.
- Em um aplicativo distribuído, no entanto, isso muitas vezes não é viável.

AGREGAÇÃO

- Uma única transação empresarial pode abranger vários repositórios de dados, ser demorada ou envolver serviços de terceiros.
- Por fim, cabe ao aplicativo, não à camada de dados, impor as variáveis necessárias para o domínio. É isso que as agregações destinam-se a modelar.

AGREGAÇÃO

- São a parte do sistema que recebem os comandos e que geram os eventos, eles são os objetos que armazenam os dados e são modificados pelos comandos.
- Pode-se utilizar o nome entidade ou dado quando falar sobre Aggregate.
- Durante o exercício, pode ser que os Aggregates se repitam ao longo da linha do tempo, mas não se deve agrupá-los.



LIMITES

LIMITES

- Bounded Context é um padrão central no design orientado a domínio.
- É o foco da seção de design estratégico da DDD que trata de lidar com grandes modelos e equipes.
- O DDD lida com modelos grandes, dividindo-os em diferentes contextos limitados e sendo explícito sobre suas inter-relações.

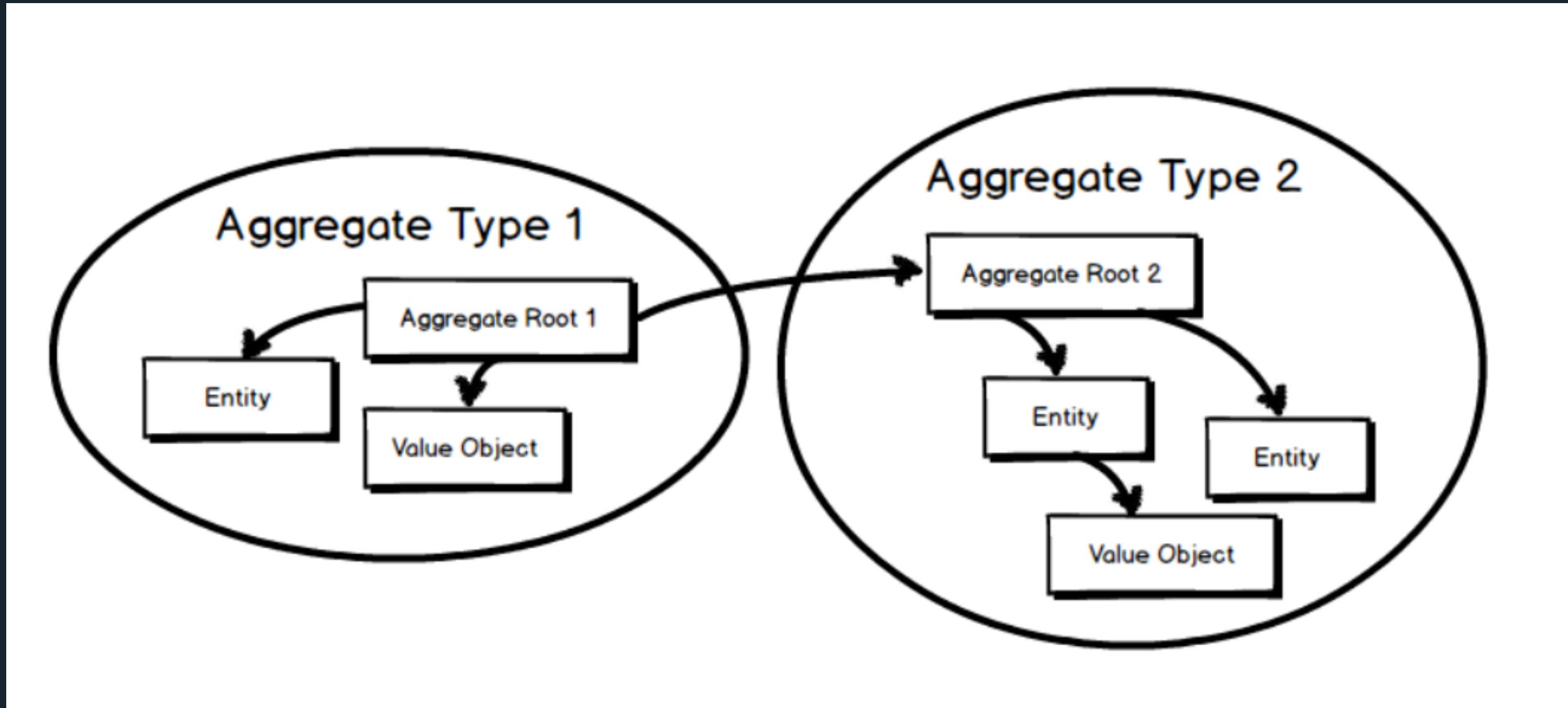
LIMITES

- À medida que você tenta modelar um domínio maior, fica progressivamente mais difícil criar um único modelo unificado.
- Diferentes grupos de pessoas usarão vocabulários sutilmente diferentes em diferentes partes de uma grande organização.
- A precisão da modelagem rapidamente se depara com isso, muitas vezes levando a muita confusão. Normalmente, essa confusão se concentra nos conceitos centrais do domínio.

LIMITES

- Contextos limitados têm conceitos não relacionados (como um tíquete de suporte existente apenas em um contexto de suporte ao cliente), mas também compartilham conceitos (como produtos e clientes).
- Contextos diferentes podem ter modelos completamente diferentes de conceitos comuns, com mecanismos para mapear entre esses conceitos polissêmicos para integração.

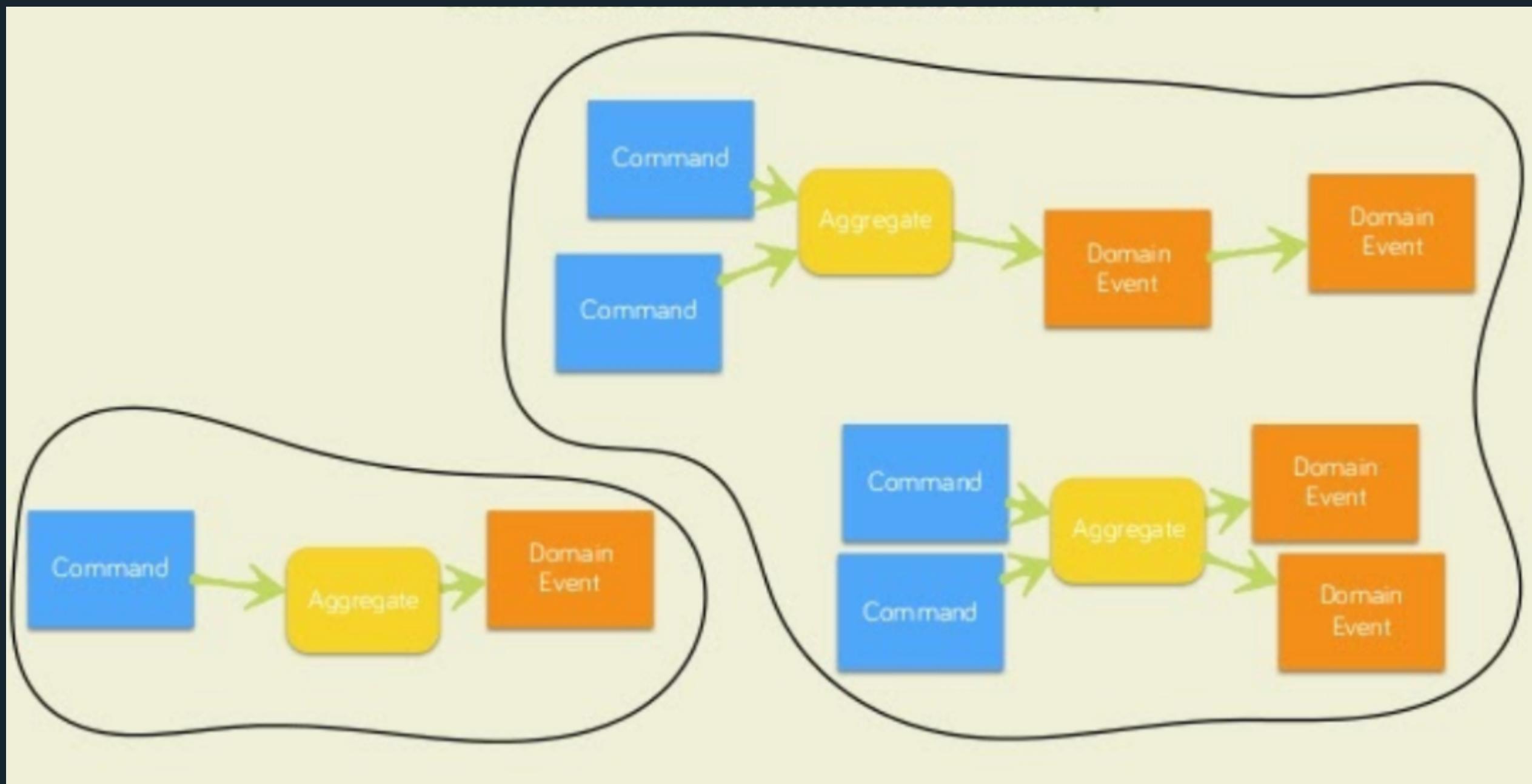
LIMITES



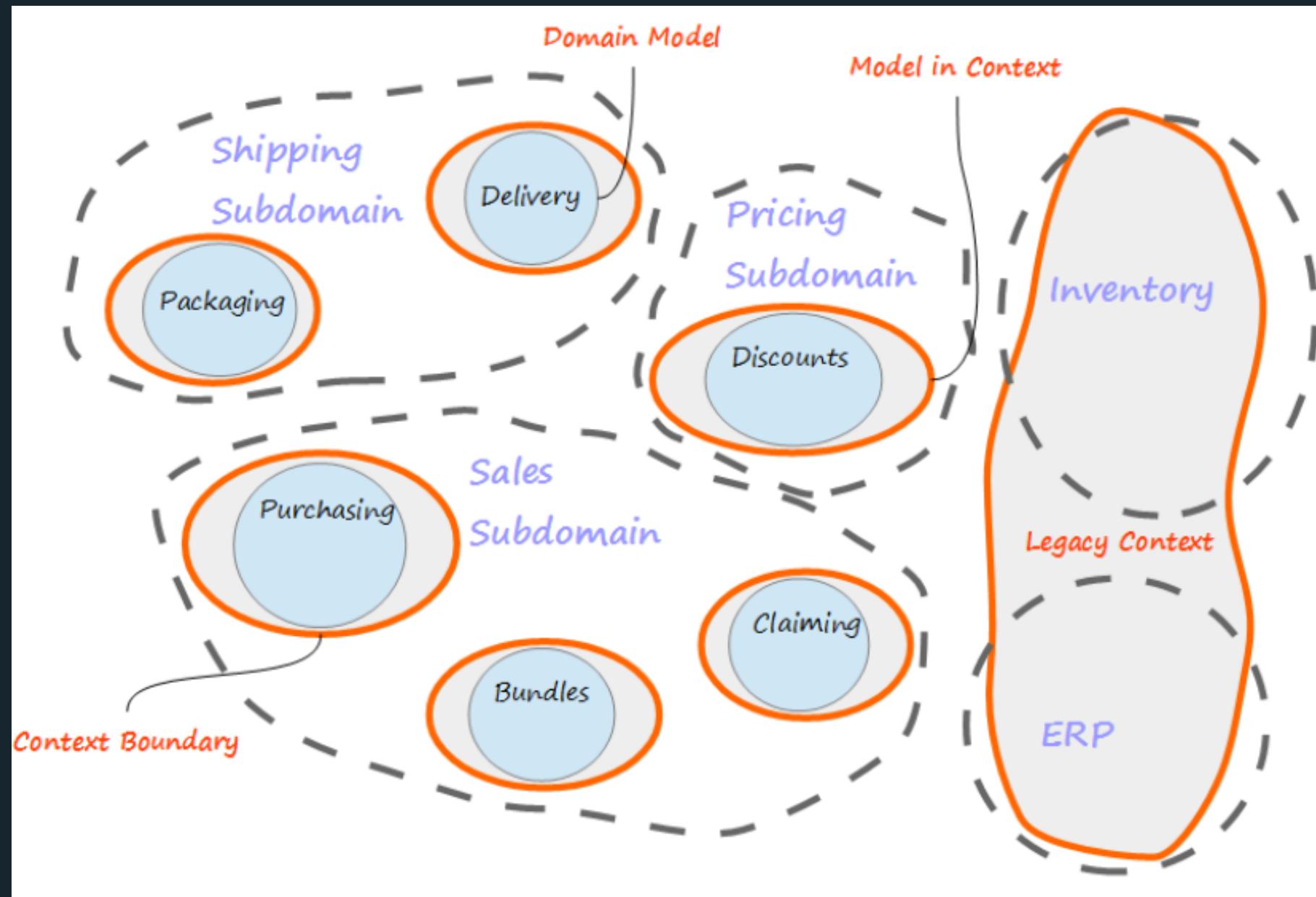
LIMITES

- Vários fatores traçam limites entre contextos.
- Normalmente, a dominante é a cultura humana, já que os modelos agem como linguagem onipresente, você precisa de um modelo diferente quando a linguagem muda.
- Você também encontra vários contextos no mesmo contexto de domínio, como a separação entre os modelos de banco de dados relacional e na memória em um único aplicativo.
- Esse limite é definido pela maneira diferente como representamos os modelos.

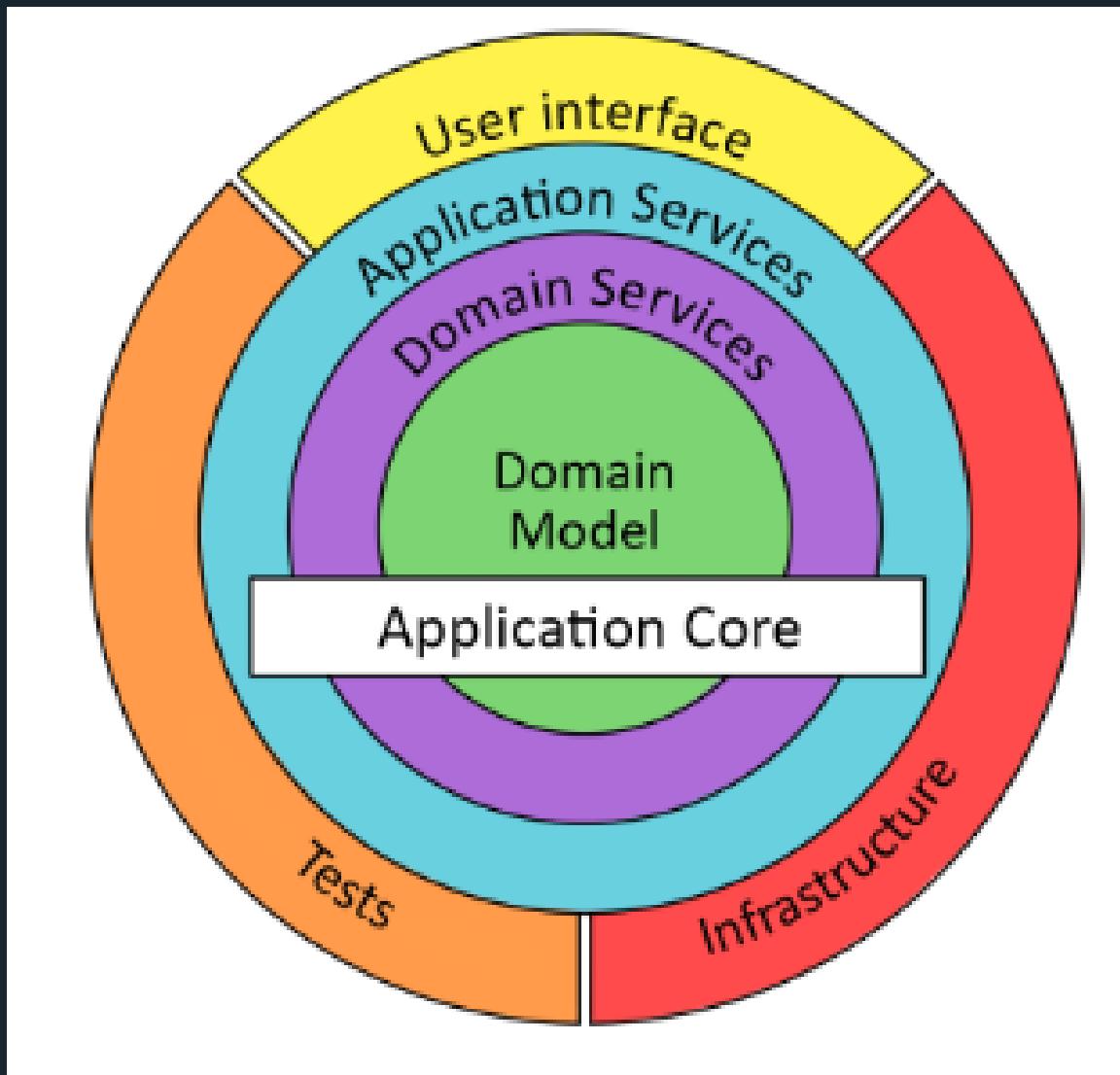
LIMITES



LIMITES

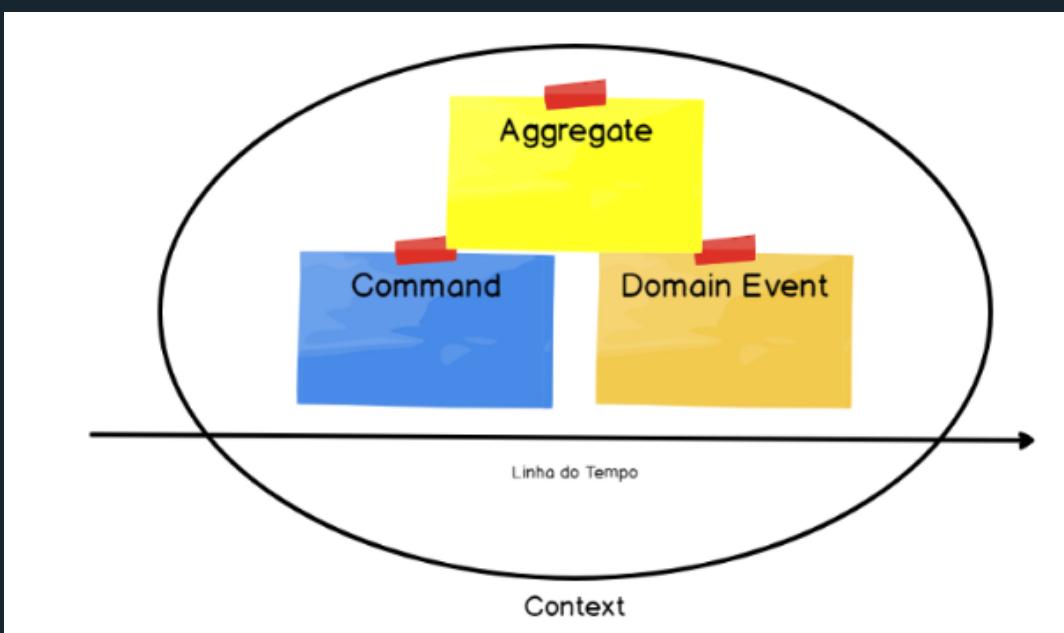


LIMITES



LIMITES

- Podem estar relacionados à divisões departamentais.
- Podem ser diferentes visões que os especialistas do negócio possuem sobre o mesmo conceito.
- Agregadores que são importantes.
- Mapear eventos que “naveguem” entre os domínios.





COMUNICAÇÃO ENTRE SERVIÇOS

Comunicação entre serviços

"HTTP é o padrão de comunicação no microservices"

Martin Fowler

"Os serviços devem ser assíncronos"

Jonar Bonér

Características

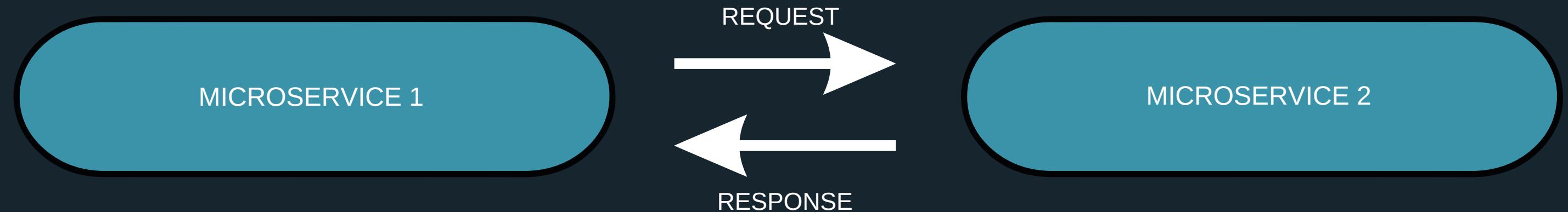
- Síncrona:
 - Espera resposta;
 - Comunicação em “tempo real”;
 - Balanceador de carga a nível de infraestrutura;
 - Tratamento de erros pode ser pelo status do http;

Características

- Assíncrona
 - Não espera resposta;
 - Comunicação entre os dados pode ser “delay” entre as estruturas;
 - Balanceador de carga pode ser uma fila;
 - Possibilidade de Service Discovery / Message broker;
 - Tratamento de erros pode ficar no gerenciador da fila;
 - Em caso de erros o gerenciador de mensagens pode tratar o reenvio;

Implementações

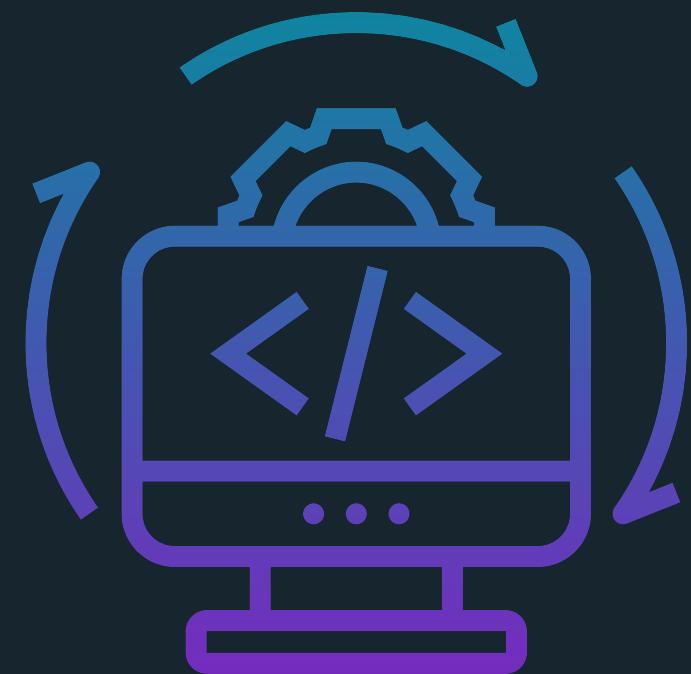
- Síncrona:
 - REST, SOAP, CDI;



- Assíncrona:
 - Mensagens, Eventos, Replicação de base;



COMUNICAÇÃO ORIENTADA A EVENTOS



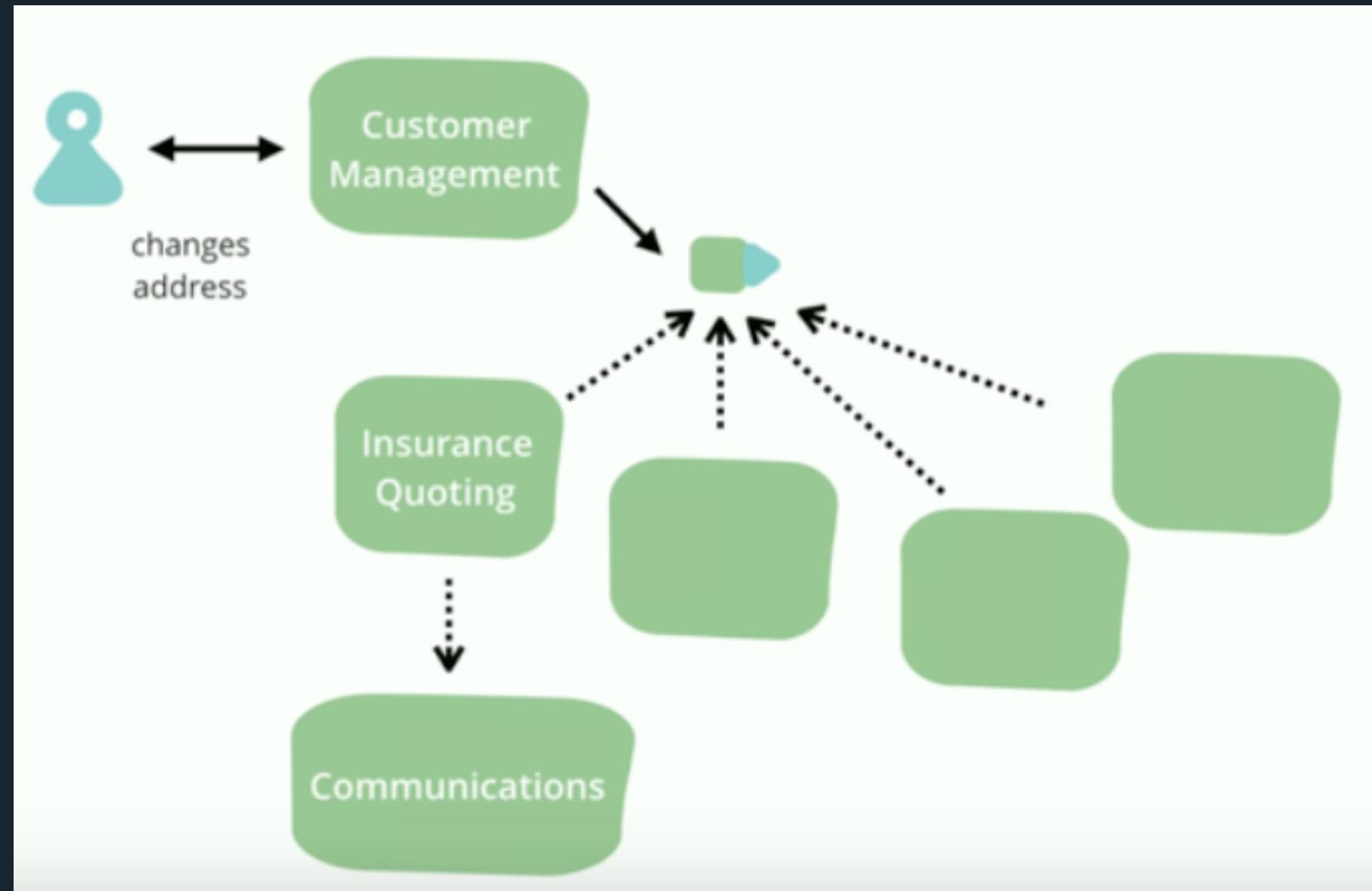
Características

- Mudanças de estado;
- Dispare e esqueça;
- Atores de eventos (consumidores x produtores);
- Acontecimentos no passado;
- Estados imutáveis;
- Diminui acoplamento entre aplicações;
- Processos assíncronos;
- Encaixa perfeitamente com patterns (CQRS / DDD);
- Reprodução de estados;

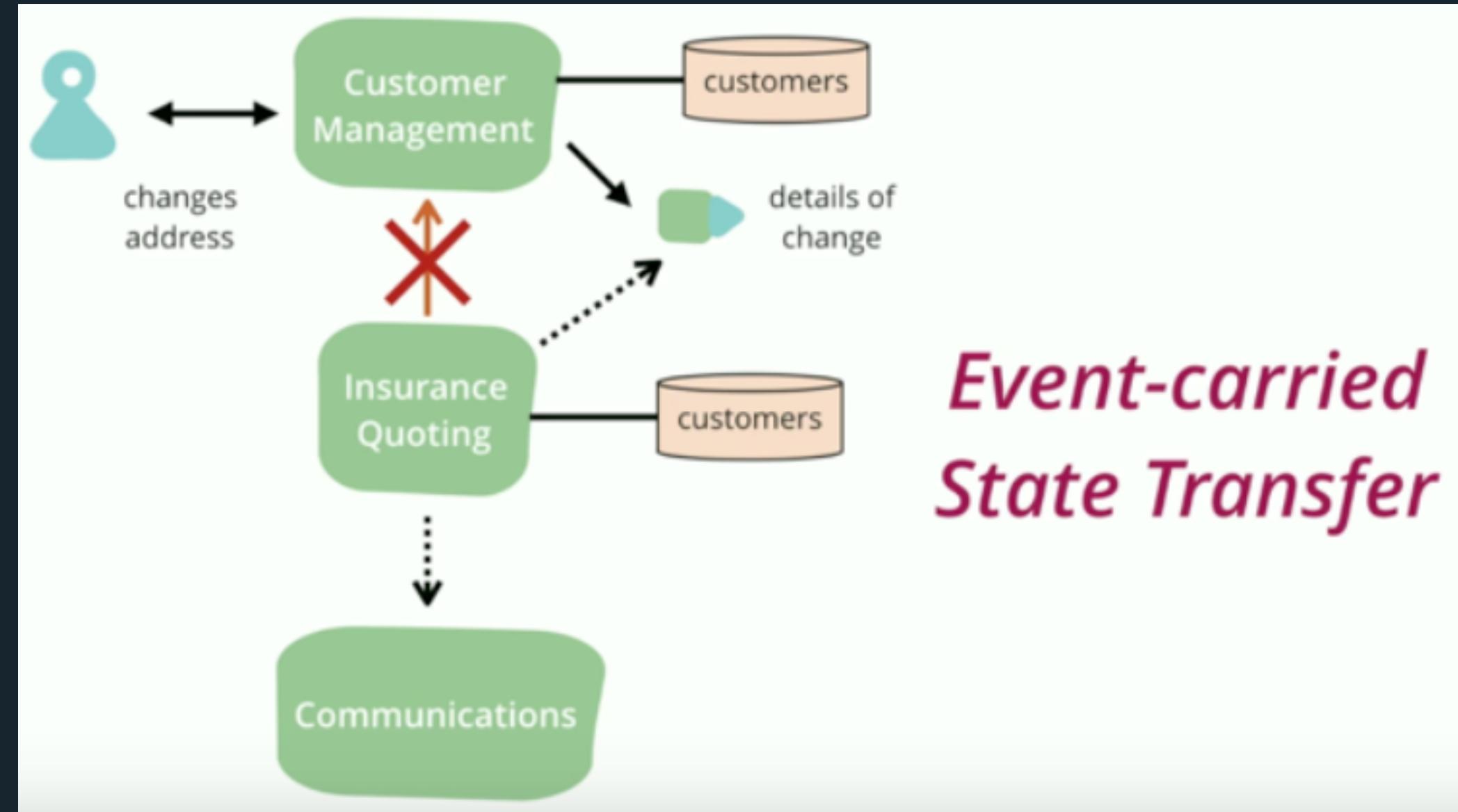
Características

- Segundo Martin Fowler, 4 temas estão sempre presentes:
 - Event Notification;
 - Event-carried state transfer;
 - Event Sourcing;
 - CQRS;

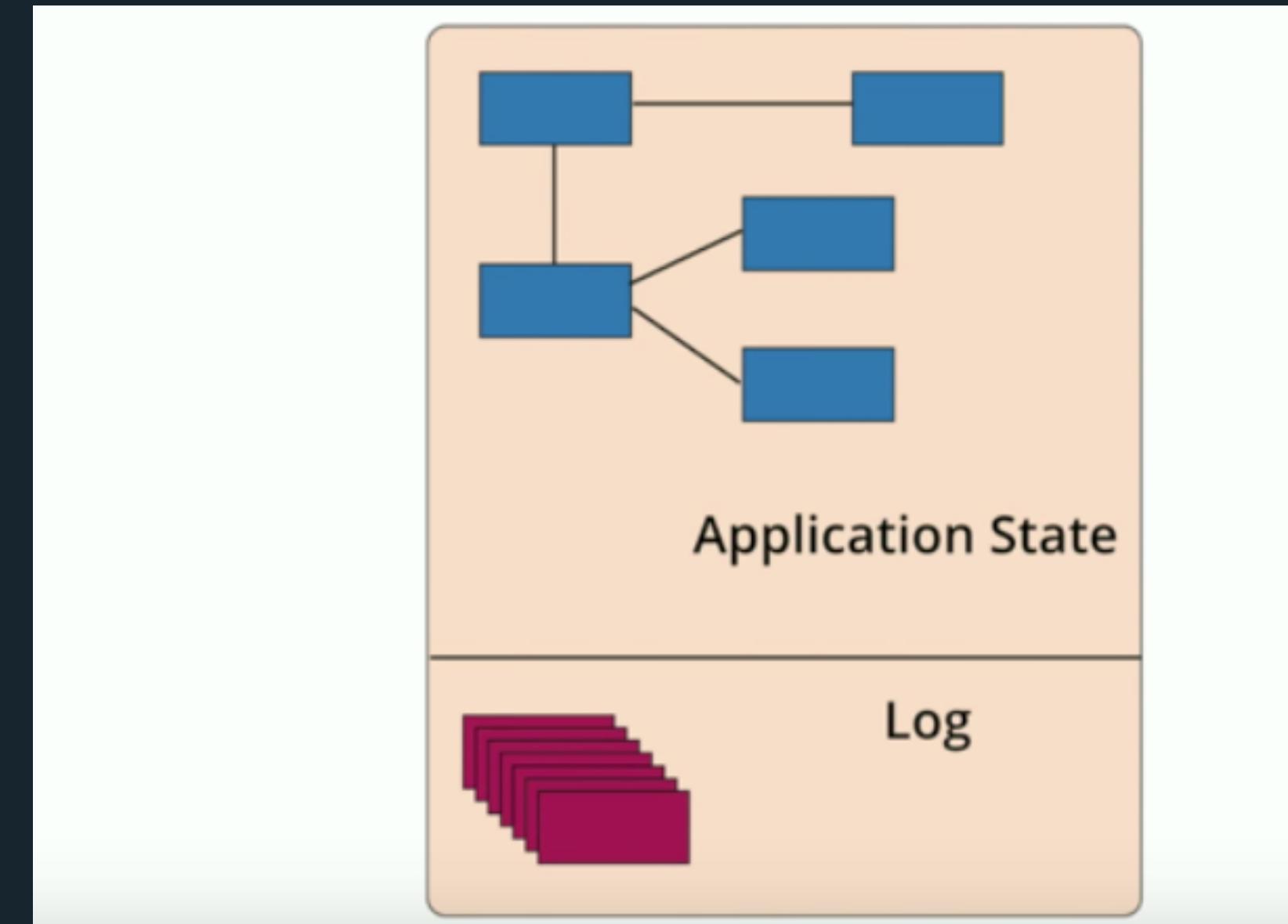
Event Notification



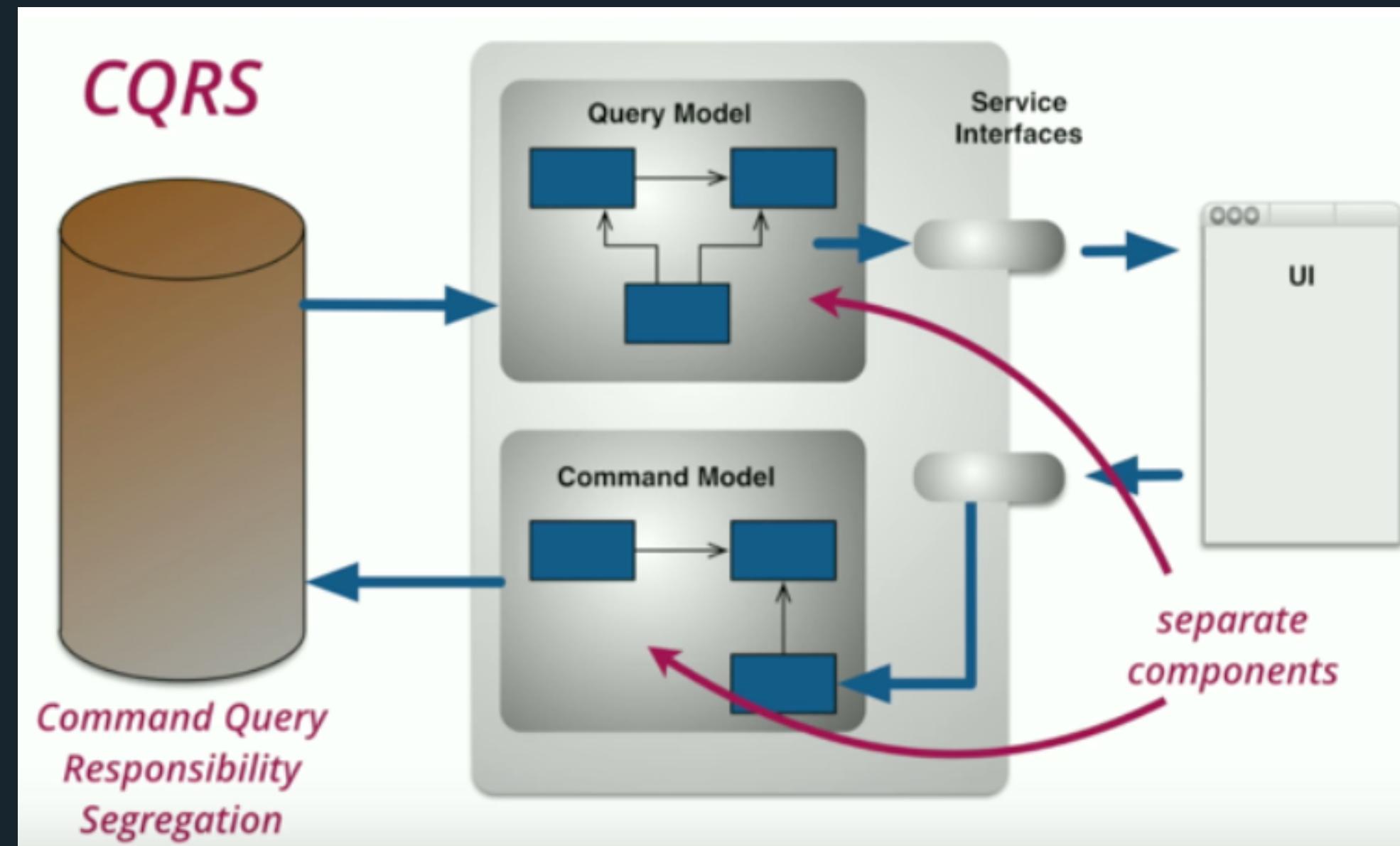
Event-carried State Transfer



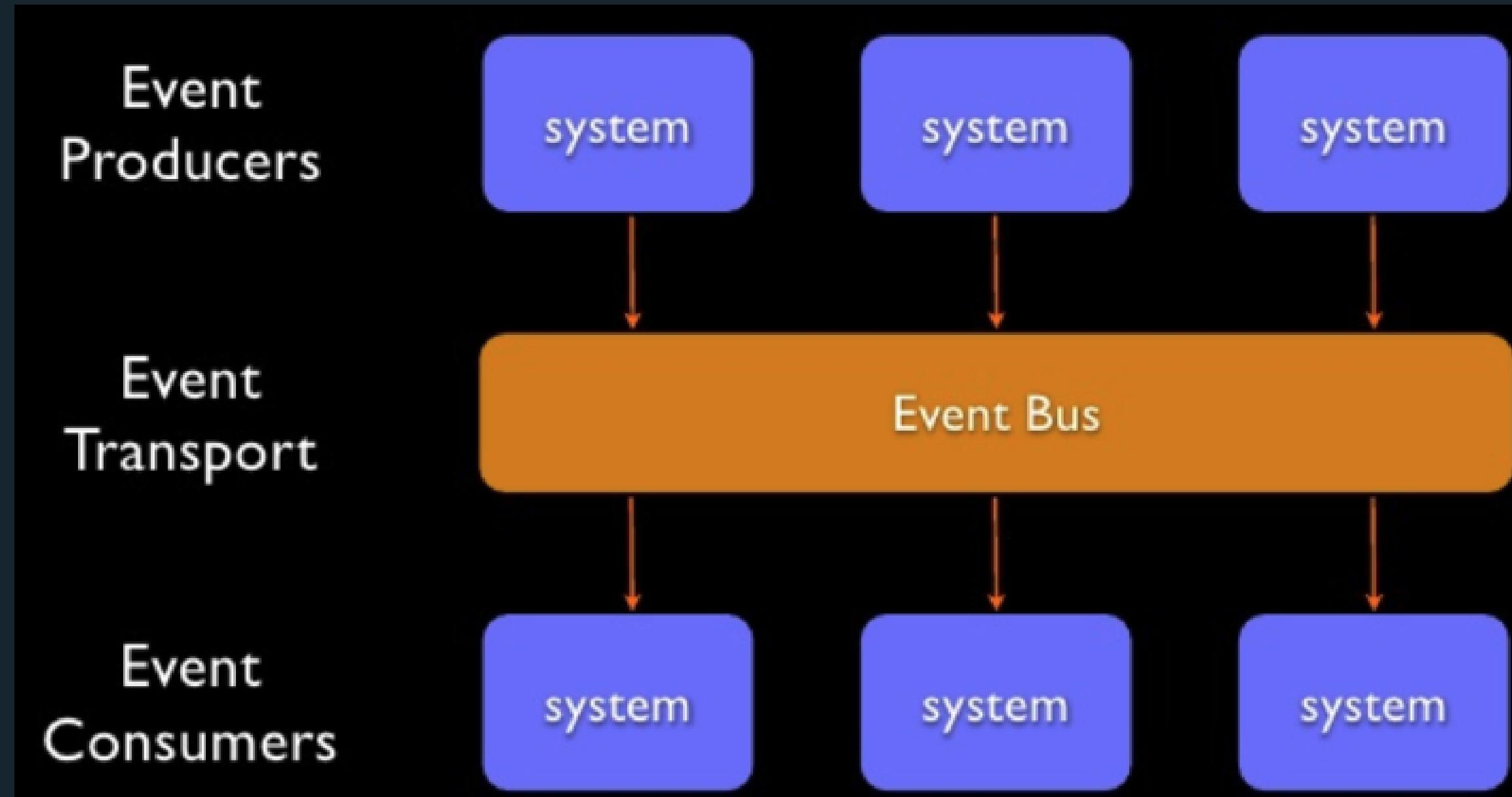
Event Sourcing



CQRS



Arquitetura Típica



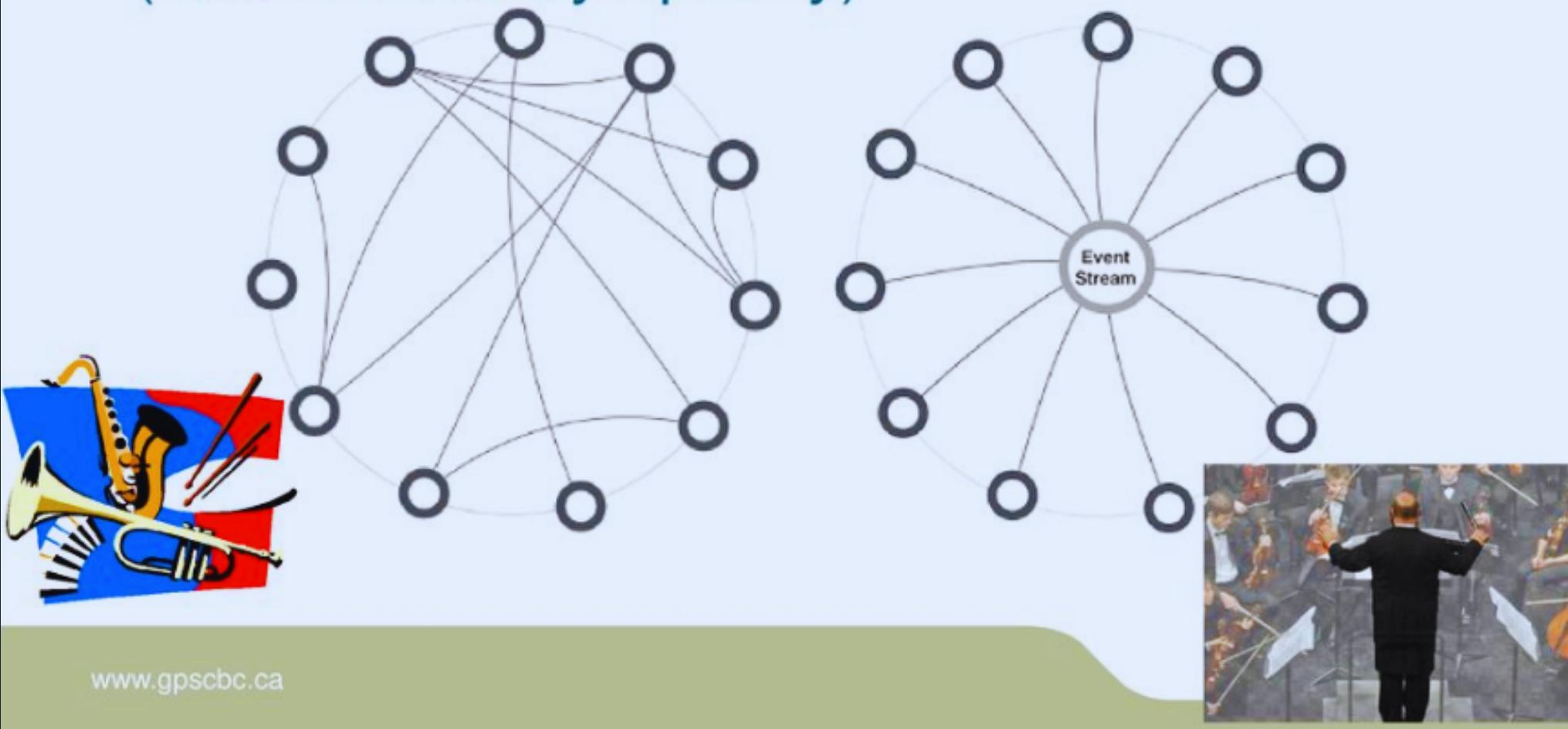
Quando utilizar

- Arquitetura distribuída;
- Arquitetura Microsserviços;
- Volumetria dos dados;
- Responsividade;
- Escalabilidade;

Benefícios

- Suporte a demanda de negócio com melhor serviço:
 - menos espera, sem processos batch;
- Sem integrações ponto a ponto:
 - Dispare e esqueça;
- Possibilita grandes performances:
 - Uso de brokers poderosos (Kafka);

Choreography versus Orchestration (Jazz versus Symphony)



Orquestração

- Processo sincronizado de comunicação;
- Possui uma lógica na sequência de eventos;
- Existe um ponto de partida e chegada;
- Você sabe exatamente o resultado ao final da execução;
- Acoplamento apertado;
- Confiança em APIs RESTFull;

Orquestração - Vantagens

- Acoplamento de serviço solto para agilidade e tolerância a falhas;
- Desenvolvimento mais rápido e ágil;
- Aplicações mais consistentes e eficientes;

Orquestração - Desvantagens

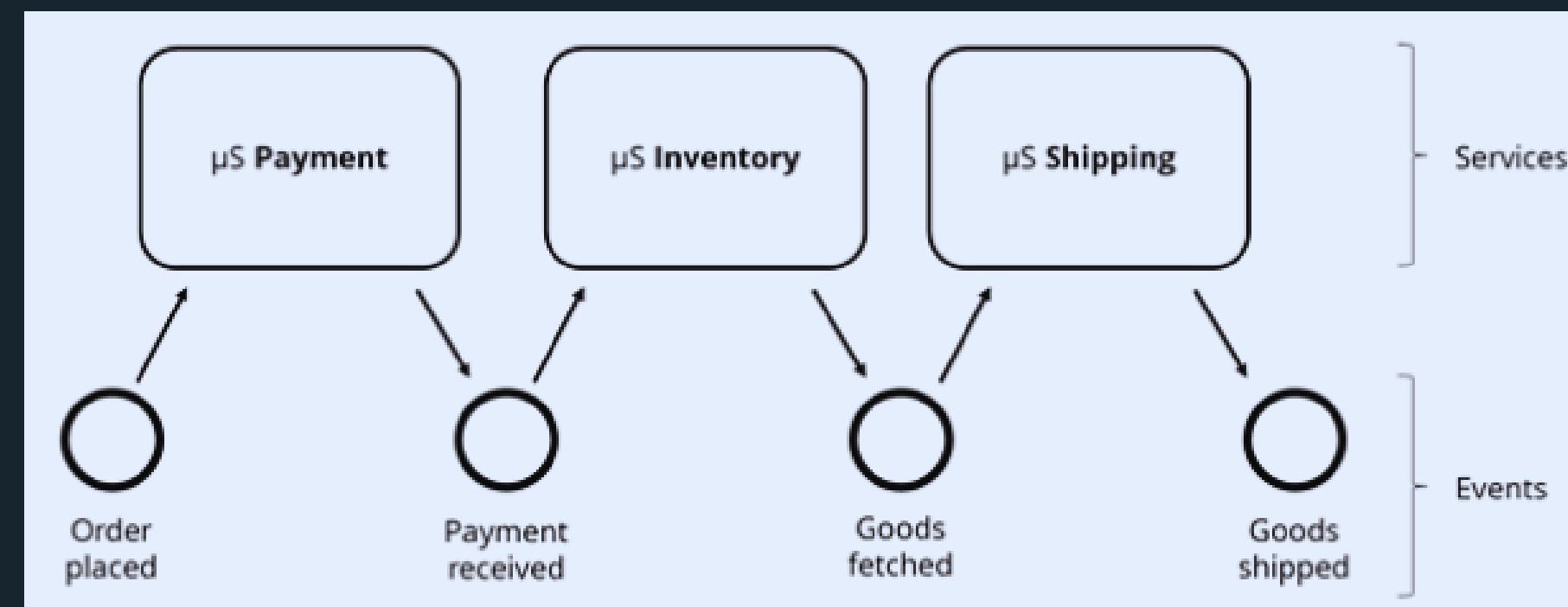
- Uma desvantagem da orquestração é que o controlador das chamadas precisa se comunicar diretamente com cada serviço e aguardar a resposta de cada um deles. Agora que essas interações estão ocorrendo na rede, as invocações demoram mais e podem ser afetadas pela disponibilidade da rede e do serviço que está sendo invocado.
- Em ambientes menores, isso pode funcionar bem, mas as coisas desmoronam quando você está falando de centenas ou mesmo milhares de microsserviços. Você basicamente criou um aplicativo monolítico distribuído que é mais lento e quebradiço do que os do passado! Assim como um maestro perderia sua capacidade de gerenciar efetivamente uma orquestra maciça, porque cada músico aguarda atenção individual, não é viável solicitar a um serviço de controle para gerenciar tantos microsserviços.

Coreografia

- Fluxos de longa duração:
 - Coreografia pode ter um longo fluxo de ações a serem realizadas. Referindo-se ao termo "longa duração", queremos dizer que pode levar minutos, horas ou até semanas até que o processamento do pedido seja concluído;
- Colaboração de eventos:
 - O ponto central da idéia de colaboração de eventos é que todos os microsserviços publicarão eventos quando algo relevante para o negócio acontecer dentro deles. Outros serviços podem se inscrever nesse evento e fazer algo com ele, por exemplo armazenar as informações associadas de forma ideal para seus próprios fins;

Coreografia

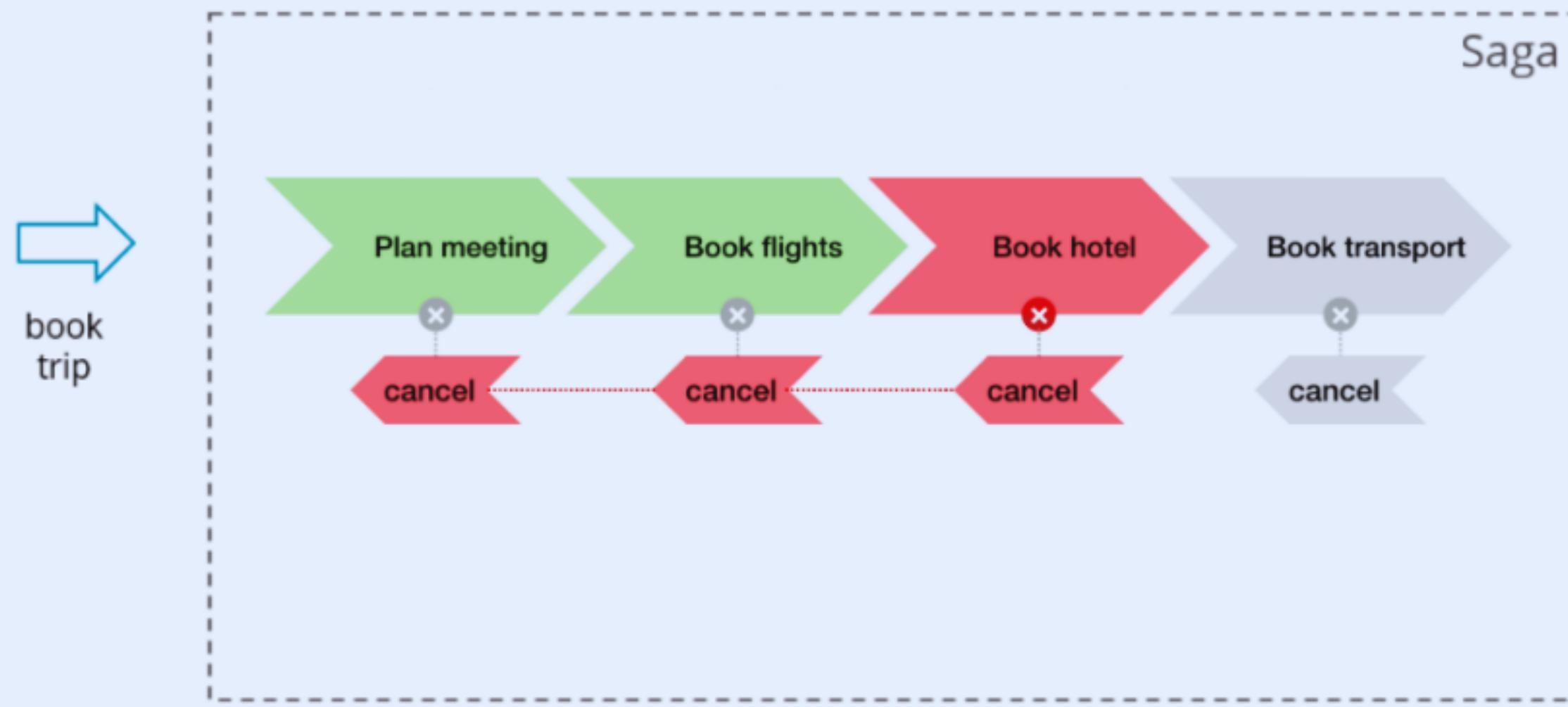
- Transformação de comando de evento:
 - Por definição, um evento visa informá-lo sobre um fato relevante que ocorreu e que algum outro serviço pode estar interessado. Mas, no momento em que exigimos que um serviço acompanhe um evento, usamos esse evento como se tivesse o significado semântico de um comando. A consequência disso: acabamos com um acoplamento mais apertado do que o necessário.



SAGAS

- Uma saga é uma sequência de transações locais. Cada serviço em uma saga realiza sua própria transação e publica um evento. Os outros serviços ouvem esse evento e executam a próxima transação local. Se uma transação falhar por algum motivo, a saga também executa transações de compensação para desfazer o impacto das transações anteriores.

A **Saga** represents a single **business process**



SAGAS

- Quando um usuário faz um pedido em um serviço de delivery, pode existir esta sequência de ações:
 - O serviço de pedidos de comida cria um pedido. Neste ponto, o pedido está em um estado PENDENTE. Uma saga gerencia a cadeia de eventos.
 - A saga entra em contato com o restaurante através do serviço de restaurante.
 - O serviço de restaurante tenta fazer o pedido no restaurante escolhido. Depois de receber uma confirmação, ele envia uma resposta.

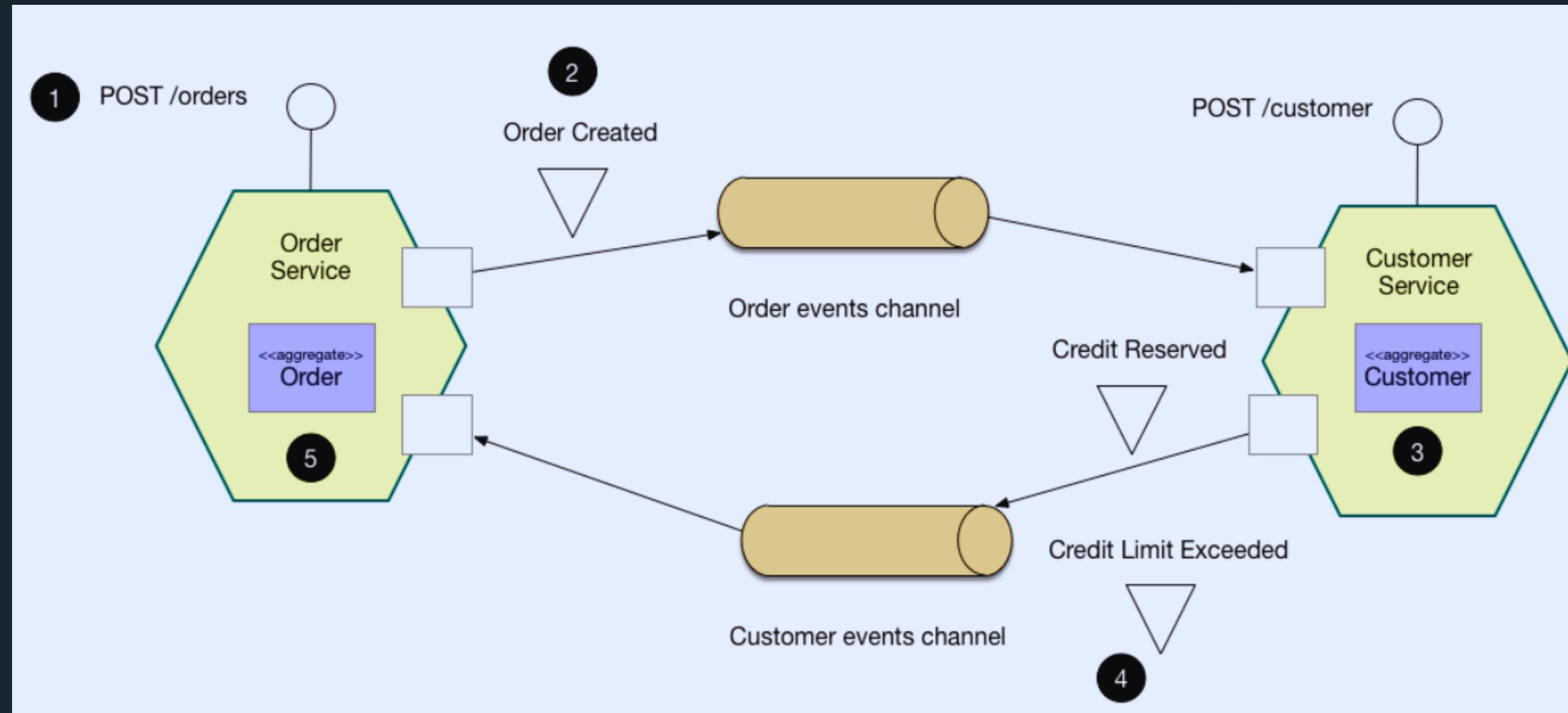
SAGAS

- A saga recebe a resposta. E, dependendo da resposta, ele pode aprovar o pedido ou rejeitá-lo.
- O serviço de pedido de comida altera o estado do pedido. Se o pedido fosse aprovado, informaria o cliente com os próximos detalhes. Se rejeitado, também informará o cliente com uma mensagem de desculpas.

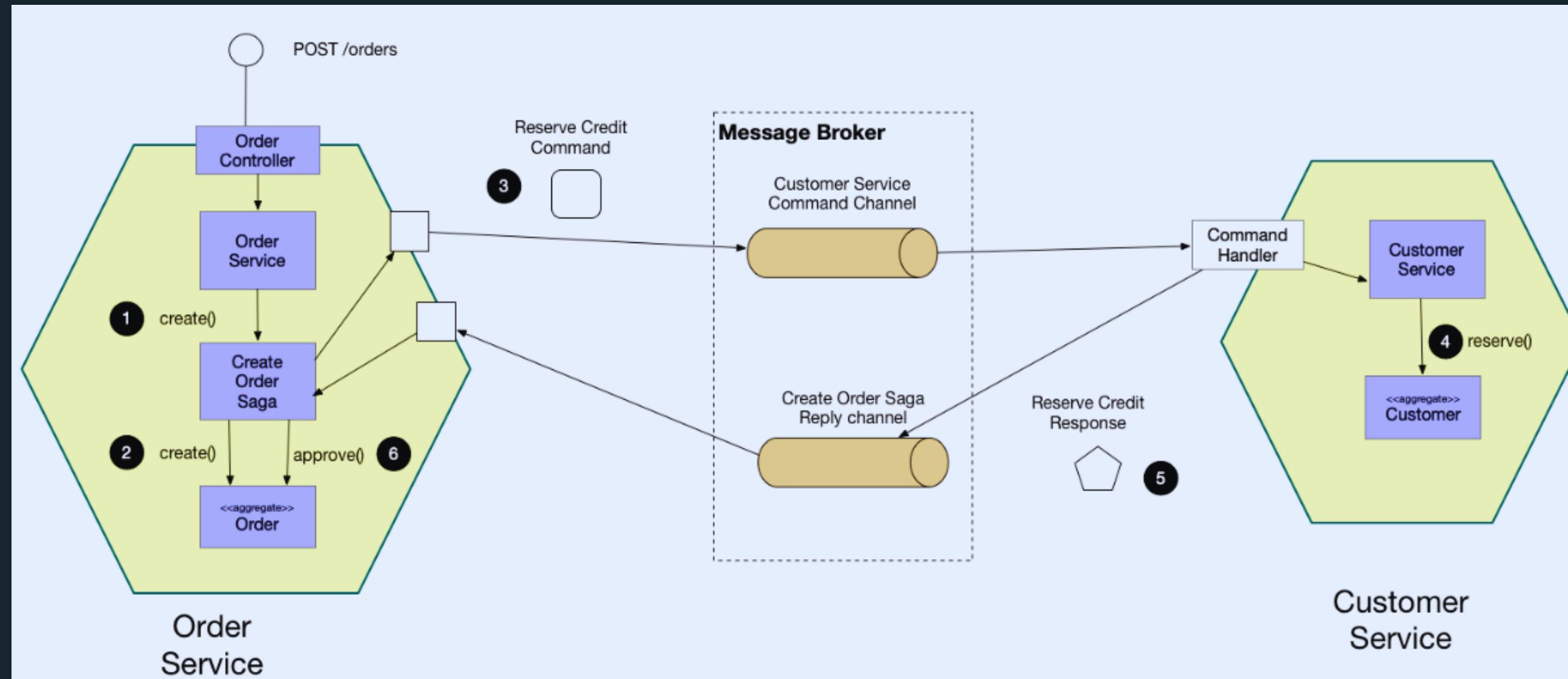
TIPOS

- Saga baseada em Orquestração:
 - Nessa abordagem, existe um orquestrador Saga que gerencia todas as transações e direciona os serviços participantes para executar transações locais com base em eventos. Este orquestrador também pode ser considerado um SAGA MANAGER.
- Saga baseada em Coreografia:
 - Nesta abordagem, não há orquestrador central. Cada serviço participante da Saga realiza suas transações e publica eventos. Os outros serviços atuam sobre esses eventos e realizam suas transações. Além disso, eles podem ou não publicar outros eventos com base na situação de negócio.

Baseado em Coreografia



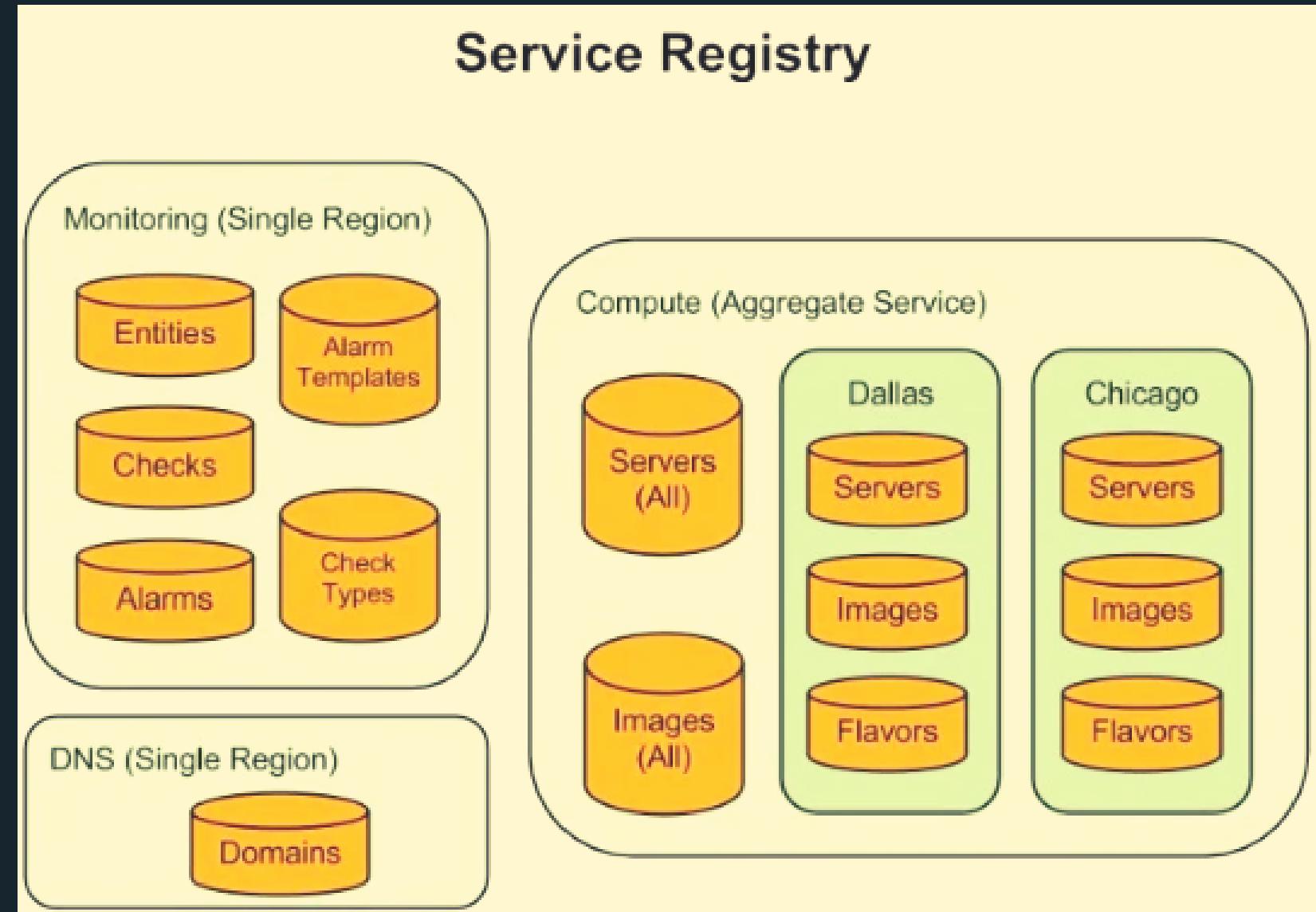
Baseado em Orquestração



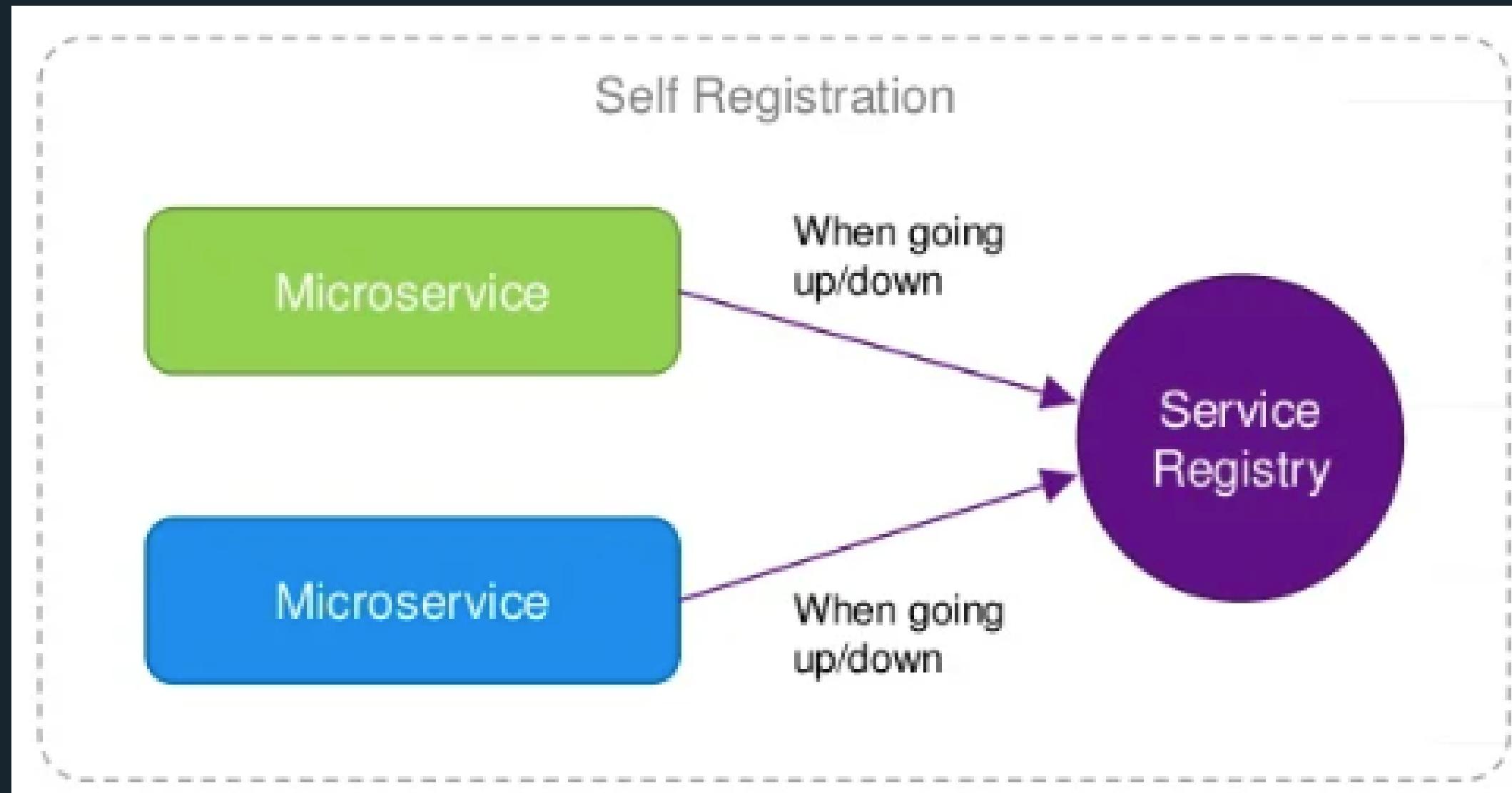
REGISTRO DE SERVIÇOS



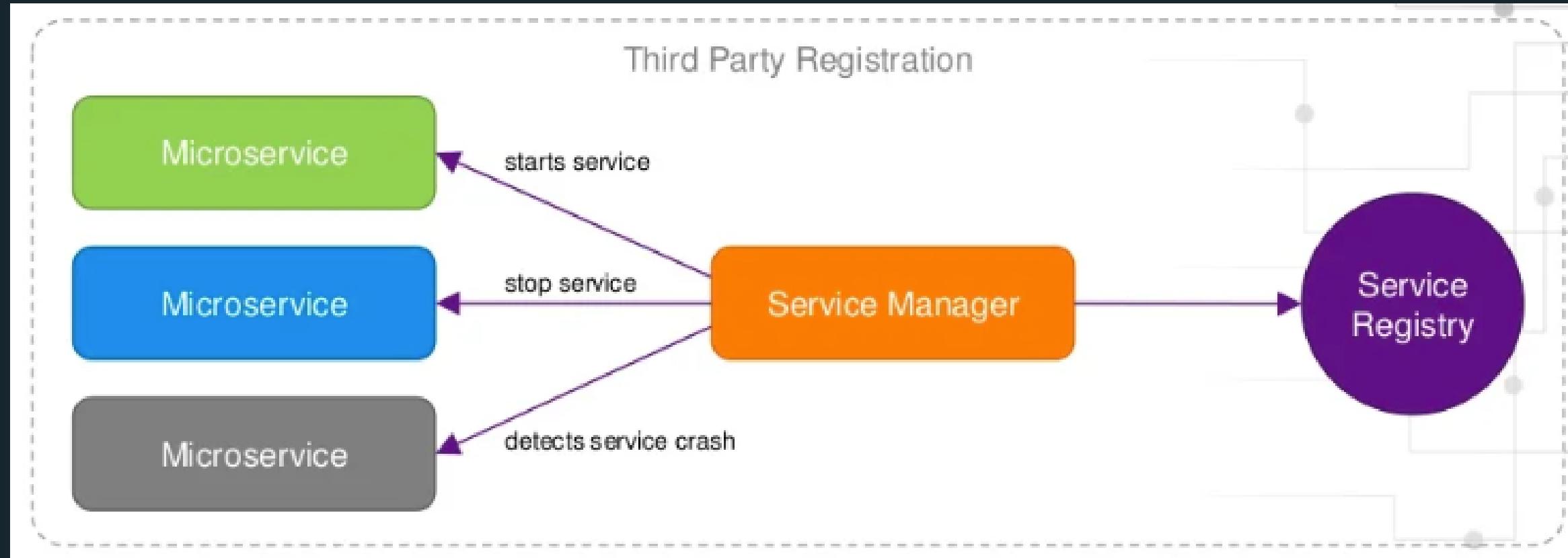
Registro de serviços



Auto-registro



Registro por terceiros



DESCOBERTA DE SERVIÇOS



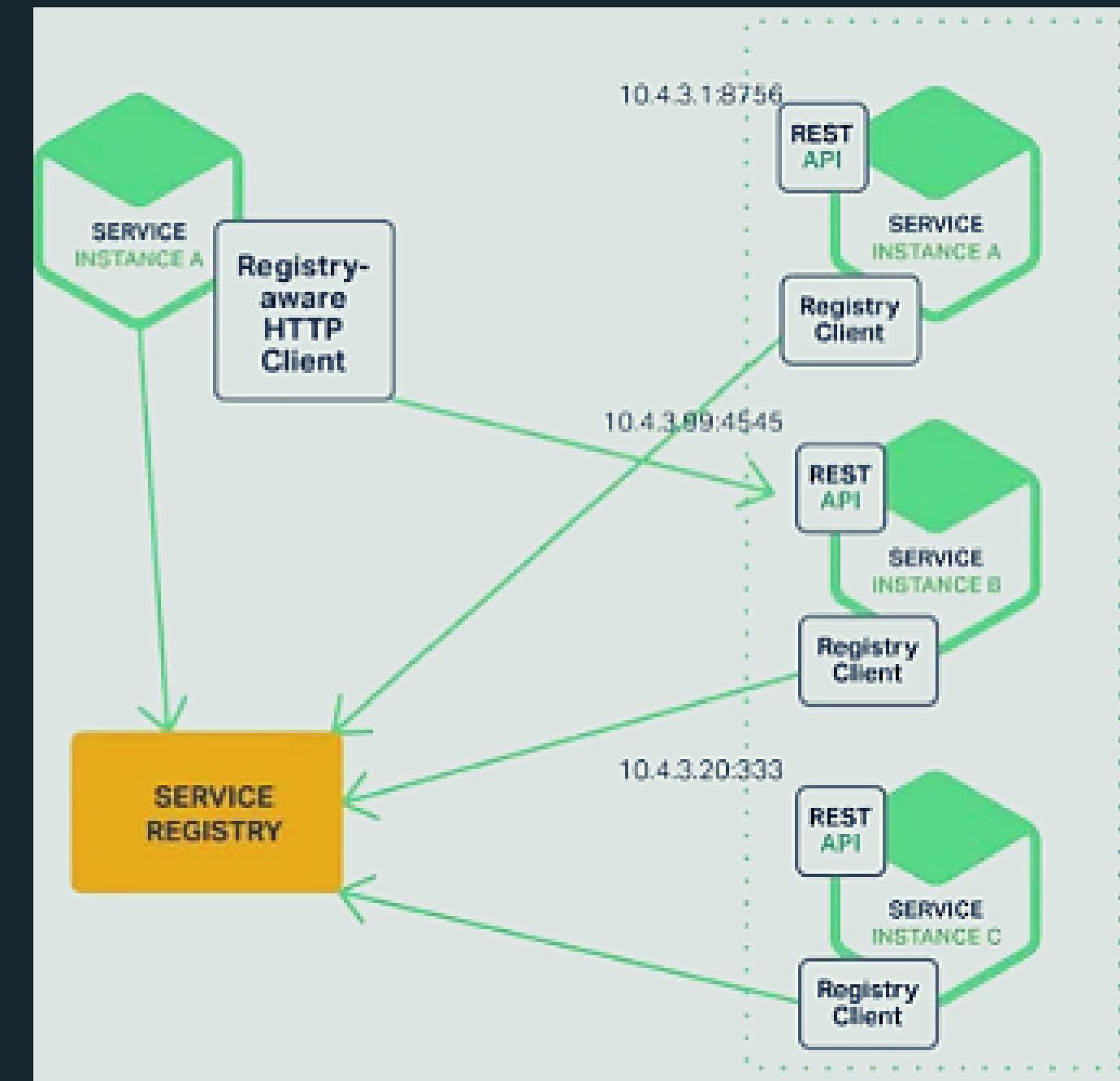
**“Como o serviço A fala
com o serviço B sem ter
nenhum conhecimento
sobre onde encontrar o
serviço B?”**

E se tivermos 10 instâncias do
Serviço B
rodando em um número arbitrário
de nós de cluster

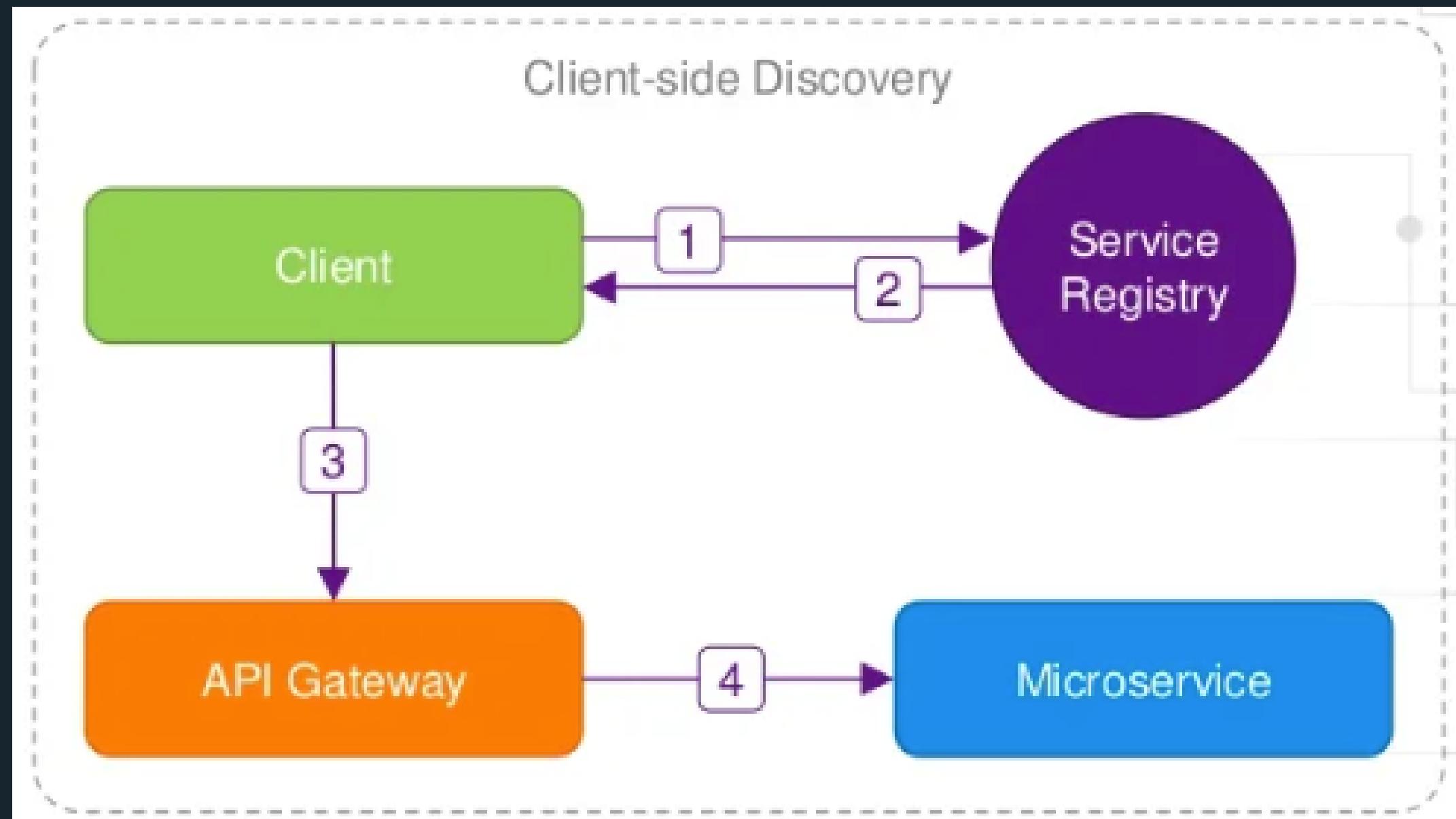


Descoberta de Serviços

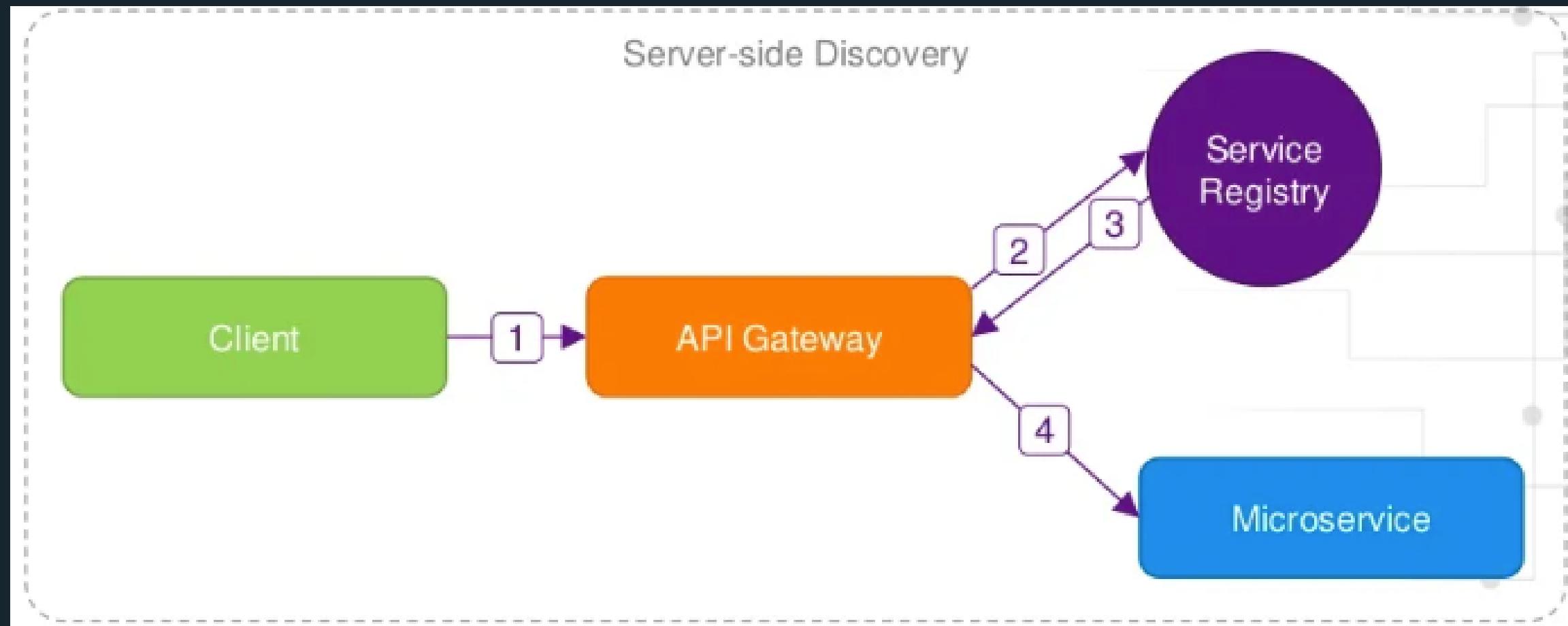
EXEMPLO



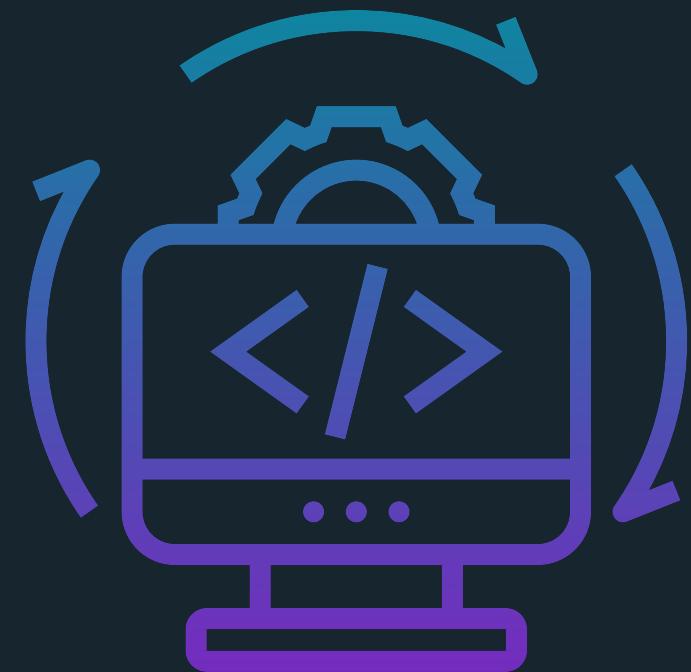
Descoberta via padrão Client-Side



Descoberta via padrão Server-Side



INTEGRAÇÃO E DEPLOY CONTÍNUO



**INTEGRAÇÃO DE CÓDIGO É UM
PROBLEMA EM SOFTWARE, ONDE
O TEMPO SE MOVE LINEARMENTE,
MAS A DOR SE MOVE
EXPONENCIALMENTE: QUANTO
MAIS VOCÊ DEMORA, PIOR O
PROBLEMA SE TORNA**

NEAL FORD - DIRETOR DA THOUGHTWORKS



Integração Contínua

- Uma prática de desenvolvimento de software onde os membros do time de desenvolvimento integram seu trabalho constantemente;
- Cada integração é feita automaticamente por um processo para detectar falhas rapidamente;
- Reduz drasticamente problemas de integração e possibilita o desenvolvimento de um software seguro e coeso;

Integração Contínua

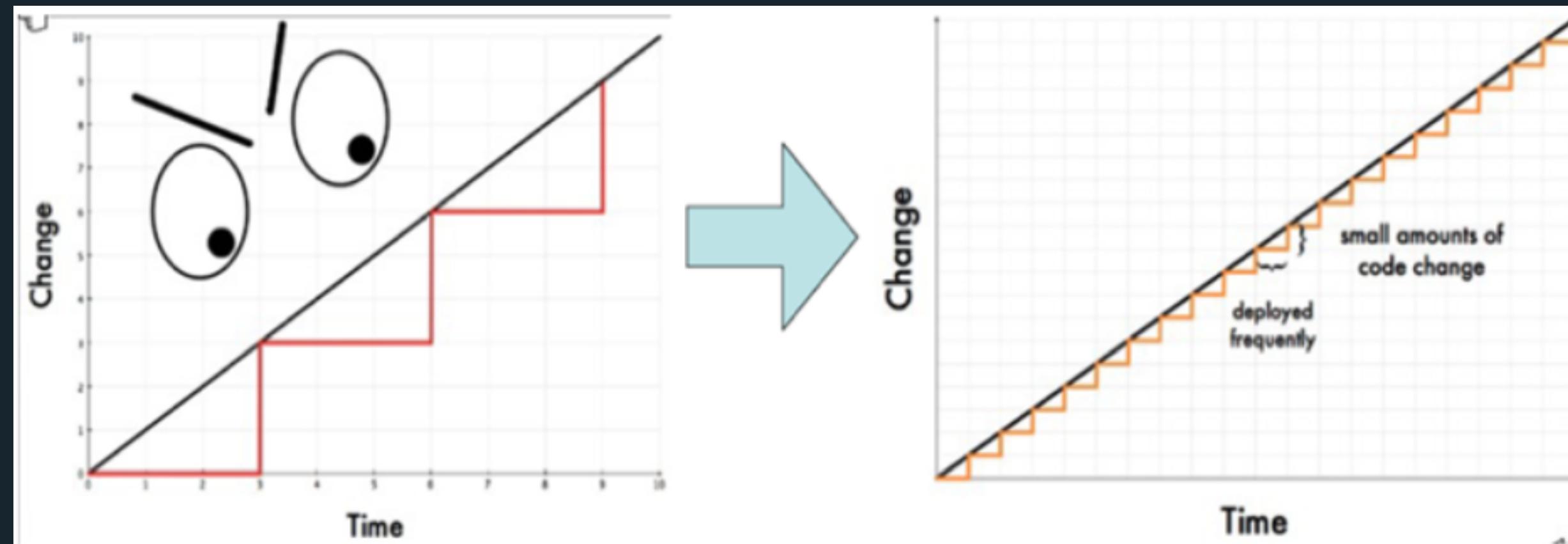
- Escreva testes automatizados para desenvolvedores;
- Execute compilações privadas;
- Confirme código com frequência;
- Não confirme código com defeito;
- Evite obter código com defeito;
- Corrija construções com defeito imediatamente;
- Todos os testes e inspeções devem passar;

Deploy Contínuo

- Quão frequentes você entrega software em produção para os seus usuários?
 - Mais de uma vez por dia?
 - Mais de uma vez por semana?
 - Mais de uma vez por mês?
 - Mais de uma vez por ano?
 - Uma vez por ano ou menos?

Deploy Contínuo

- Releases entregue em produção de forma frequente:
- Medida real de progresso
- Feedback do usuário
- Reduz consideravelmente o risco de release

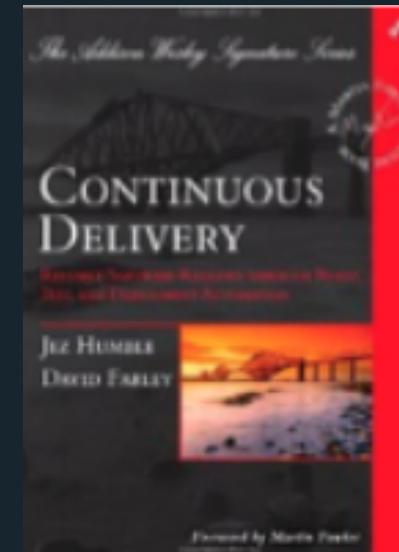


Deploy Contínuo

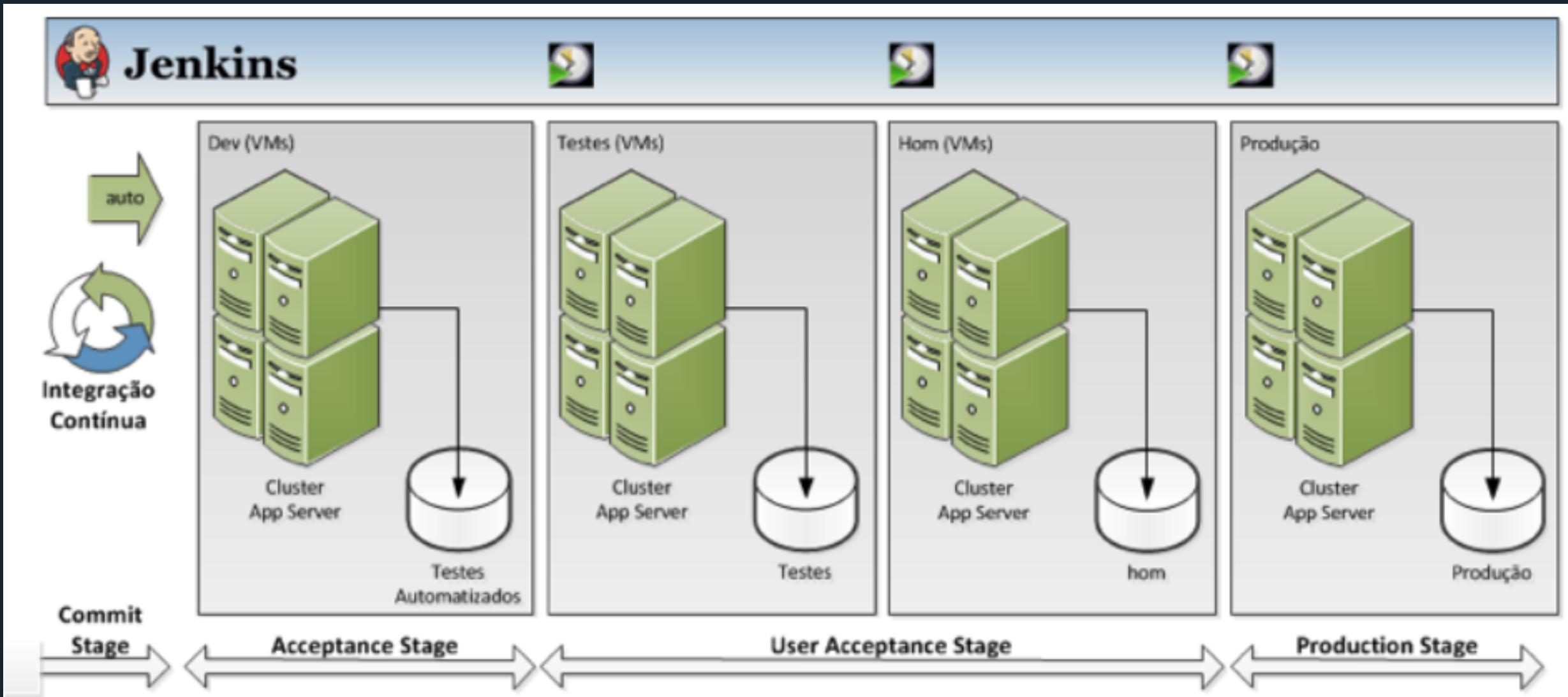
- Princípios:
 - Cada commit gera um release candidate;
 - Automatize tudo que pode ser automatizado;
 - If it hurts, do it more often and bring the pain forward”;
 - A qualidade interna é obrigatória;
 - Testes automatizados são essenciais;
 - Todo mundo é responsável pela release;
 - Pronto significa released;
 - Melhoria Contínua(Continuous Improvement);

**“CONTINUOUS DELIVERY DEIXOU
DE SER UM DIFERENCIAL
TECNOLÓGICO PARA SER UMA
NECESSIDADE DENTRO DAS
ORGANIZAÇÕES”**

JEZZ HUMBLE



Deploy Contínuo



Build Pipeline

- Commit Stage:
 - Poll SCM;
 - Compilação;
 - Cobertura dos testes;
 - Unitários e integração;
 - Qualidade do código fonte;
 - Conformidades com padrões empresariais;
 - Possíveis bugs;
 - Boas práticas;
 - Empacotamento e publicação;

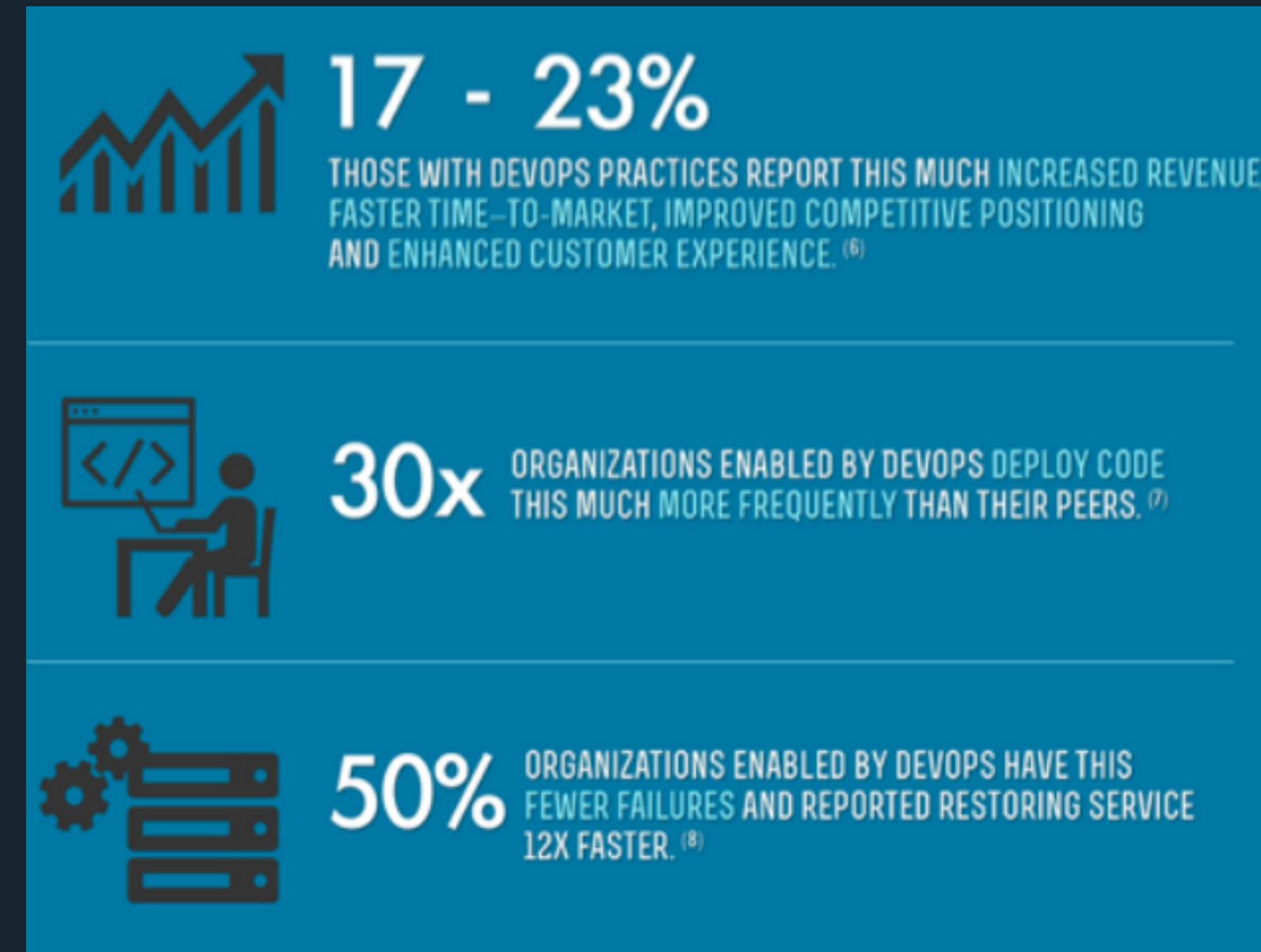
Build Pipeline

- Acceptance Stage:
 - Testes automatizados;
 - Testes funcionais;
 - Testes de aceitação;
 - Testes de performance;
 - Testes de segurança;

Considerações

- Automatização + testes = sucesso;
- Reduza os riscos com releases incrementais;
- Existem inúmeras formas de se construir um build pipeline. Descubra a sua;
- Cultura de DevOps é necessária;
- Prepare-se: quanto tempo você demora para reconstruir o seu ambiente a partir do zero?
- Dificuldades:
 - Exige mudança cultural;
 - Maior dificuldade encontrada nas implantações de Continuous Delivery;
 - Exige Ciclos Curtos;
 - Empresas organizadas em silos/áreas;
 - Requer envolvimento de todos;

Considerações





OBRIGADO



<https://www.linkedin.com/in/vinicio Durantisoares/>



<https://www.instagram.com/vinicio.d.soares/>

PUCRS online  uol edtech