

Estilos Arquiteturais

Estilos arquiteturais e padrões arquiteturais



Estilos arquiteturais

São abordagens de design amplas que definem a estrutura e a organização de sistemas de software. Fornecem uma visão geral da estrutura de um sistema e como seus componentes interagem.

Eles descrevem uma estrutura de alto nível para organizar os componentes de um sistema, suas interações e as restrições que guiam o desenvolvimento e a evolução do sistema.

- Estrutura global
- Componentes e conexões
- Escopo amplo

Quais os benefícios?

Reutilização: Permitem a reutilização de soluções testadas e comprovadas para problemas recorrentes, aumentando a eficiência do desenvolvimento.

Compreensão e Comunicação: Eles fornecem uma linguagem comum para descrever e discutir a arquitetura de um sistema.

Flexibilidade e Escalabilidade: A aplicação de estilos arquiteturais pode resultar em sistemas que são mais fáceis de entender, modificar e escalar.

Padronização: Eles ajudam a padronizar o desenvolvimento de software em uma organização, promovendo consistência e qualidade.

Facilidade de Manutenção: Quando bem definidos, podem facilitar a manutenção e a evolução do sistema, pois fornecem uma estrutura clara e organizada para o código e os componentes do sistema.



Tipos de estilos

Client-Server

Shaw e Garlan descrevem o estilo arquitetural client-server como um paradigma que distribui funcionalidades entre servidores que fornecem serviços e clientes que consomem esses serviços.

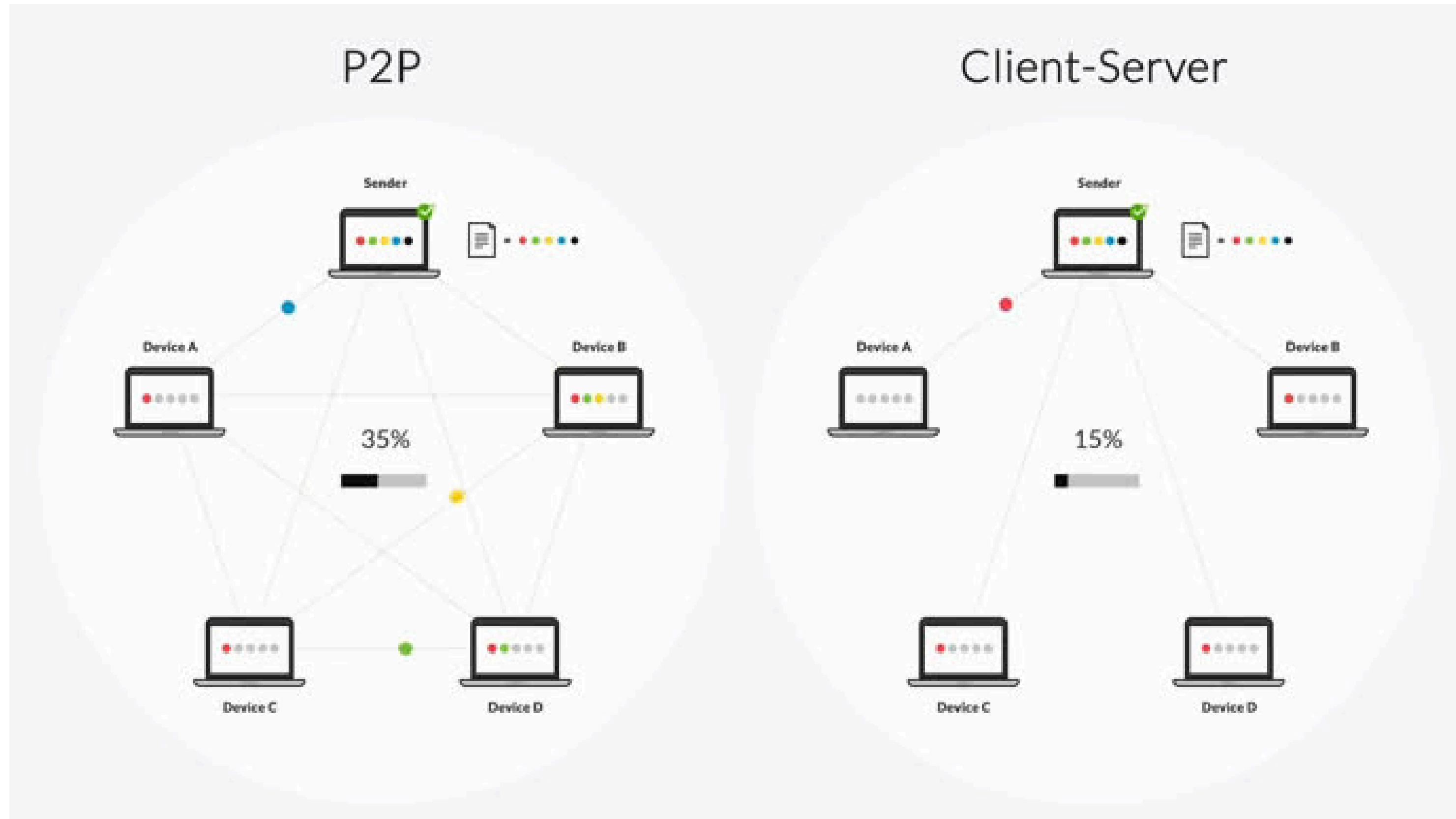
Thin Client vs. Fat Client: Em um modelo de thin client, a maior parte do processamento e armazenamento é feita no servidor, com o cliente focando apenas na interface de usuário. Em um modelo de fat client, o cliente também executa uma parte significativa da lógica de aplicação.

Contextos de Aplicação: Aplicações web e sistemas de banco de dados são exemplos típicos de arquiteturas client-server.

Normalmente combinado com outros estilos, Component-based, layered e utilizado em padrões arquiteturais como MVC.



Client-Server



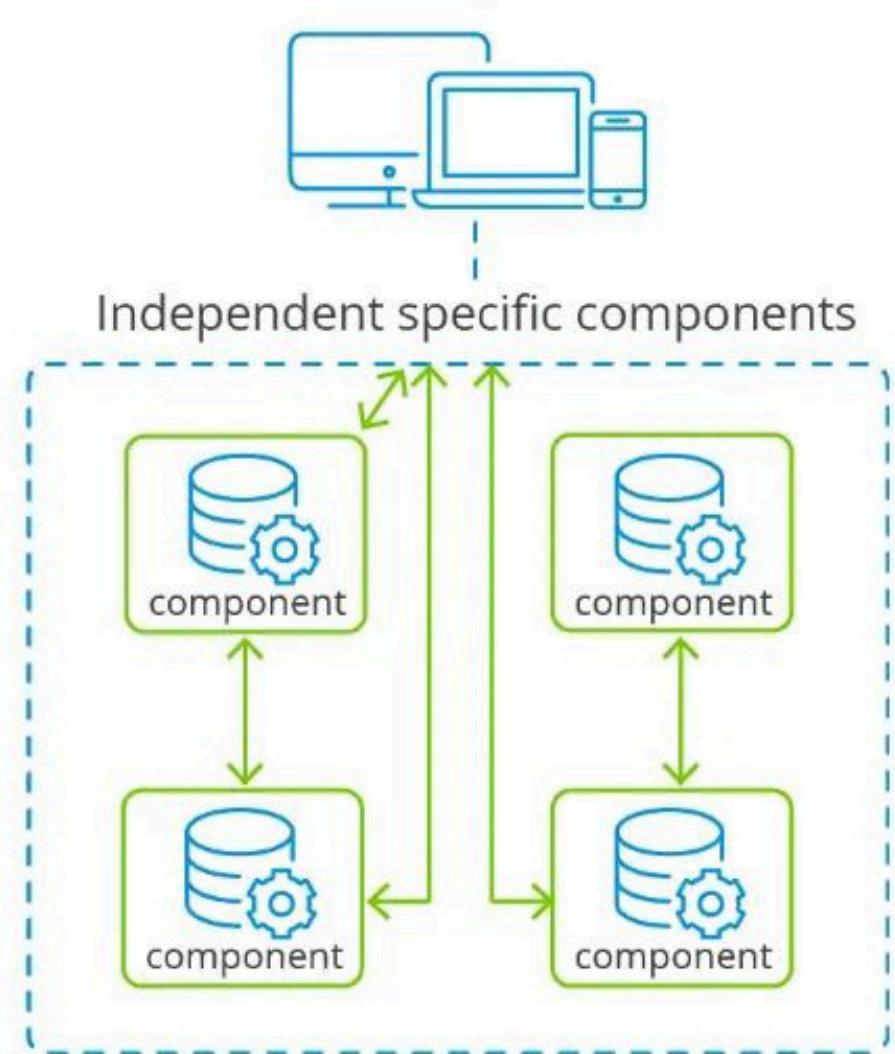
Component-based

Fowler descreve Component-Based Architecture como uma abordagem para organizar sistemas de software em componentes reutilizáveis, que podem ser desenvolvidos e implantados de forma independente.

Principais benefícios:

Reutilização: Componentes bem definidos podem ser reutilizados em diferentes sistemas, reduzindo o tempo de desenvolvimento e os custos.

Facilidade de Manutenção: A modularidade dos componentes facilita a manutenção e a evolução do sistema, permitindo que componentes individuais sejam atualizados ou substituídos sem afetar o restante do sistema.



Exemplos de Component-based

Exemplo:

- Sistema de gestão empresarial (ERP) - Monólito modular
- Plataforma de comércio eletrônico - Magento, Shopify.
- Sistema de gerenciamento de conteúdo(CMS) - WordPress, Drupal.
- Componentes em aplicações web
- E Microserviços?

No livro "Building Microservices: Designing Fine-Grained Systems", Sam Newman argumenta que os microserviços são uma extensão natural dos princípios de Component-Based Architecture, aplicada ao desenvolvimento de sistemas distribuídos modernos.

Citação: "Microservices are a form of component-based architecture where the components are services that are independently deployable and scalable."

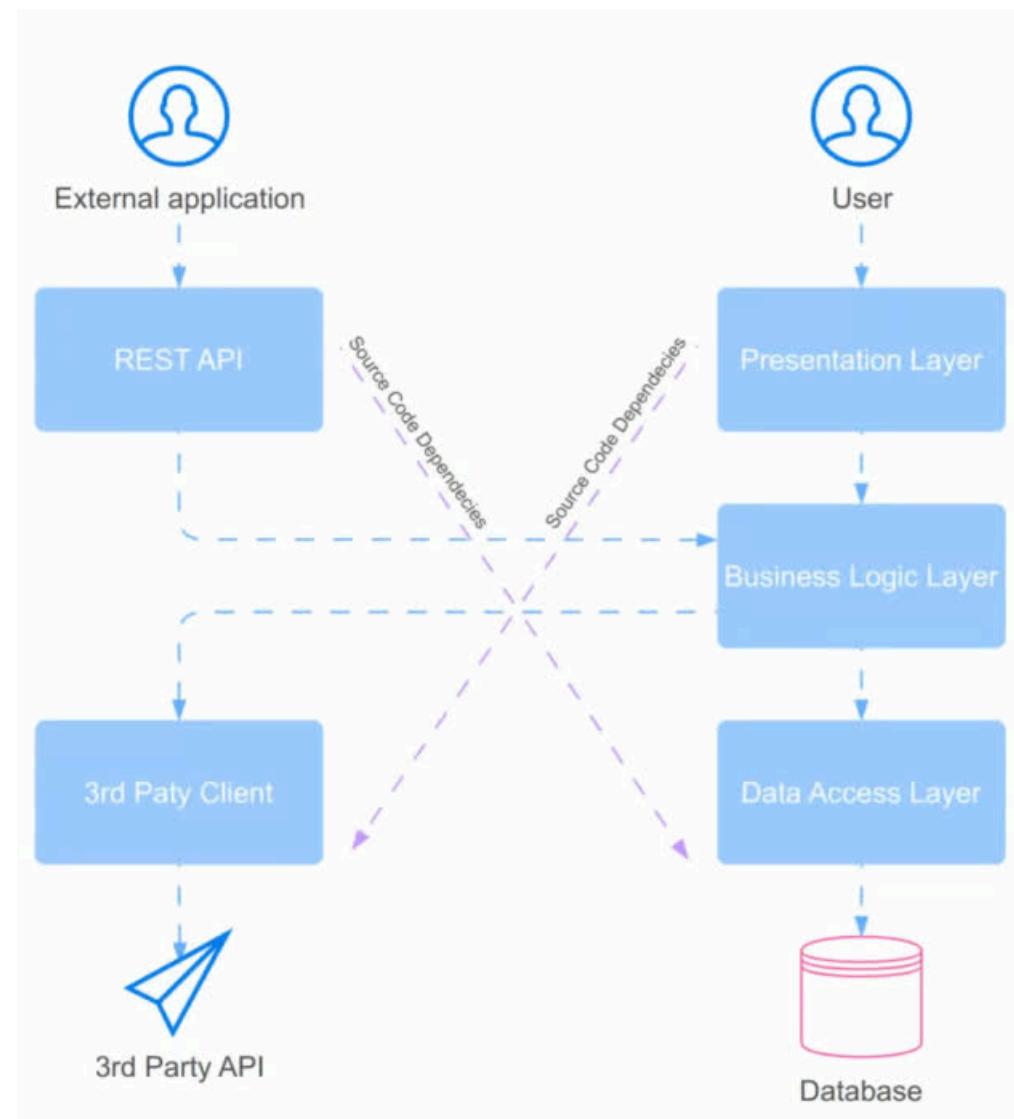


Layered & N-tiers



Layered

É uma abordagem onde o sistema é organizado em camadas, cada uma com responsabilidades específicas e interações bem definidas. Este estilo promove a separação de preocupações, facilitando o desenvolvimento, a manutenção e a evolução do software.



Fowler descreve a arquitetura em camadas como um padrão estrutural comum em aplicações empresariais, dividindo a aplicação em camadas como apresentação, aplicação, domínio e infraestrutura. Ele defende padrões específicos que se encaixam na arquitetura em camadas, como "Presentation Layer," "Domain Layer" e "Data Source Layer".

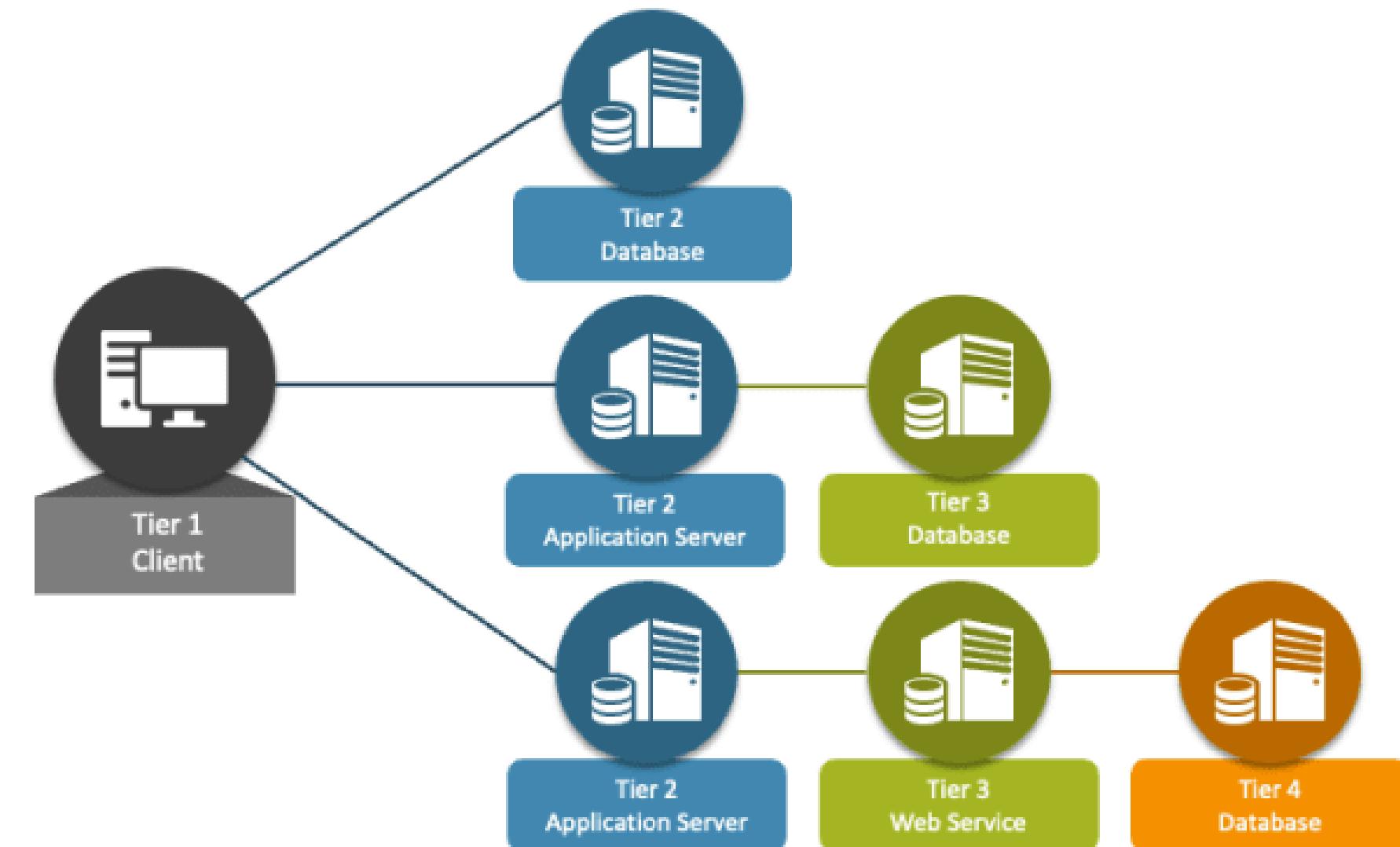
Em resumo, vai possuir uma interação com o usuário, seja ela qual for, vai possuir um domínio e vai ter uma camada de acesso a dados.

N-tiers

Refere-se à separação física das diferentes camadas da aplicação, onde cada nível pode ser distribuído em diferentes máquinas ou servidores. Esse estilo é focado na distribuição e escalabilidade do sistema.

A separação física permite que cada nível seja escalado de forma independente, aumentando a capacidade do sistema de lidar com uma maior carga de trabalho.

N-TIER ARCHITECTURE



Tier, layer?

Em discussões de arquitetura de três camadas(tier), o nível é frequentemente usado de forma intercambiável e equivocadamente pela camada, como um 'nível de apresentação' ou 'nível lógico de negócios'. Não são o mesmo!

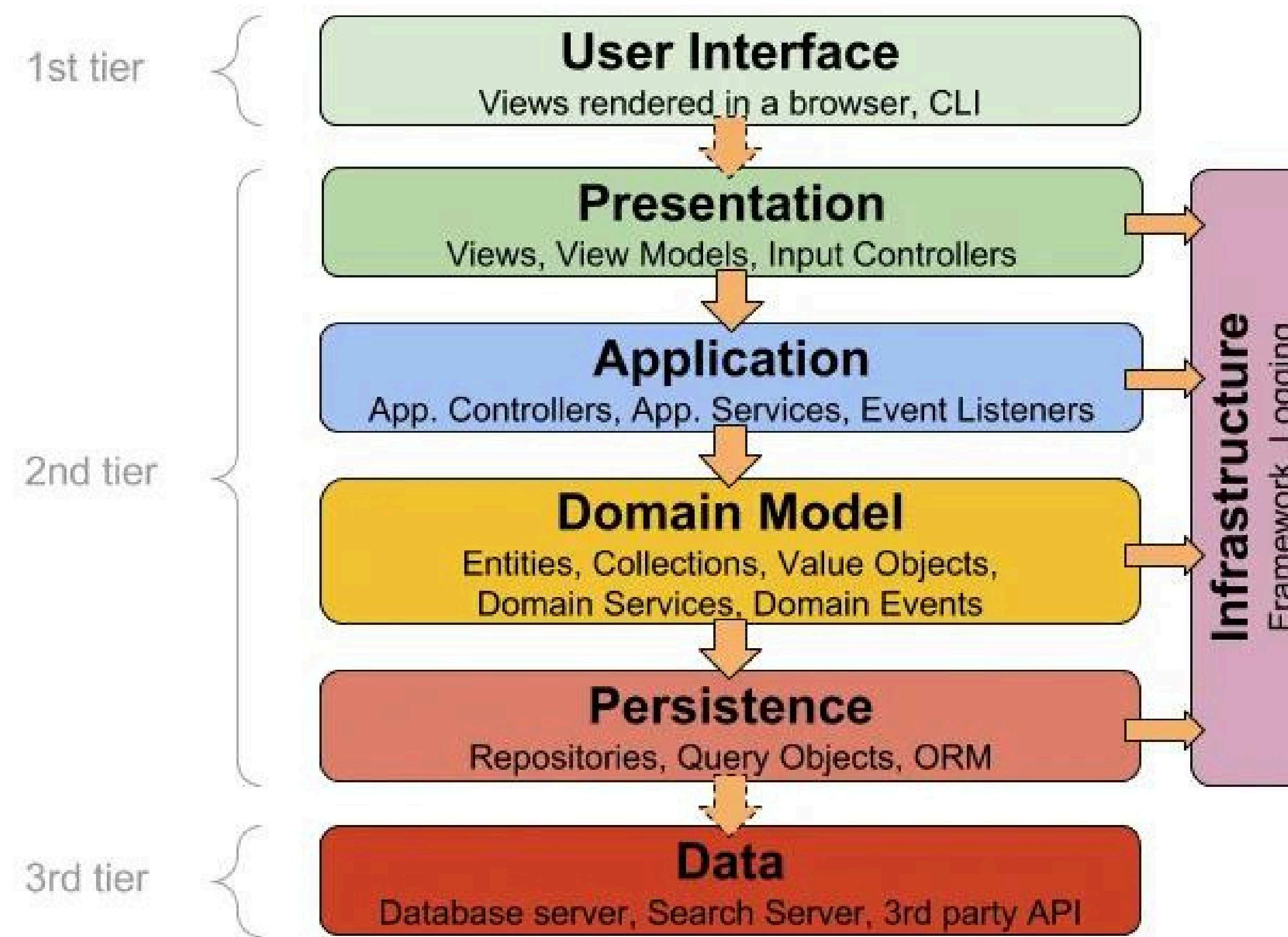
Tier(Nível)

Refere-se normalmente a uma divisão física do software executada em infraestrutura separada das outras divisões.

Layer(Camada)

Se refere a uma divisão lógica da sua aplicação, sendo normalmente executada em uma única infra e processos distintos.

Por exemplo, o aplicativo Contatos em seu telefone é um aplicativo de três camadas(layers), mas de nível único, porque as três camadas são executadas em seu telefone.





Event-driven architecture

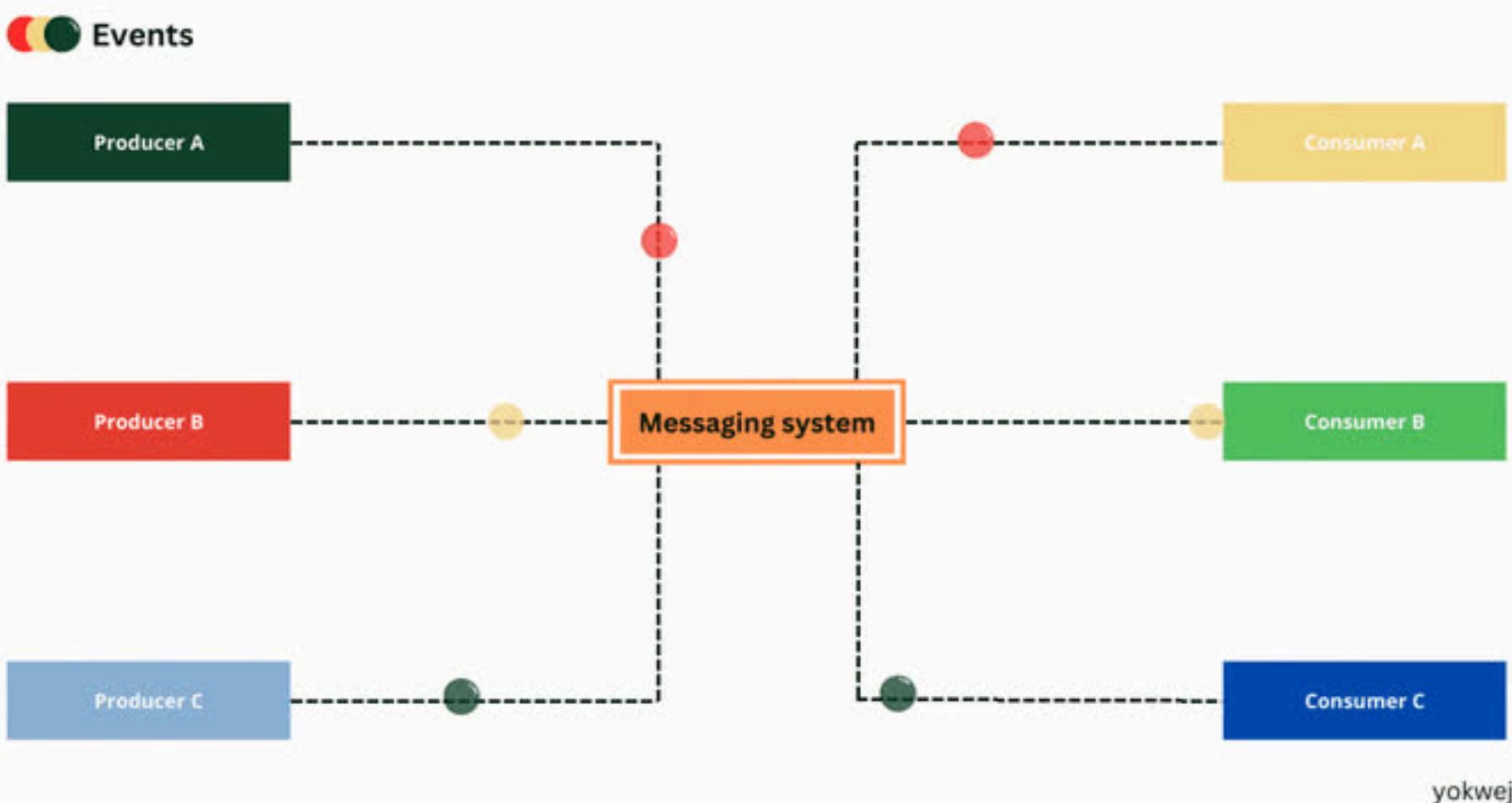
Event-driven Architecture(EDA)

É um estilo arquitetural em que os componentes do sistema se comunicam através da geração e manipulação de eventos. Um evento é uma mudança significativa de estado ou uma ocorrência que pode ser capturada e respondida por outros componentes do sistema.

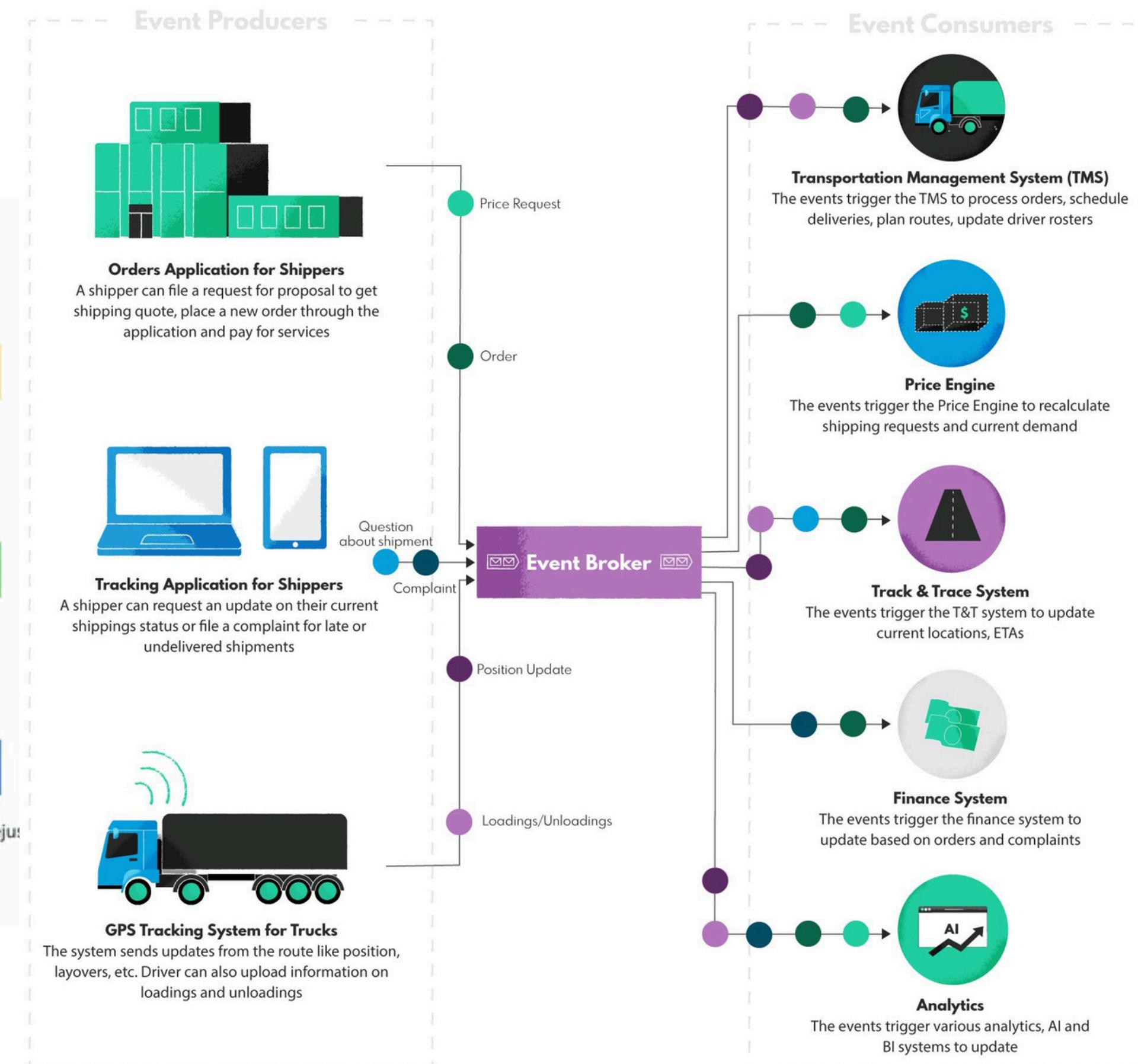
- Event producers (Publicadores): Geram e emitem eventos.
- Event consumers (Assinantes): Recebem e processam eventos.
- Event channel (Broker de Mensagens): Middleware que gerencia a distribuição de eventos dos publicadores para os assinantes.
- Event processors: Componentes que processam eventos e podem gerar novos eventos.

Exemplo: 10 exemplos de utilização de EDA no setor de logística.

E na prática?



Exemplo real?



Microkernel

Estilo arquitetural que busca manter o núcleo do sistema o mais simples e reduzido possível, delegando a maioria das funcionalidades para módulos ou serviços externos adicionados ao núcleo.

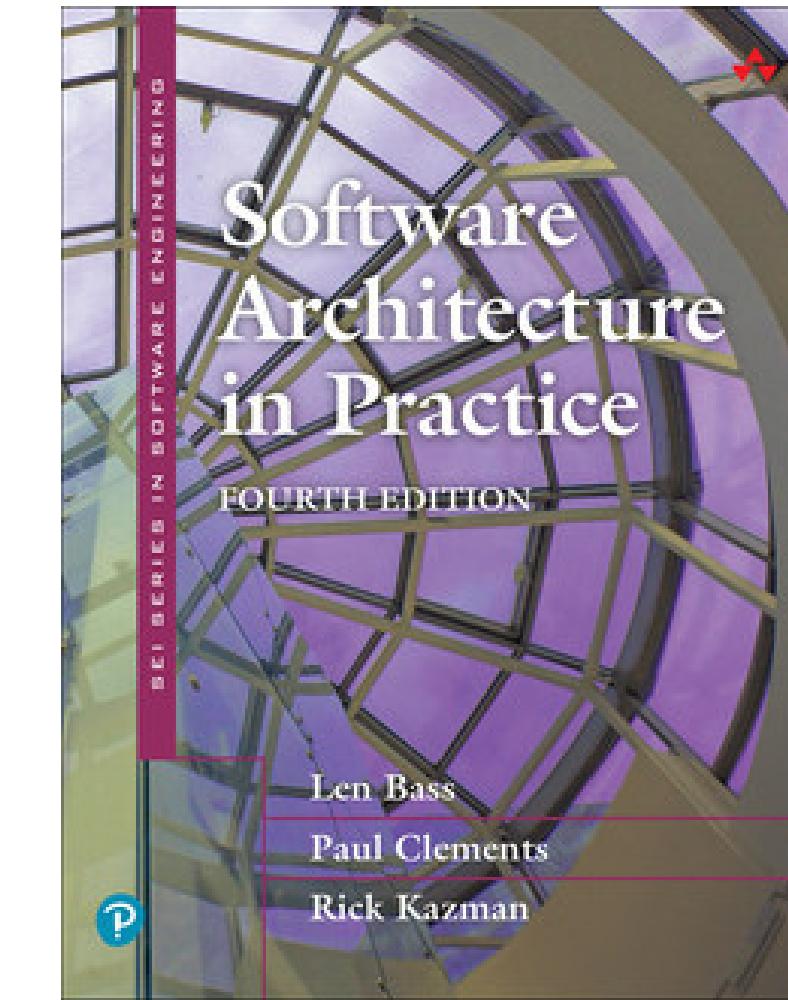
Características:

- Núcleo Mínimo: Apenas funcionalidades essenciais
- Serviços e Módulos: Funcionalidades adicionais
- Comunicação: Normalmente feita por meio de chamadas de sistema ou mensagens

Exemplos:

Sistemas Operacionais: Minix, QNX.
Frameworks: Eclipse IDE.

"The microkernel architecture style involves designing a minimal core system with additional features provided by separate modules or services. This approach is considered a style because it defines a high-level organization of components and their interactions, rather than a specific implementation pattern."



(Software Architecture in Practice, p. 207)

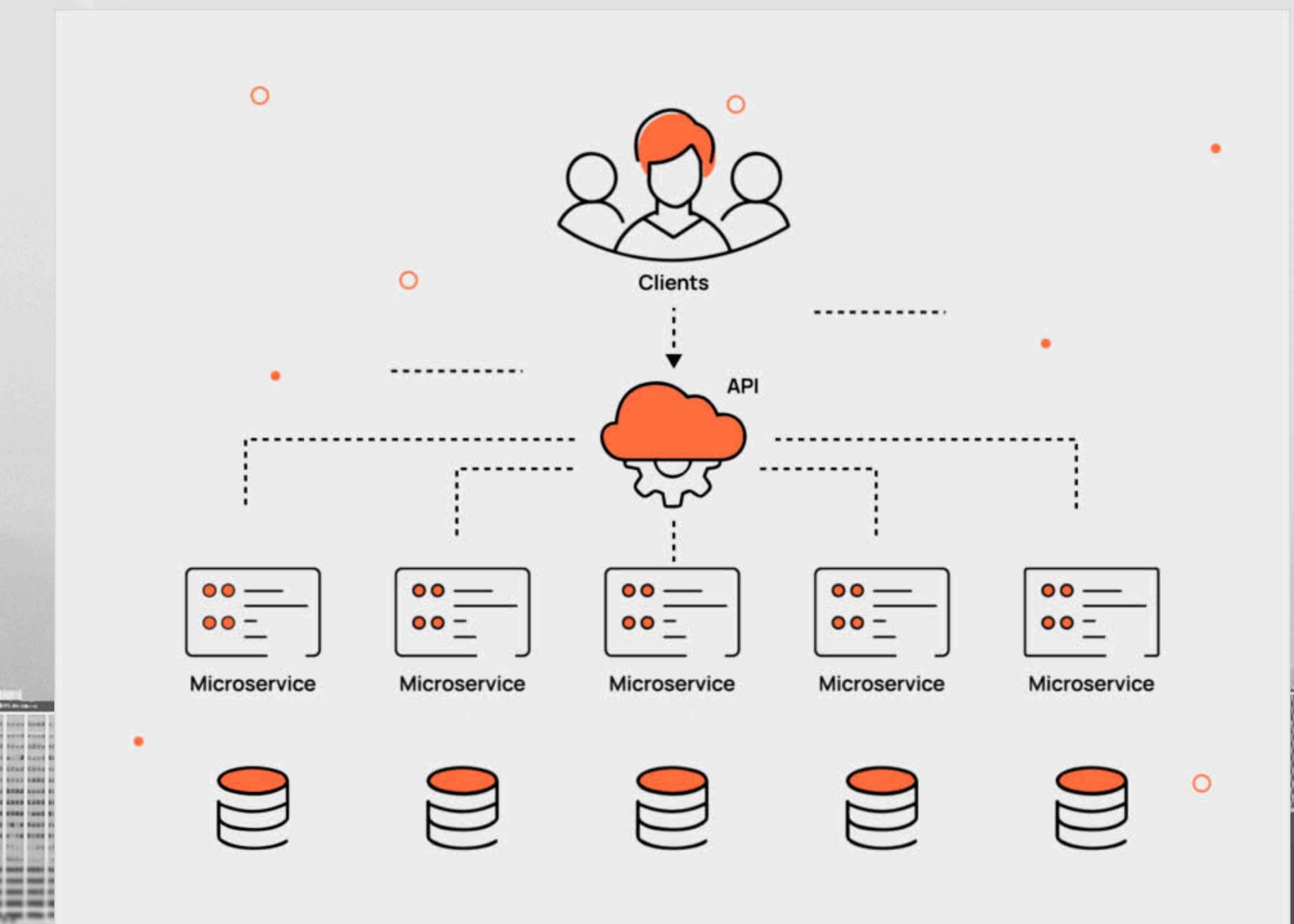


Microservices

É um estilo arquitetural que estrutura uma aplicação como um conjunto de pequenos serviços independentes, cada um executando um processo único e comunicando-se por meio de APIs bem definidas. Cada microserviço é desenvolvido, implantado e escalado de forma independente, permitindo uma maior flexibilidade e agilidade no desenvolvimento e na manutenção de software.

Características:

- Desenvolvimento independente
- Desenvolvimento poliglota
- Desempenho e escalabilidade
- Descentralização



Trade-offs

- Complexidade do gerenciamento
- Latência na comunicação entre microserviços
- Consistência de dados
- Custos operacionais e segurança

Repo utilizando microservices

4 exemplos de microservice:
Amazon, Netflix, Uber e Etsy



Pipes and Filters: Organiza o sistema como uma série de filtros conectados por pipes (tubos). Cada filtro é responsável por processar dados, enquanto os pipes transportam os dados entre os filtros. Este estilo é ideal para sistemas que processam dados em etapas sequenciais, como pipelines de processamento de dados. Exemplo clássico é o shell do linux:

```
$ ls | grep b | sort -r | tee arquivo.out | wc -l
```

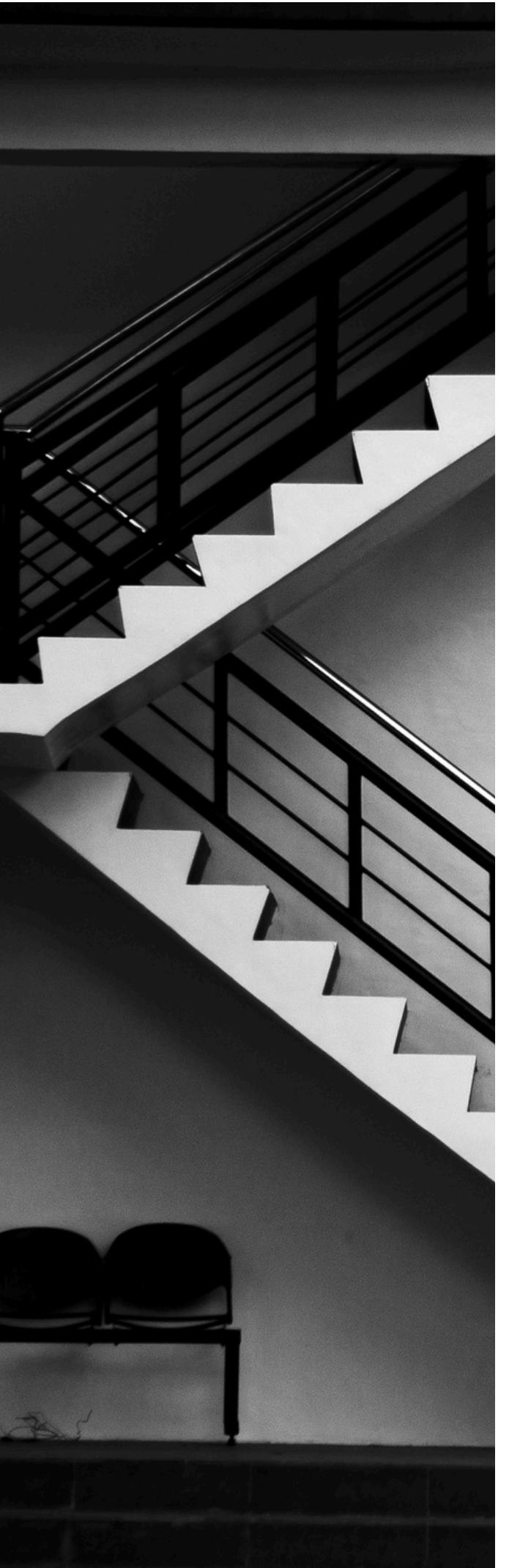
Service-Oriented Architecture (soA): Estilo arquitetural que organiza o sistema como um conjunto de serviços independentes que se comunicam entre si por meio de interfaces bem definidas. Cada serviço é responsável por uma parte específica da funcionalidade do sistema e pode ser acessado remotamente por outros serviços ou aplicações. Possuem o mesmo banco e são uma parte completa do domínio.

Representational State Transfer (REST): É um estilo arquitetural para sistemas distribuídos que utiliza o protocolo HTTP e opera sobre os conceitos de recursos e representações.

Em REST, os recursos são identificados por URLs e manipulados usando métodos HTTP.



Clean Arch & Hexagonal



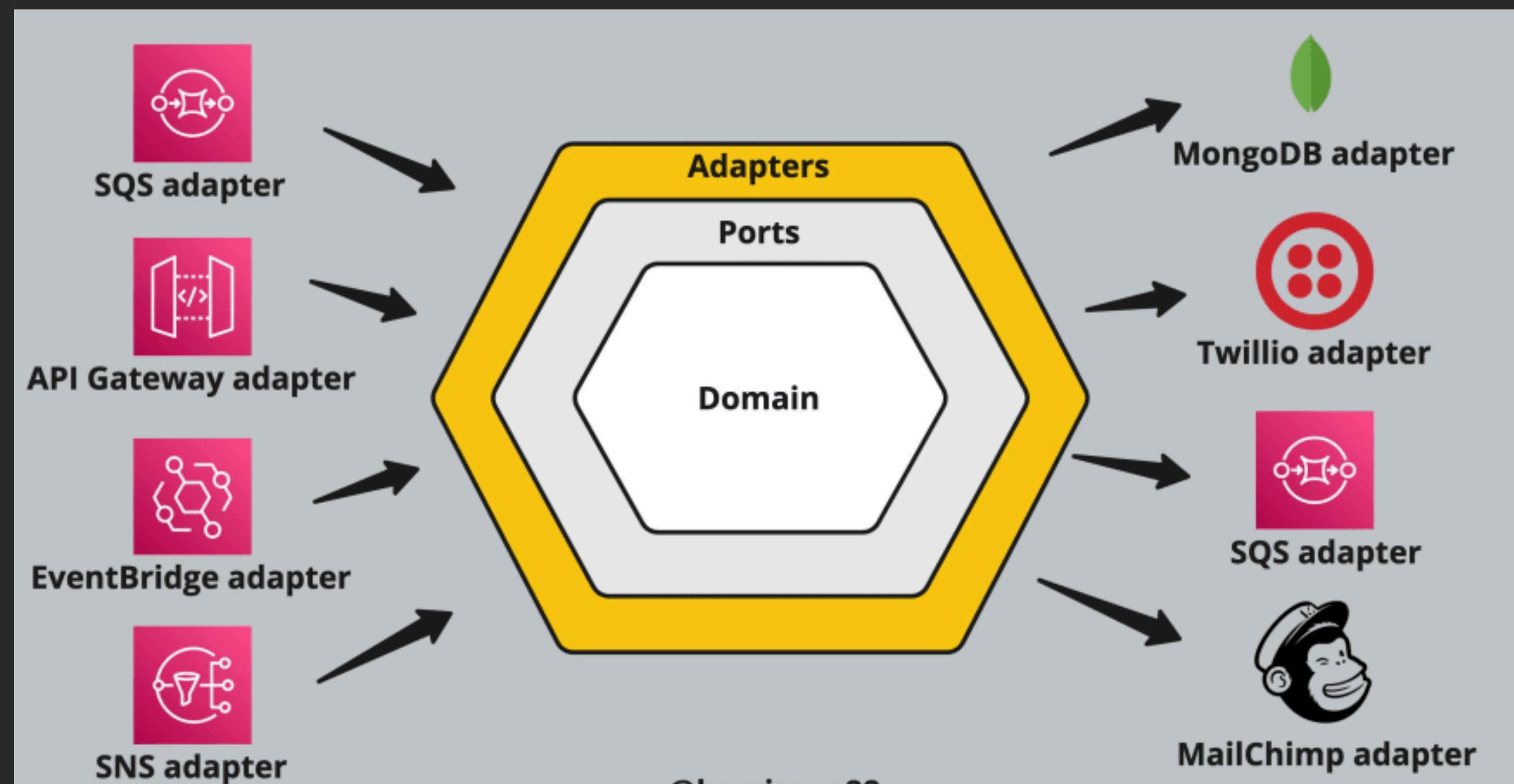
Arquitetura Limpa

Este estilo de arquitetura enfatiza a separação de preocupações e a independência dos frameworks, interfaces de usuário, bancos de dados e outros detalhes externos. Podemos considerar como um avanço da onion architecture.

"The Clean Architecture is an architectural style that can be applied to a wide range of applications. It emphasizes the separation of concerns, making sure that the business rules are isolated from frameworks, user interfaces, and external agencies." (Clean Architecture: A Craftsman's Guide to Software Structure and Design, p. 18).

Hexagonal-Ports and Adapters

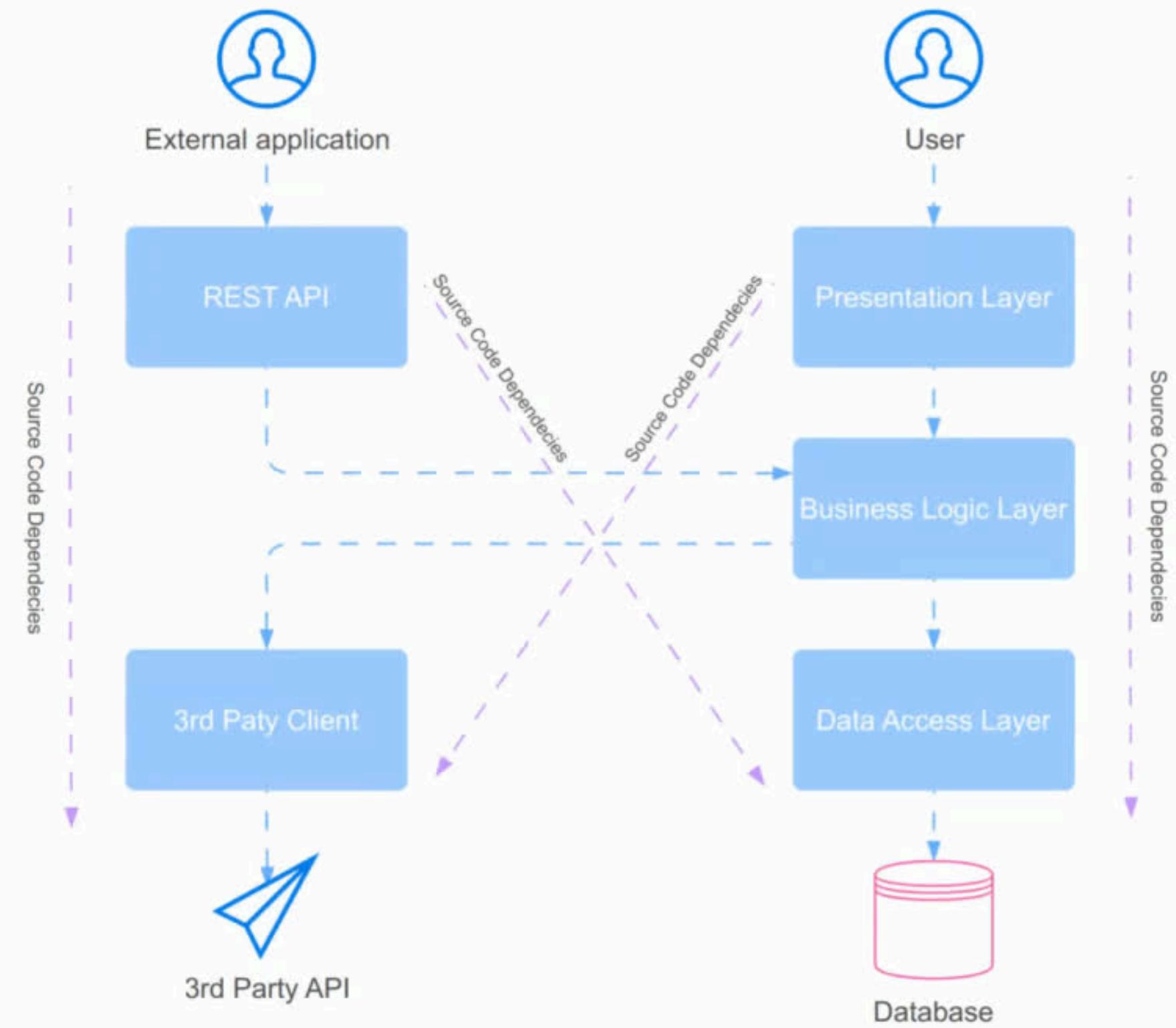
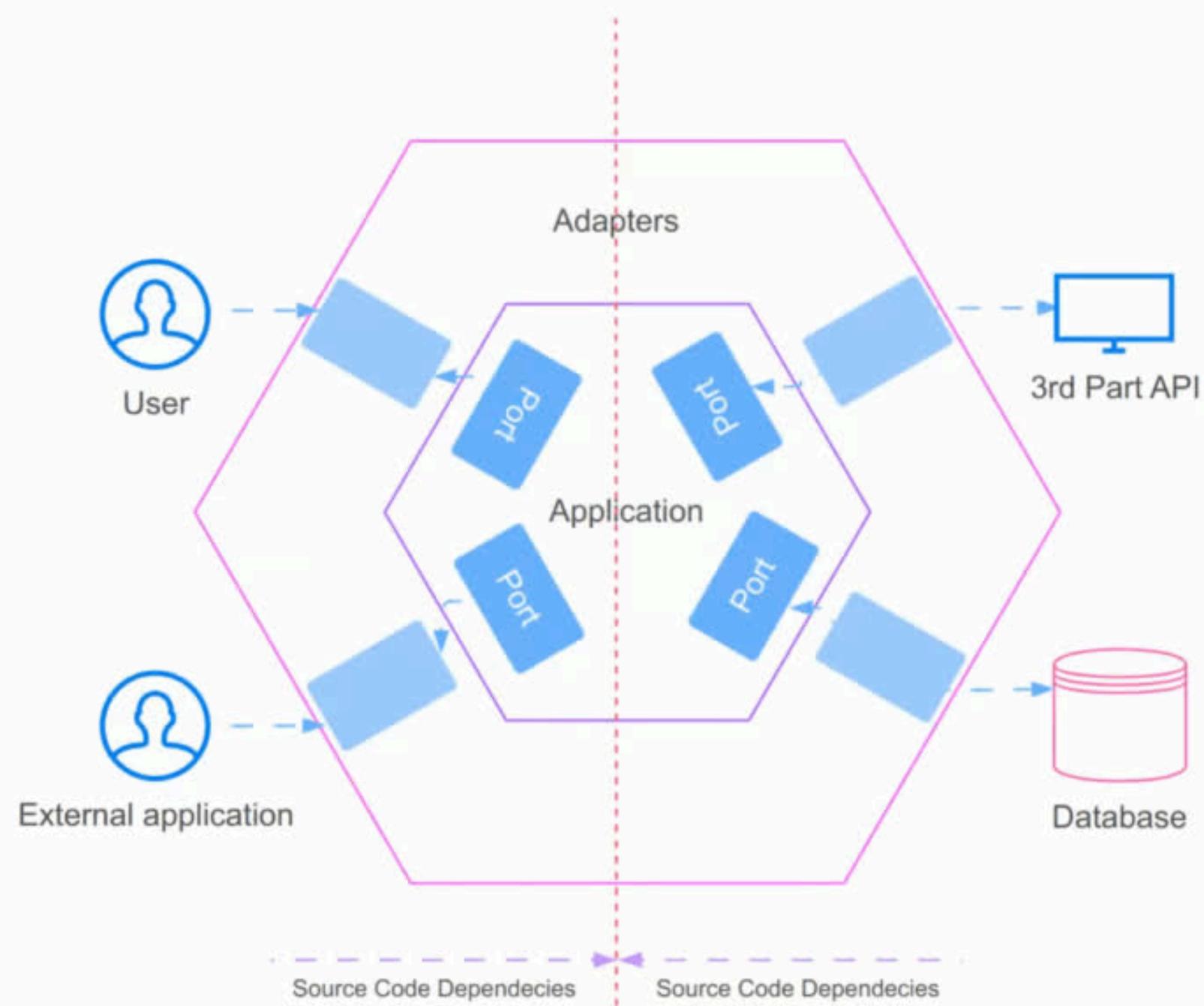
Foi proposta por Alistair Cockburn, visa criar sistemas de software que sejam altamente desacoplados e testáveis, permitindo que as partes centrais da aplicação permaneçam independentes de suas interfaces de entrada e saída.



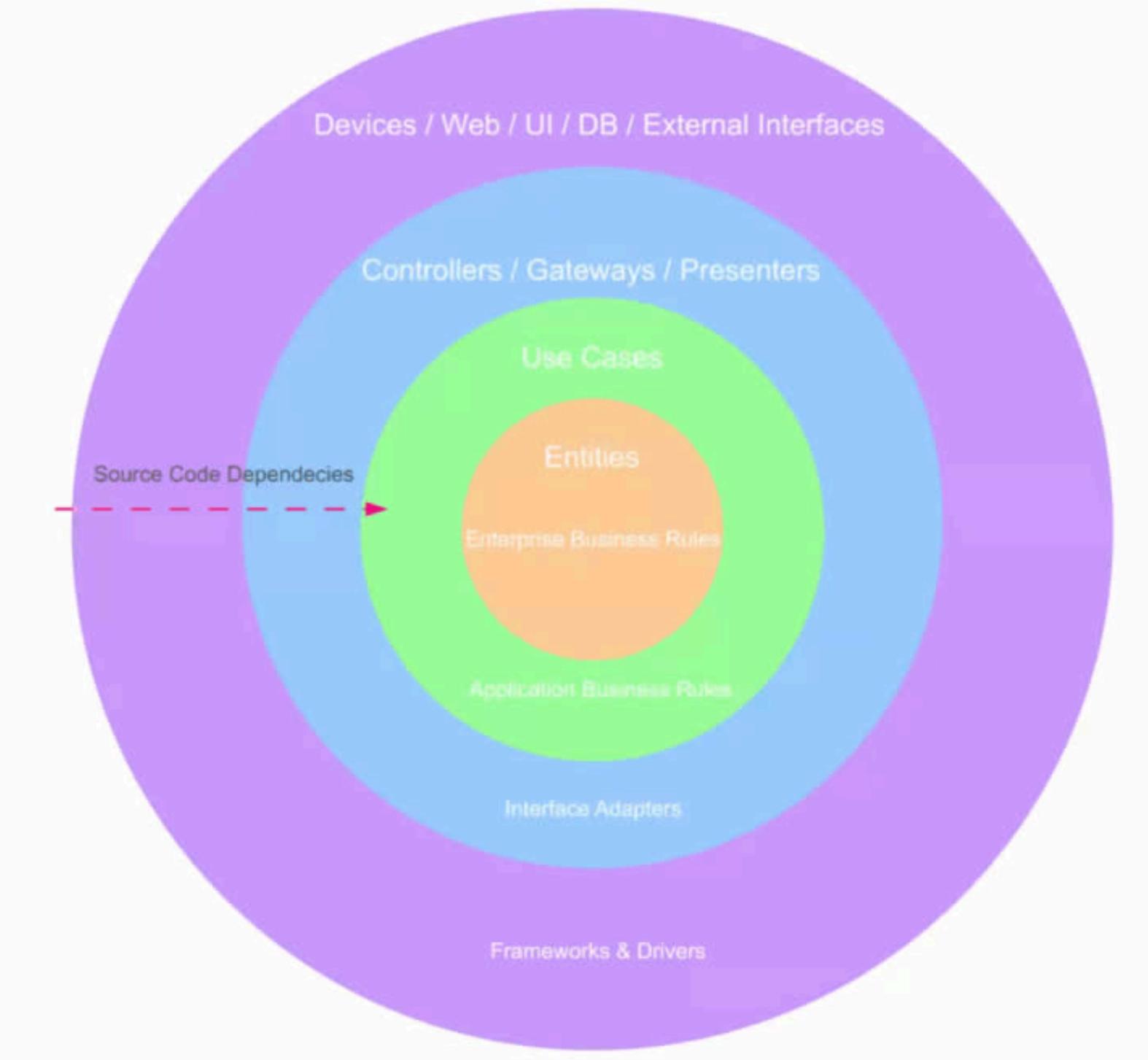
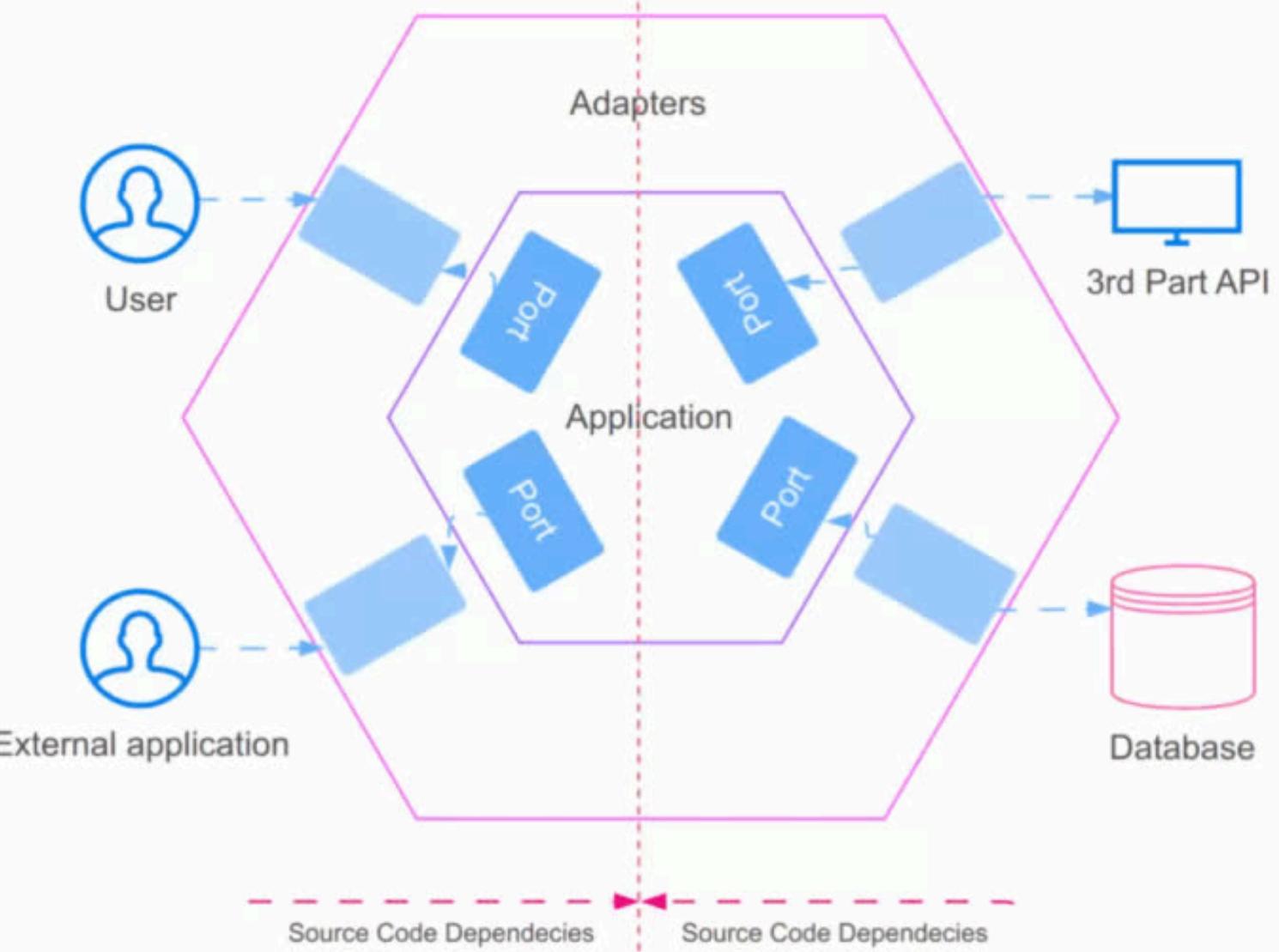
Comparação entre estilos



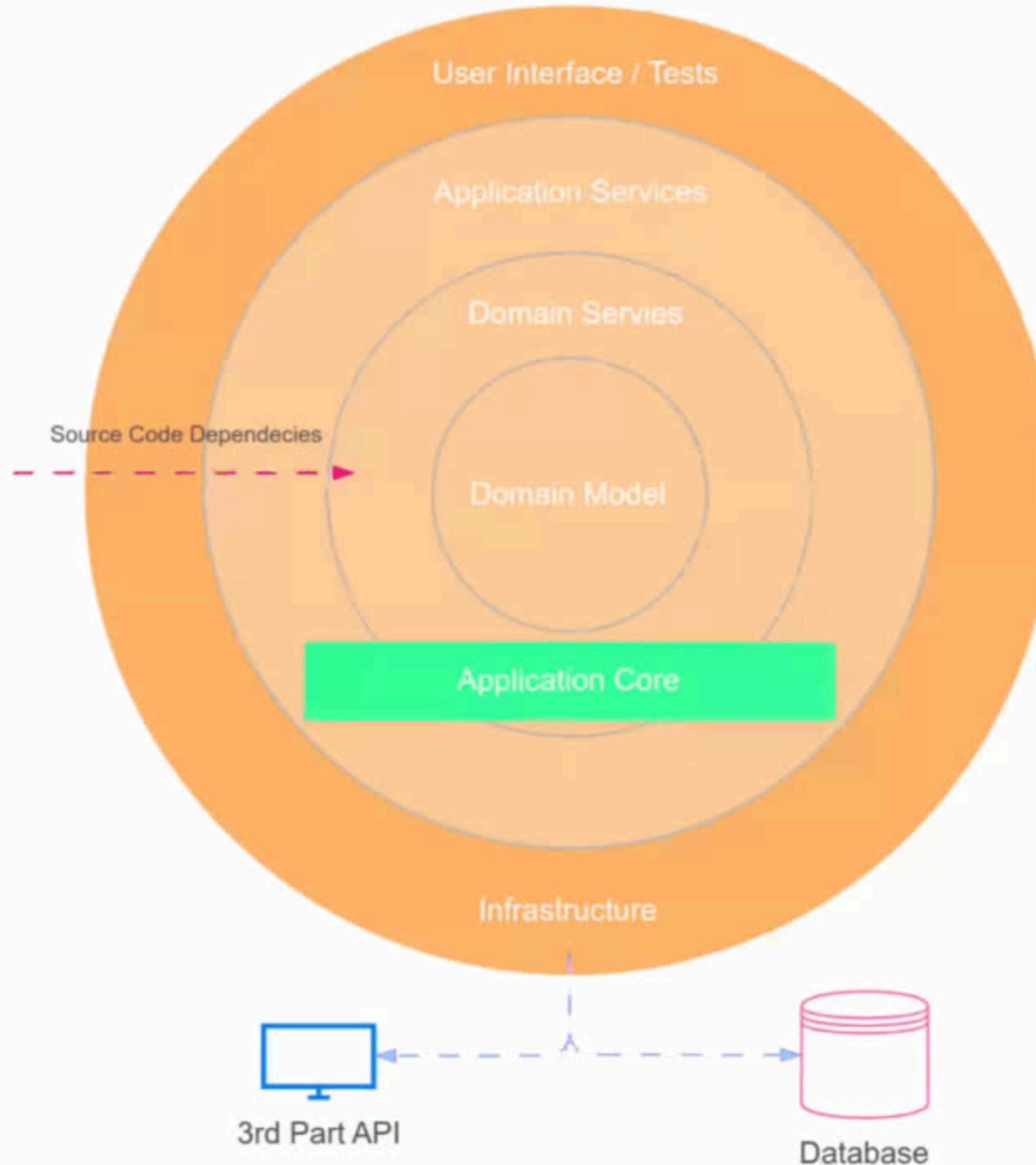
Clean Arch x Hexagonal x Onion



Layered é criticada por focar no banco de dados, causando dependências transitivas, limites confusos e dificuldades na manutenção e testes. Já a arquitetura hexagonal prioriza a lógica de negócios, desenvolvendo-a antes da persistência dos dados. Ela se adapta melhor a aplicativos complexos com componentes adicionais, como APIs REST, evitando dependências excessivas e duplicação da lógica de negócios.



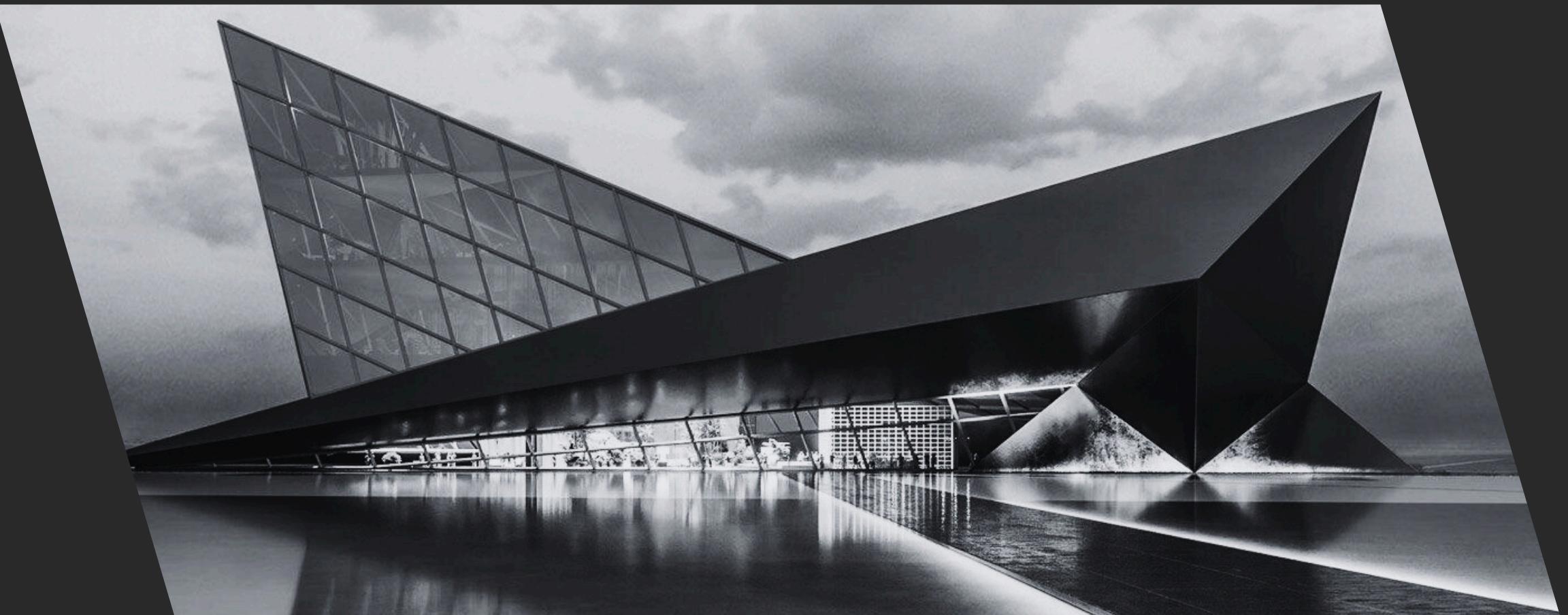
A Arquitetura Limpa coloca a lógica de negócios no centro, com adaptadores de interface ao redor conectando a interface do usuário, banco de dados e outros componentes externos. Todas as dependências do código apontam para o núcleo, seguindo o Princípio de Inversão de Dependência. Arquitetura Hexagonal mapeia-se quase diretamente para a Arquitetura Limpa.



Arquitetura Limpa pode ser vista como uma evolução da Arquitetura Onion e Hexagonal, devido à sua abordagem mais detalhada e abrangente para a organização de sistemas de software.

A Arquitetura Limpa refina e expande os conceitos da Arquitetura Hexagonal, oferecendo uma estrutura clara e sistemática para a construção de aplicações desacopladas, modulares e testáveis.

Serverless



Arquitetura serverless é um modelo de computação em nuvem onde os provedores de nuvem gerenciam a execução do código, alocando dinamicamente os recursos necessários.

Esse modelo permite que os desenvolvedores se concentrem mais na lógica de negócios e menos na infraestrutura, uma vez que tudo isso é responsabilidade do provedor de serviços de nuvem.

Características:

- Execução sob demanda, escalabilidade automática
- Custo-eficiência, gerenciamento simplificado

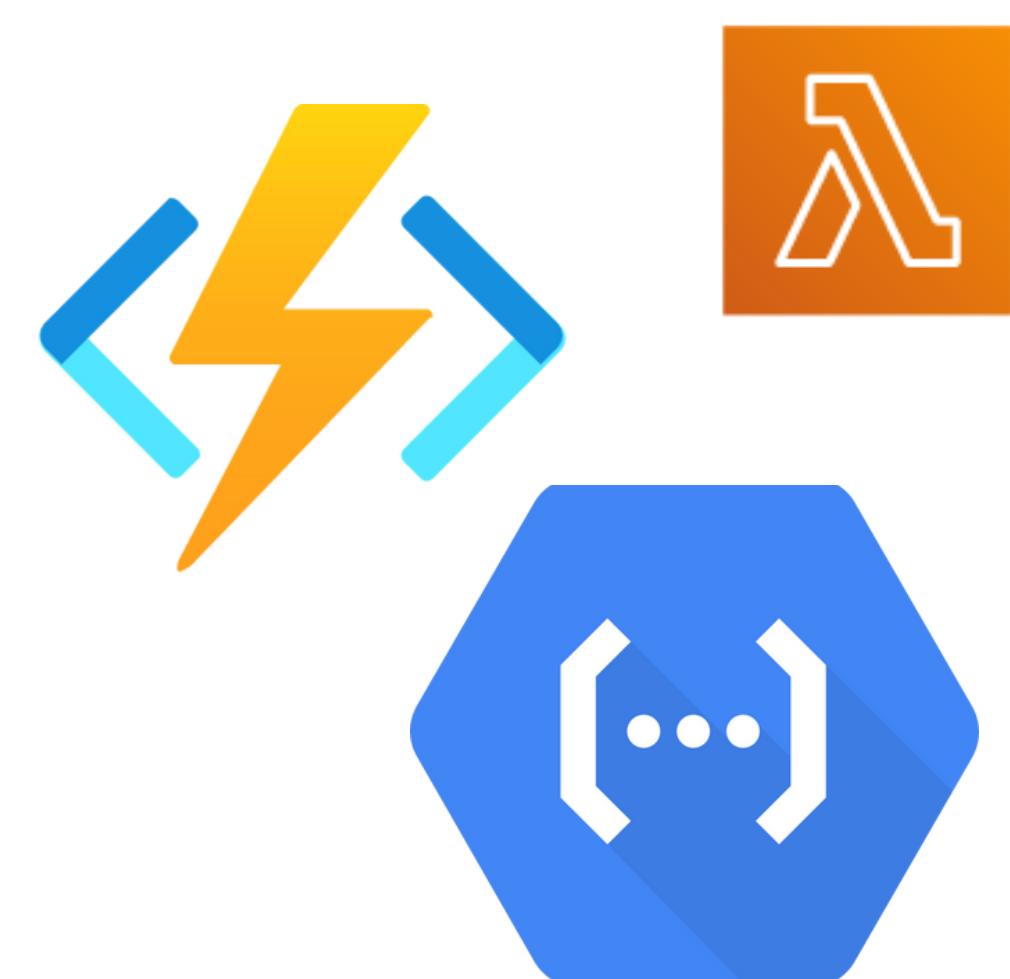
Funções como Serviço (FaaS)

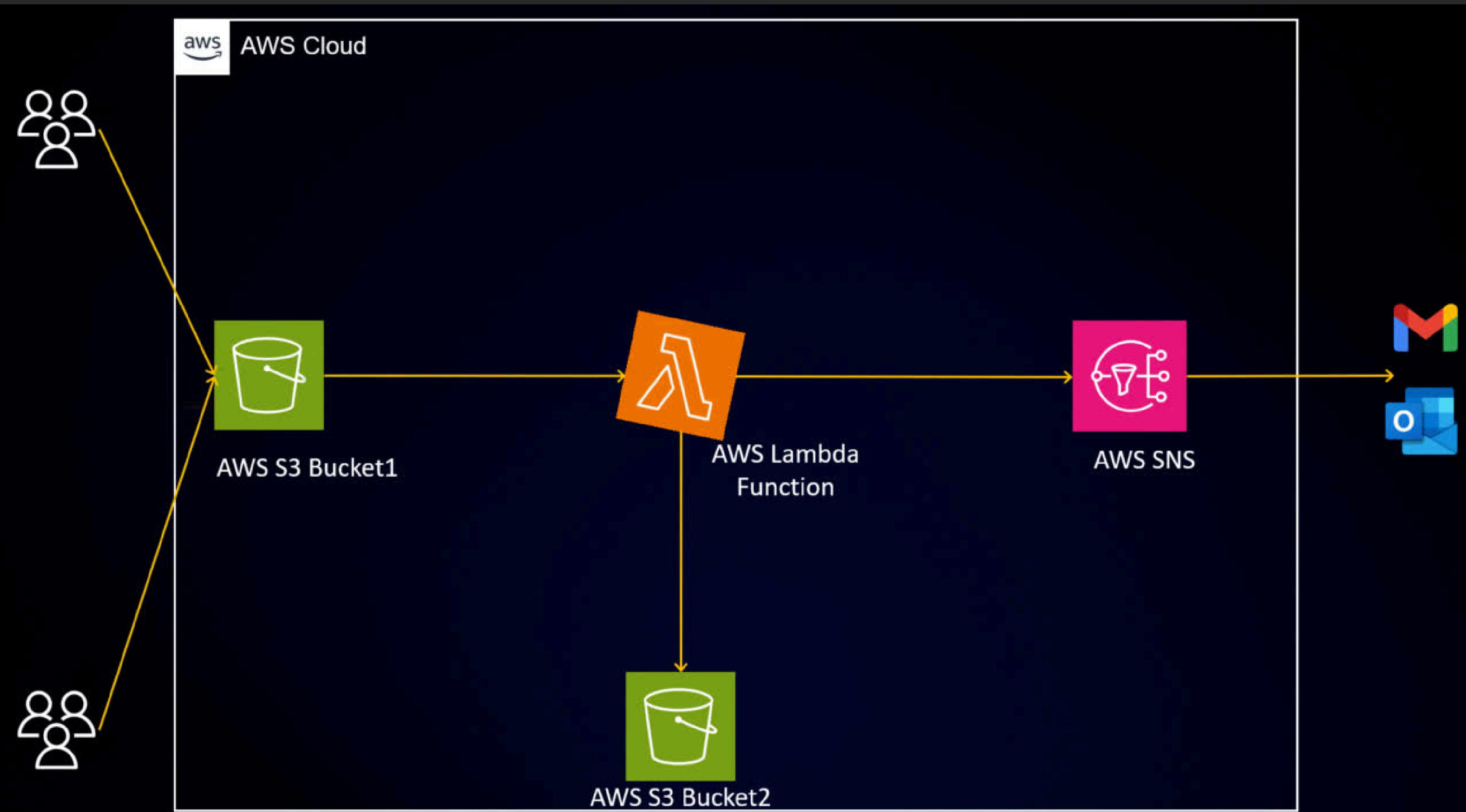
- AWS Lambda, Google Cloud Functions, e Azure Functions

Backend como Serviço (BaaS)

- Firebase Authentication, Firestore e Amazon S3

Podemos encontrar muitos dos conceitos no [artigo de Mike Roberts](#).





Qual o estilo arquitetural correto?

Depende....

Temos de levar em conta o problema que queremos resolver, budget do projeto, senioridade do time, isso também vale para quando buscamos alterar o estilo atual.



1º - Quero criar uma nova aplicação web, uma ferramenta de gestão de projeto, o que seria interessante ter nela?

- O budget para implementar é relativamente baixo
- Deve permitir adicionar novas funcionalidades
- Senioridade do time é baixa



**VOCÊ NÃO PODE USAR MVC E
MONÓLITO EM TUDO!**

2º Quero fazer uma API simples com Spring para o PDV de uma loja, qual a melhor escolha?

- Projeto pessoal/MVP
- Budget baixo



**KKK MAIS UM FREELA PRO
PAI**

Padrões Arquiteturais



Padrões arquiteturais

São soluções recorrentes para problemas específicos de design em um contexto arquitetural. Eles são mais específicos que estilos arquiteturais e fornecem detalhes sobre a implementação.

- Solucionam problemas específicos
- Guias de implementação
- Reutilização de soluções

MVC - Model, View e Controller

É um padrão de design de software que separa uma aplicação em três partes principais:

1. Model (Modelo):

- Representa os dados e a lógica de negócios.
- Gerencia o comportamento e os dados da aplicação.

2. View (Visão):

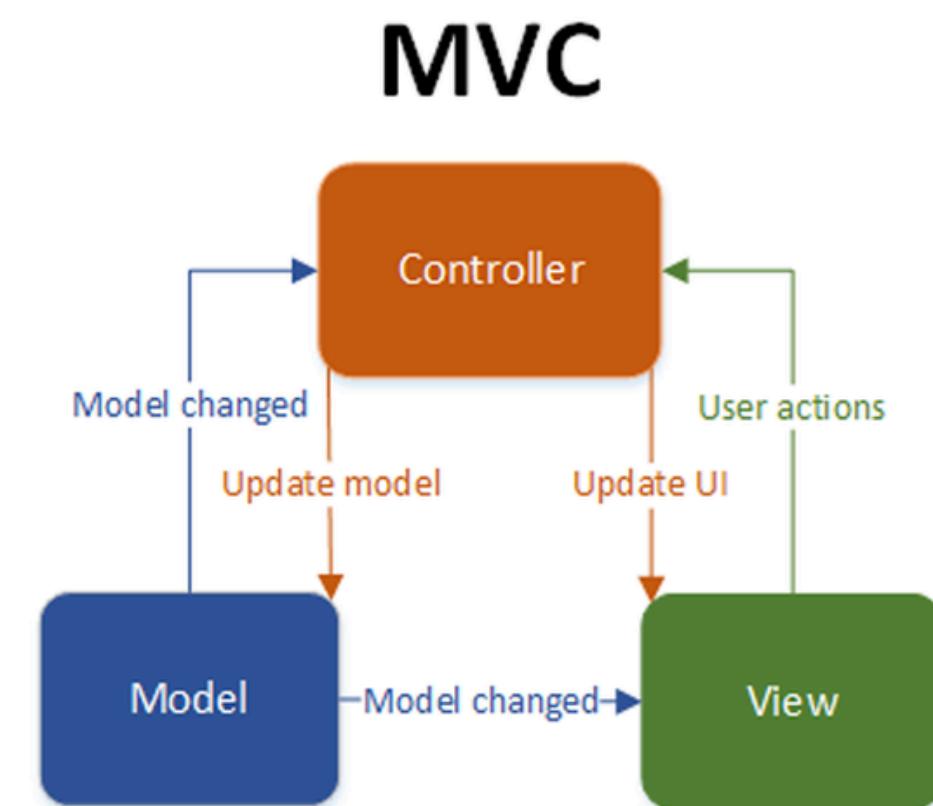
- Apresenta os dados ao usuário.
- Atualiza a interface do usuário quando o modelo muda.

3. Controller (Controlador):

- Recebe a entrada do usuário.
- Interage com o modelo e seleciona a visão para apresentar a saída.

Benefícios:

- Separação de preocupações: Facilita a manutenção e a escalabilidade.
- Reutilização de código: Componentes podem ser reutilizados em diferentes partes da aplicação.



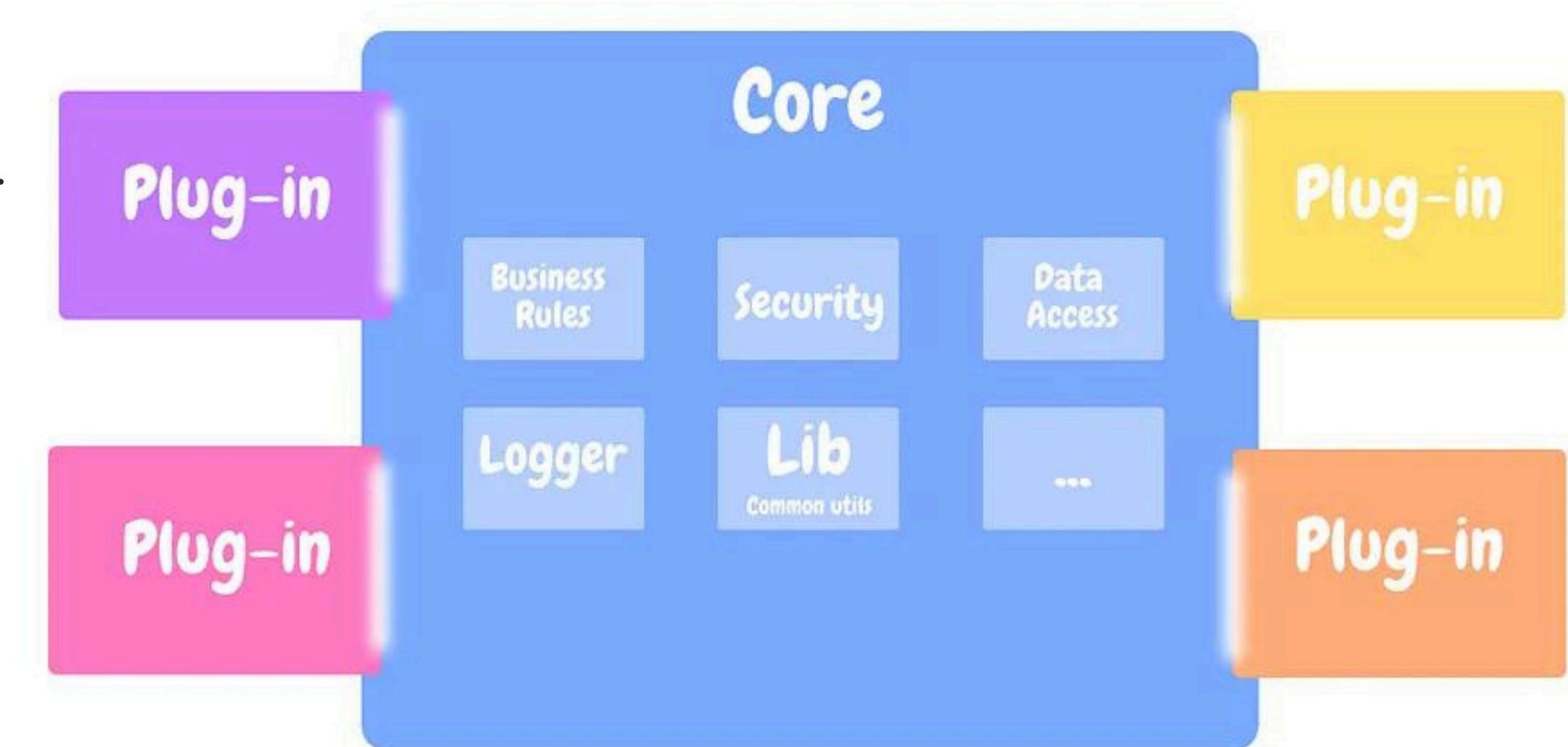
Plug-ins

Núcleo e plug-ins ou extensões que adicionam funcionalidades. O núcleo fornece a base e a infraestrutura, enquanto os plug-ins fornecem funcionalidades específicas.

Alguns locais citam como estilo arquitetural, porém partilho da visão de que ela como solução não possui "tamanho" suficiente para ser um estilo completo, se enquadrando como padrão que pode ser composto em estilos arquiteturais como o Layered e Componend-based.

Exemplos: Navegadores, IDE's como Eclipse.

Artigo que se oprofunda no padrão.



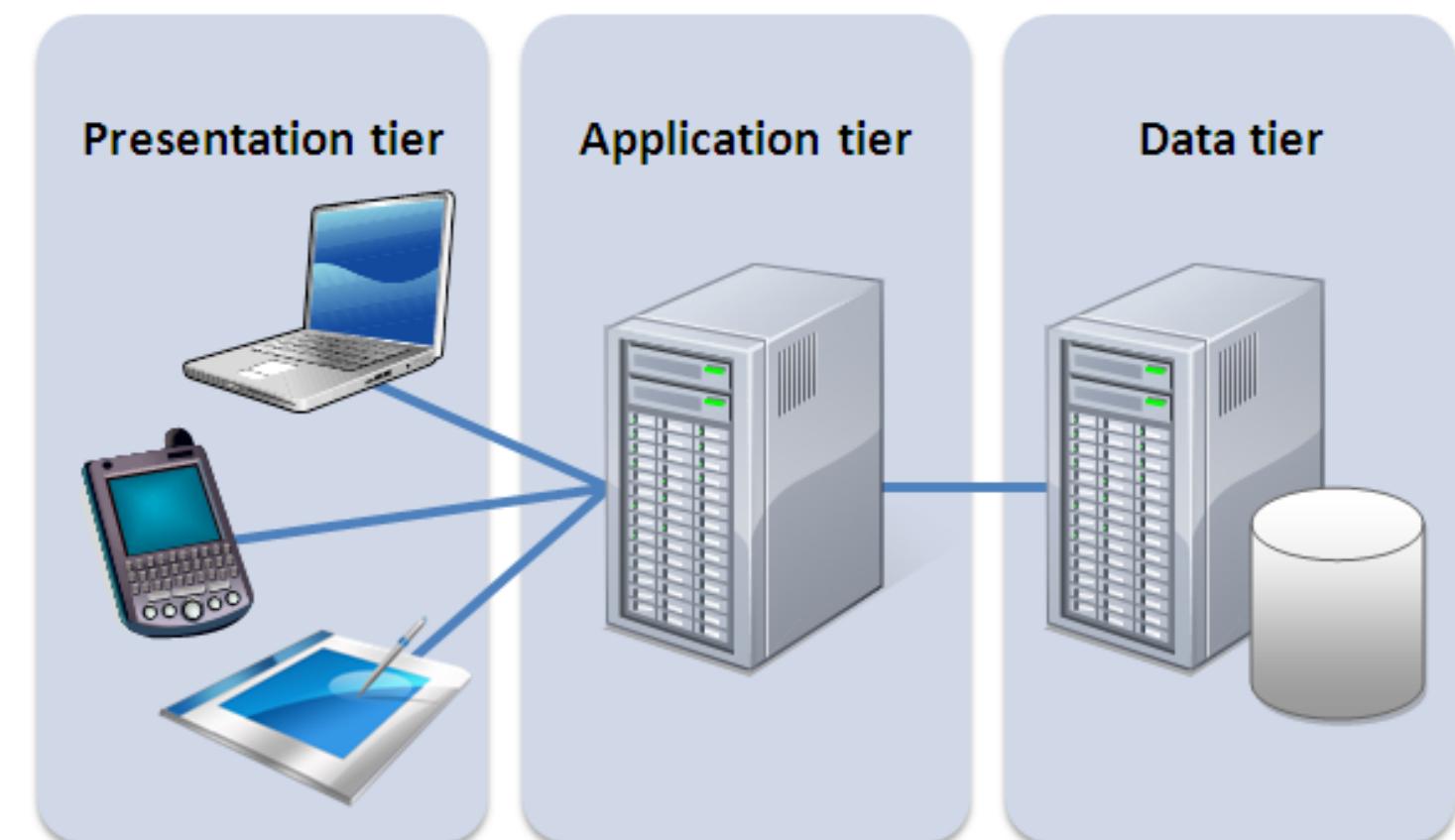
Three-tier

A arquitetura de três camadas é um modelo de arquitetura de software que separa uma aplicação em três camadas físicas distintas: a camada de apresentação (Presentation Tier), a camada de lógica de negócios (Business Logic Tier), e a camada de dados (Data Tier).

Essas camadas são distribuídas em diferentes servidores ou máquinas para melhorar a escalabilidade, a segurança e a manutenibilidade do sistema.

Benefícios

- Escalabilidade
- Segurança
- Manutenibilidade
- Flexibilidade e reutilização

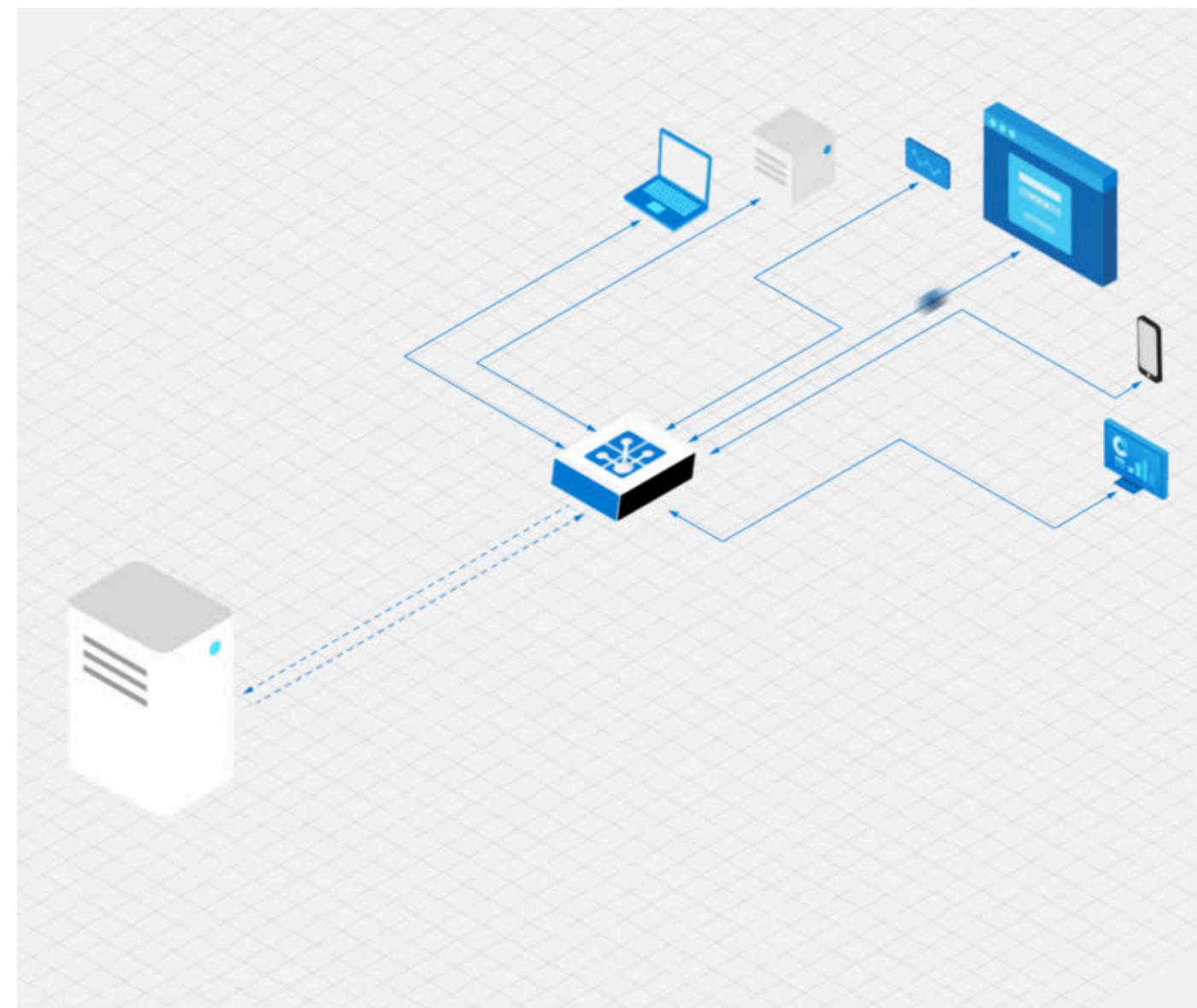


Publisher-subscriber

Padrão arquitetural que permite que os sistemas de software se comuniquem de forma assíncrona e desacoplada. Nesse modelo, os componentes de software são divididos em dois principais papéis: publicadores (publishers) e assinantes (subscribers). Esse padrão é amplamente utilizado em sistemas distribuídos e arquiteturas orientadas a eventos.

Benefícios:

- Desacoplamento: Publishers e subscribers são desacoplados, permitindo que componentes do sistema evoluam independentemente.
- Escalabilidade: Facilita a escalabilidade, pois múltiplos subscribers podem receber a mesma mensagem sem aumentar a carga no publisher.
- Flexibilidade: Suporta diversos padrões de comunicação (um-para-n, n-para-n).



Como escolher o padrão correto?

O que faça mais sentido com o estilo, ou estilos de arquitetura utilizados no projeto. Podemos ter um certo relacionamento entre os estilos e seus padrões que possuem mais sinergia.



Abaixo temos uma tabela com os principais estilos arquiteturais e seus respectivos padrões de arquitetura, juntamente com autores relevantes para cada estilo.

Estilo Arquitetural	Descrição	Principais Padrões	Autor Relevante
Arquitetura em Camadas (Layered)	Divide o sistema em camadas lógicas, cada uma responsável por uma parte específica da funcionalidade.	- Padrão de Camada de Serviço (Service Layer)	Martin Fowler
Arquitetura Cliente-Servidor (Client-Server)	Divide o sistema em dois componentes principais: clientes e servidores.	- Padrão de Proxy - Padrão de Servidor de Aplicações (Application Server) - Padrão de Banco de Dados Distribuído	Andrew S. Tanenbaum
Arquitetura de Microsserviços (Microservices)	Divide o sistema em serviços pequenos, independentes e implementáveis separadamente.	- Padrão de API Gateway - Padrão de Registro de Serviços (Service Registry) - Padrão de Circuit Breaker	Sam Newman
Arquitetura Orientada a Serviços (SOA)	Utiliza serviços desacoplados e interoperáveis que podem ser orquestrados para formar processos de negócios.	- Padrão de Serviço Web (Web Service) - Padrão de Barramento de Serviços Empresariais (ESB) - Padrão de Orquestração de Serviços	Thomas Erl
Arquitetura Baseada em Eventos (Event-Driven)	Baseia-se na produção, detecção, consumo e reação a eventos.	- Padrão de Publish-Subscribe - Padrão de CQRS - Padrão de Event Sourcing	Martin Kleppmann
Arquitetura de Microkernel (Microkernel Architecture)	Núcleo mínimo e permite a adição de funcionalidades ao sistema através de módulos ou componentes plugáveis.	- Padrão de Plug-ins (Plugin) - Padrão de Extensibilidade Modular - Padrão de Injeção de Dependências	Martin Fowler
Arquitetura Limpa (Clean Architecture)	Organiza o código em camadas concêntricas, onde a parte mais externa depende da parte mais interna.	- Padrão de Camadas Concêntricas - Padrão de Regra de Dependência (Dependency Rule) - Padrão de Entidade-Caso de Uso-Interface	Robert C. Martin
Arquitetura Hexagonal (Hexagonal)	Promove a independência dos componentes internos do sistema das suas interfaces externas.	- Padrão de Portas e Adaptadores - Padrão de Interface de Aplicação - Padrão de Adaptador de Interface	Alistair Cockburn
Arquitetura RESTful (REST)	Baseia-se em princípios da web para construir serviços escaláveis e facilmente mantíveis.	- Padrão de Interface Uniforme - Padrão de Stateless - Padrão de Cacheabilidade	Roy Fielding
Arquitetura Pipes and Filters	Os dados passam através de uma série de processadores conectados por canais (pipes).	- Padrão de Filtros Independentes - Padrão de Encadeamento de Filtros - Padrão de Canal de Comunicação	Frederick P. Brooks