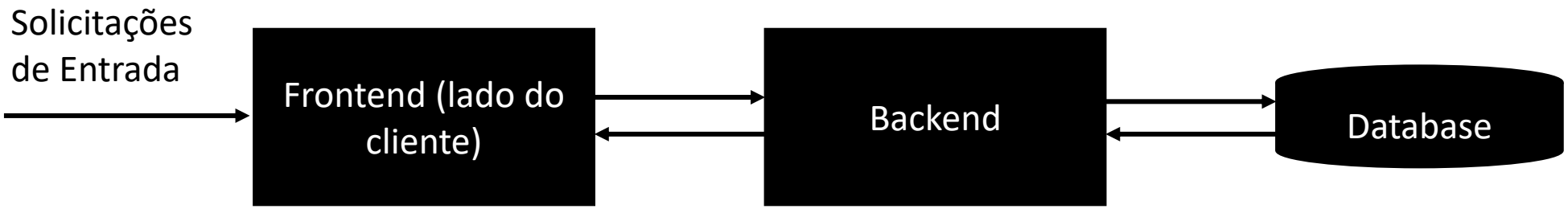
Programação Web com Linguagens de Script

Thiago Rodrigues

Arquitetura em Três Camadas

- Quase todas as aplicações de software escritas atualmente podem ser divididas em três elementos distintos:
  - Frontend
  - Backend
  - Armazenamento de dados



Arquitetura em Três Camadas

- Existe três maneiras diferentes de combinar esses elementos para criar uma aplicação.
    - Frontend e Backend na mesma aplicação e banco de dados separado
    - Frontend e Backend como aplicações diferentes, acompanhados por uma banco de dados externo
    - Frontend e Backend na mesma aplicação com armazenamento em memória.

Impacto da escalabilidade da aplicação

Evolução de uma aplicação

- **Estágio inicial de uma empresa**
  - Aplicações Simples
  - Poucos desenvolvedores que contribuem para um mesmo code-base

- **Quando a empresa vai crescendo**
  - Mais desenvolvedores são contratados
  - Novos recursos são adicionados a aplicação

- **O crescimento da empresa gera algumas consequências**

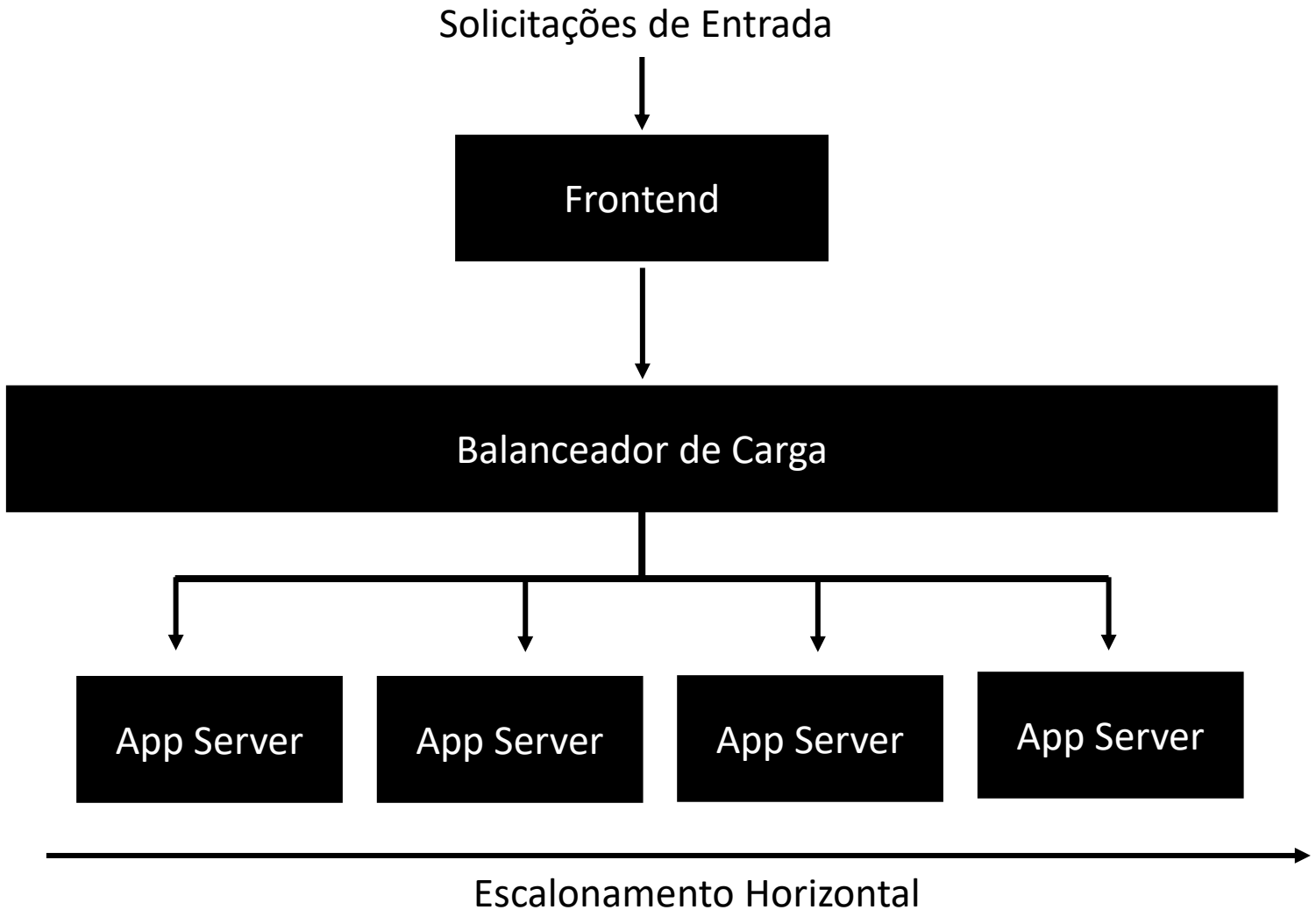
Consequências da Evolução

- Aumento de carga de trabalho operacional
  - Associado à execução e manutenção da aplicação
  - Contratação de engenheiros operacionais (TechOps e DevOps)

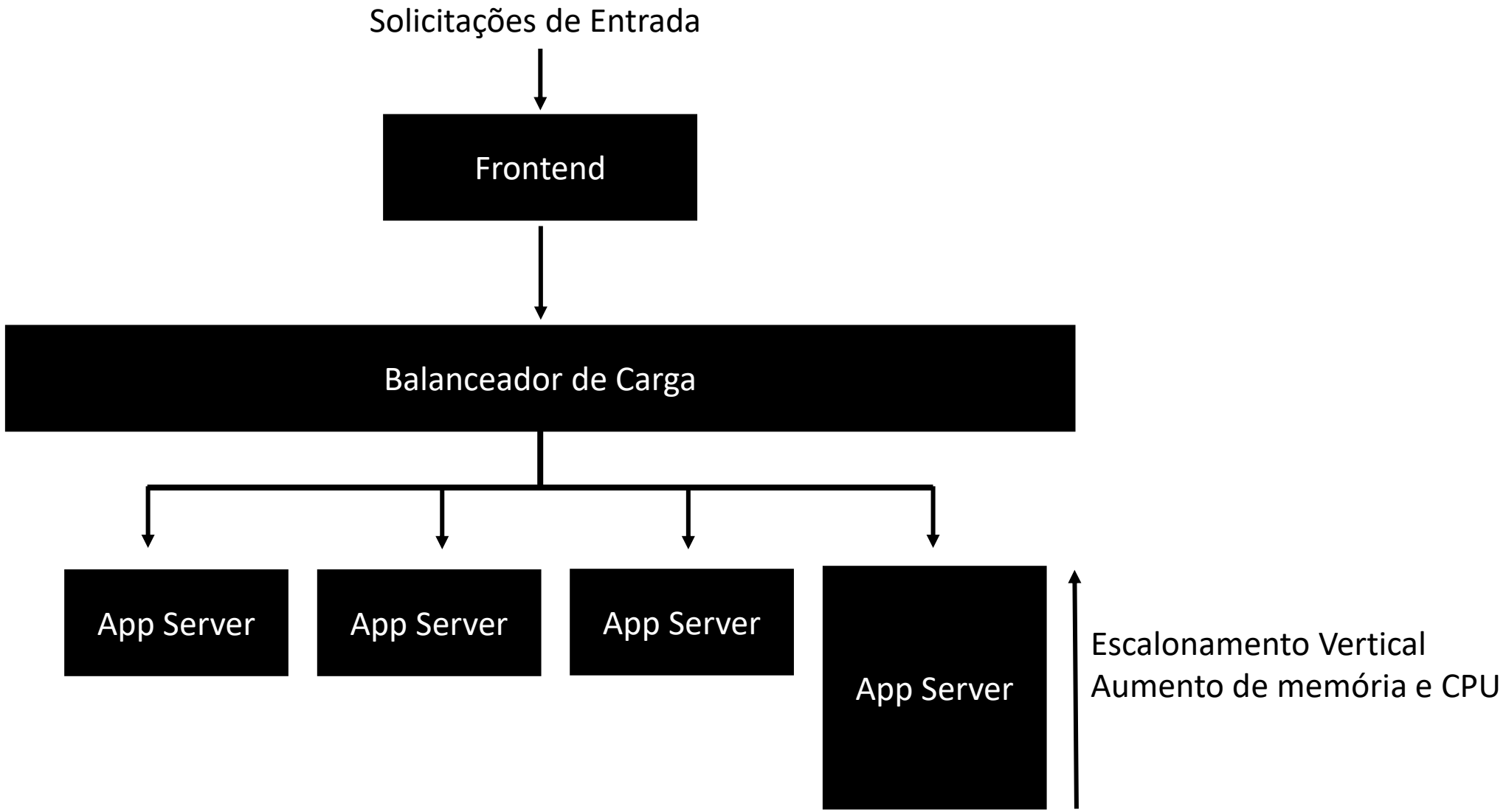
- Aumento na complexidade da aplicação

- Necessário dimensionamento **vertical** e **horizontal** da aplicação

Dimensionamento Horizontal



Dimensionamento Vertical



Resultado do Crescimento

- Engenheiros ultrapassam a grandeza das centenas
- Tudo começa a ficar mais complicado
- A complexidade da aplicação cresce constantemente
  - Patchs de correções, novos recursos
- Centenas de testes sendo feitos para garantir que as alterações feitas não comprometa a integridade de milhares de linhas de códigos existentes.
- Correções sendo adiadas, aumentando a defasagem técnica

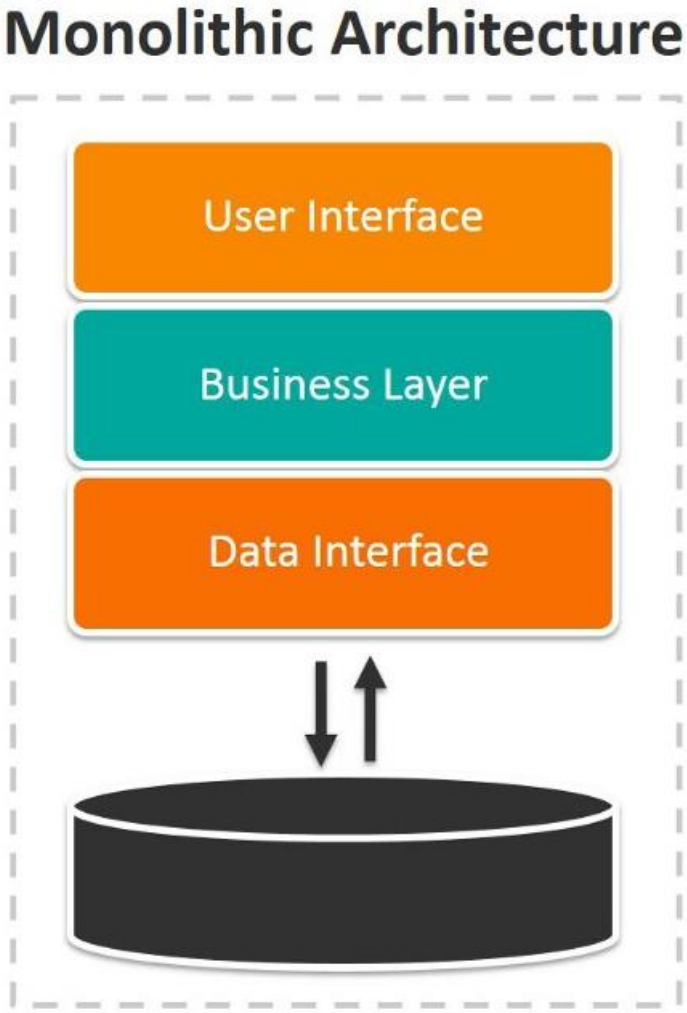
Dimensionamento de uma Aplicação

- Uma aplicação precisa Escalar, para isso são necessários
  - Simultaneidade
  - Segmentação

- **Simultaneidade**
  - Uma tarefa ser dividida em partes menores
  - Não podemos ter um único processo para fazer todo trabalho
- **Segmentação**
  - Processar as pequenas tarefas em paralelo

Arquitetura Monolítica

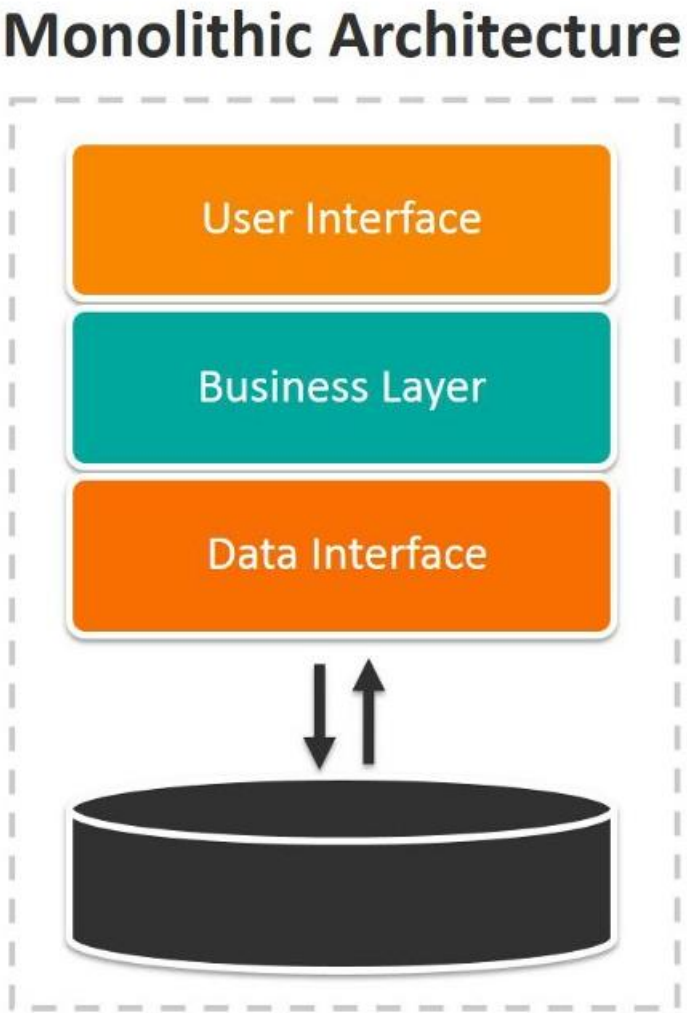
- Arquitetura Monolítica é um sistema único, não dividido, que roda em um único processo, uma aplicação de software em que as camadas de arquitetura dependem umas das outras.



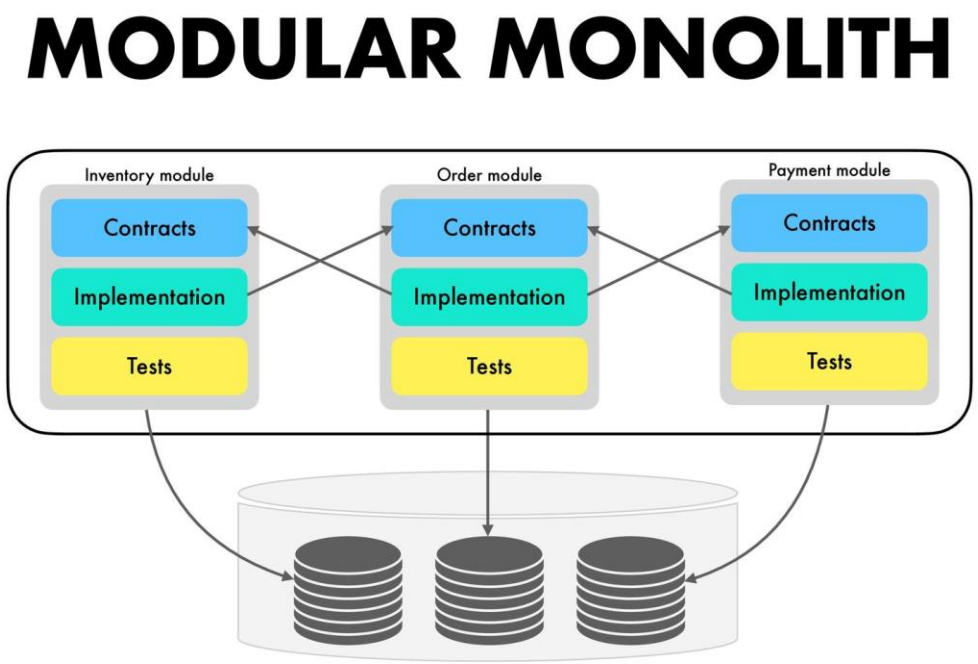
Tipos de Sistemas Monolíticos

- Sistema Monolítico de um só Processo
- Sistema Monolítico Modular

Sistema Monolítico de um só Processo



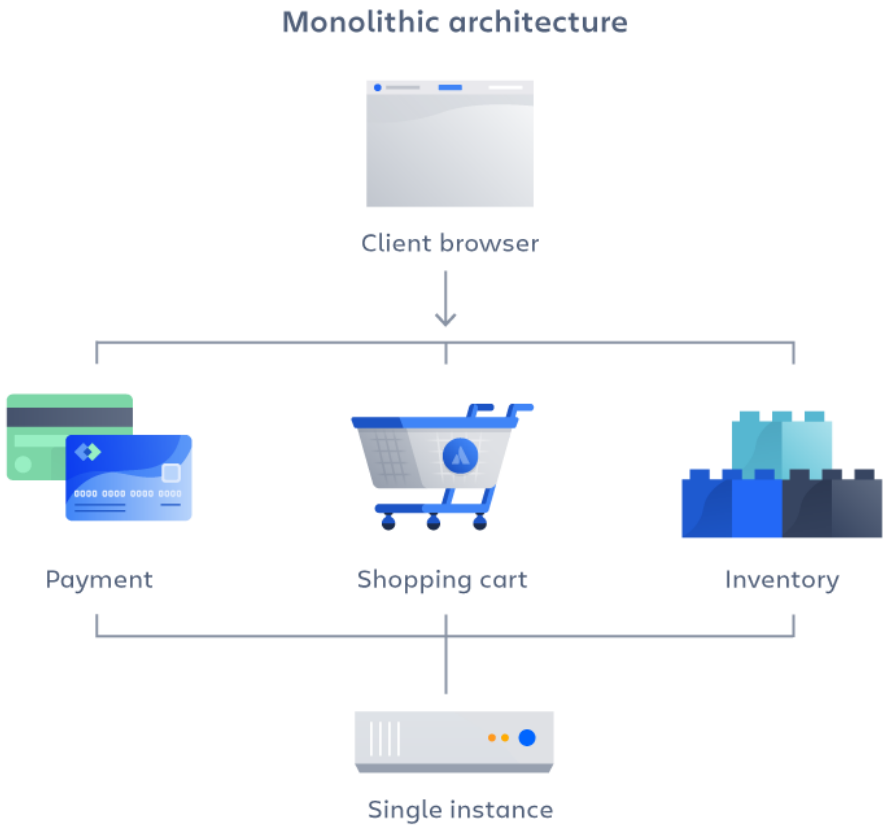
Sistema Monolítico Modular



- A troca de informações entre os módulos ocorre dentro de um mesmo processo.

Arquitetura Monolítica

- Como exemplo, imagine a criação de uma aplicação para uma loja de venda de produtos, onde há os setores de **estoque**, **pagamento** e **carrinho**.
- Todos os usuários utilizarão o mesmo sistema, e será preciso que ele esteja dividido em algumas partes, são elas:
  - Autenticação e perfis de usuários (administração, contabilidade, estoque, vendedor);
  - Gráficos para a administração com os dados diários da loja;
  - Compra de produtos;
  - Estoque;
  - Vendas.



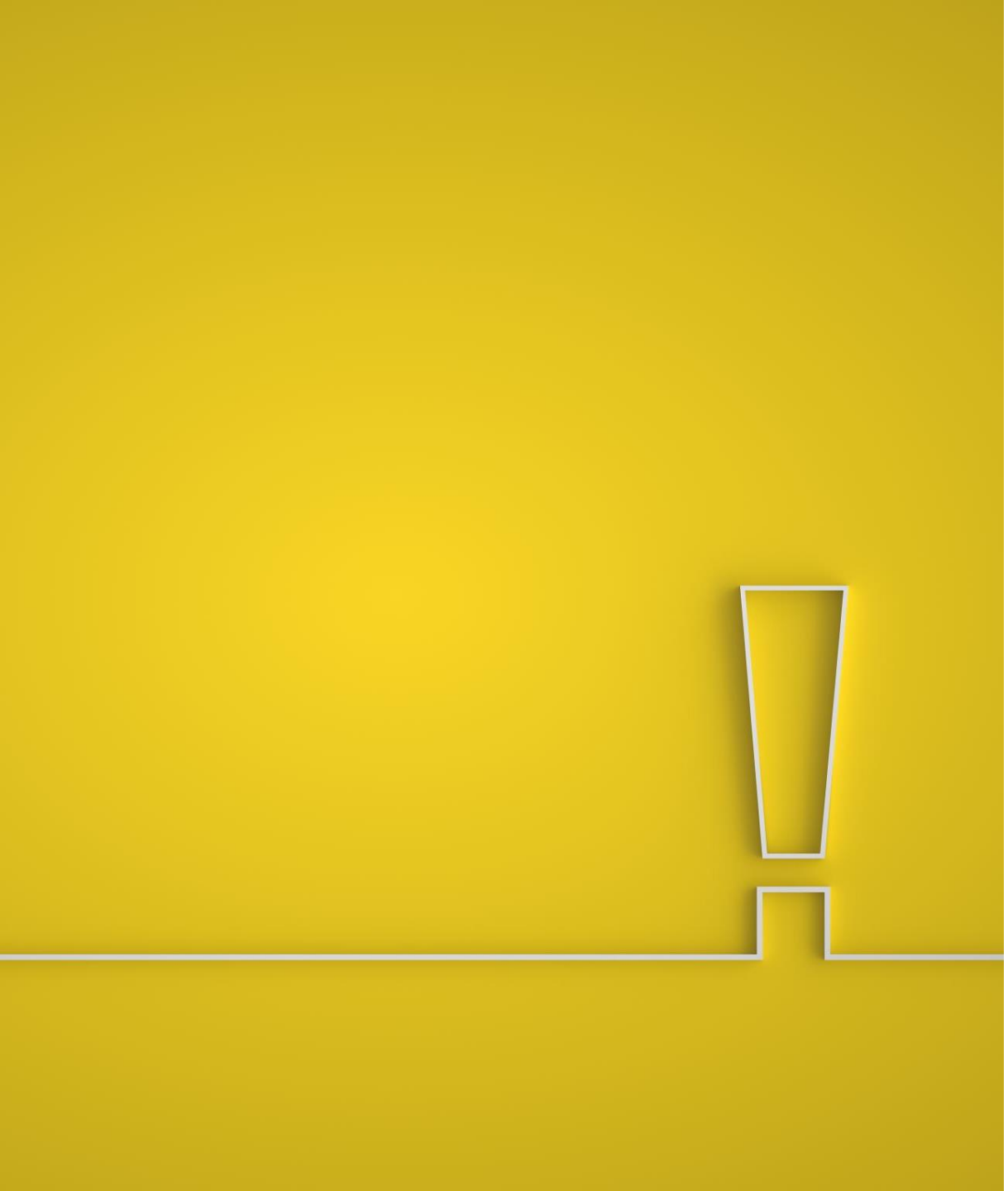
Observações

- Mais vulneráveis aos perigos do acoplamento.
- Quanto mais pessoas estiverem trabalhando em um mesmo lugar, elas começarão a atrapalhar umas às outras.
- Desenvolvedores diferentes querendo alterar a mesma parte do código.
- Desavença nas entregas.
  - Desafios das linhas confusas de responsabilidade

Arquitetura Monolítica não deve ser vista como algo que deve ser Evitado

Arquitetura Monolítica

- **Vantagens**
  - **Mais simples de desenvolver (código):** a organização fica concentrada em um único sistema;
  - **Simples de testar:** é possível testar a aplicação de ponta a ponta em um único lugar;
  - **Simples de fazer o deploy para o servidor:** a alteração é simplesmente feita e pronto;



Arquitetura Monolítica

- **Desvantagens**
  - **Baixa Escalabilidade:** grande base de código;
  - **Manutenção:** a aplicação se torna cada vez maior de acordo com o seu tamanho, o código será cada vez mais difícil de entender e o desafio de fazer alterações rápidas e ter que subir para o servidor só cresce;
  - **Alterações:** para cada alteração feita, é necessário realizar um novo deploy de toda a aplicação;
  - **Linha de código:** uma linha de código que subiu errada pode quebrar todo o sistema e ele ficar totalmente inoperante;
  - **Linguagens de programação:** não há flexibilidade em linguagens de programação. Aquela que for escolhida no início do projeto terá que ser seguida, sempre. Se o desenvolvimento de uma nova funcionalidade exigir outra linguagem de programação, existem duas possibilidades: ou todo o código é alterado ou a arquitetura do sistema precisará ser trocada.

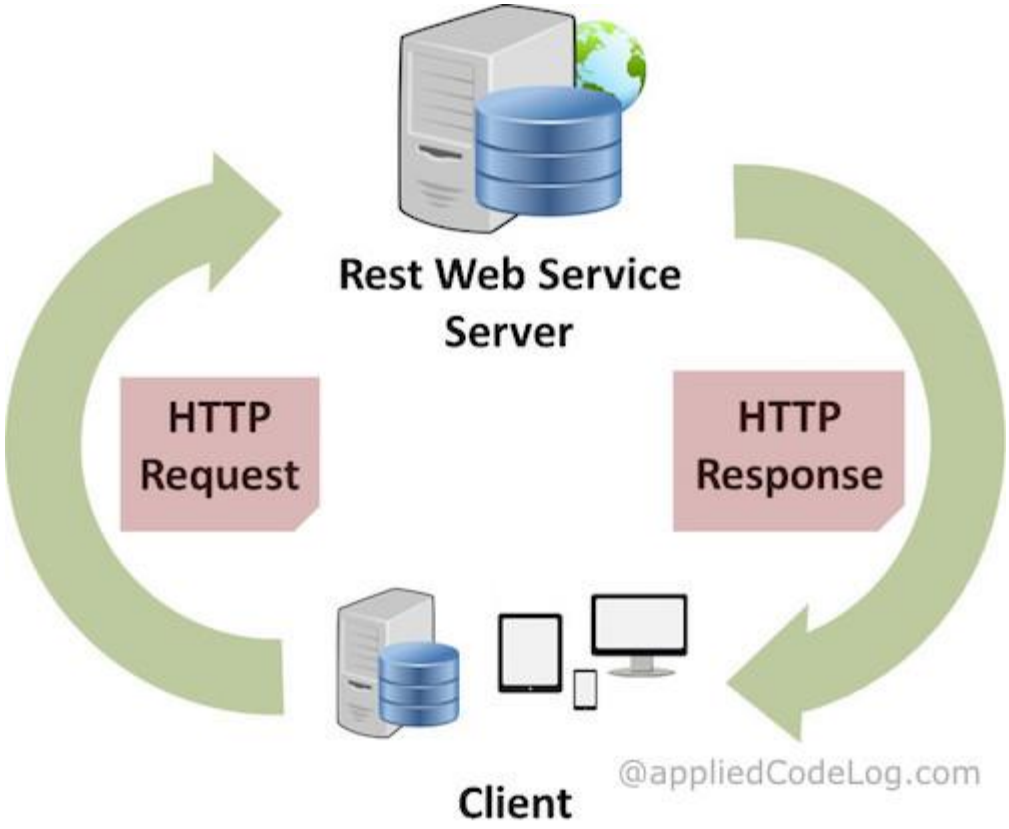
Web Service

- A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.

WebService

- Solução utilizada na **integração** de sistemas e na **comunicação** entre **aplicações**.

- Através dos web services, **sistemas** podem abrir suas portas (de maneira controlada) para o mundo, conseguindo se **comunicar** com outros sistemas de uma maneira **padronizada** e **independente** de sua plataforma ou **linguagem de programação**.



WebService

- *Com webservices, é possível expor funcionalidades de um sistema através de um protocolo padronizado.*

- Os **web services** são **baseados** em um conjunto de **padrões** da internet definidos pelo **W3C**, não requerem configurações especiais nos firewalls, pois o protocolo http, o qual é o mais utilizado, atua como transportador na comunicação entre cliente e web service.

WebService

- **Padrões**
  - SOAP (**Protocolo Simples de Acesso a Objetos**)
  - REST (**Transferência Representacional de Estado**)
  - XML-RPC (**Chamada de Procedimento Remoto**)
  - ...

SOAP

- Abreviação para Simple Object Access Protocol
- Especificação para a troca de informação entre sistemas
- Especificação de formato de dados para envio de estruturas de dados entre serviços, com um padrão para permitir a interoperabilidade entre eles.
- Design parte do princípio da utilização de XMLs para a transferência de objetos entre aplicações, e a utilização, como transporte, do protocolo de rede HTTP

SOAP

- Os XMLs especificados pelo SOAP seguem um padrão definido dentro do protocolo
- Web Service Description Language (WSDL) é um arquivo XML onde se encontra as definições dos métodos que compões um webservice SOAP.
- Muito utilizado em ambientes coorporativos e sistemas mais legados.
- Utiliza o método HTTP POST para realizar as ações no WebService

REST

- O **REST** parte do princípio de **seguir** as **boas práticas** da **criação de serviços HTTP** e utilizar esses padrões para desenvolver web services simples e performáticos.

- Na arquitetura REST é muito importante uso correto dos métodos disponibilizados pelo HTTP.

- Uma arquitetura **REST prevê** que, dentro de um **cenário ideal**, o **método HTTP a ser utilizado seja diretamente relacionado à funcionalidade do serviço a ser consumido**.

REST

- **Exemplo**
  - Serviços de busca de informações são feitos através de métodos **GET**
  - Serviços de atualização de informação através de métodos **PUT**
  - Serviços de criação de dados através do método **POST**
  - Serviços de deleção através do **DELETE**
  - Serviços para atualização parcial de um recurso **PATCH**
  - Serviços de busca de informações sem os dados do body são feitos através de métodos **HEAD**
  - Serviços para recuperar os métodos HTTP e outras opções que são suportadas por um servidor web ou uma operação, sem implicar uma ação de recurso ou iniciar uma recuperação de operação utiliza-se o **OPTIONS**

REST

- A partir do correto uso dos verbos HTTP ganhamos também a vantagem de não termos diversas URLs para cada um dos nossos serviços, podendo expor somente uma URL e, conforme o método, executar uma ação diferente.

- O REST não impõe restrições com o formato dado a ser trafegado, podendo ser XML ou JSON.

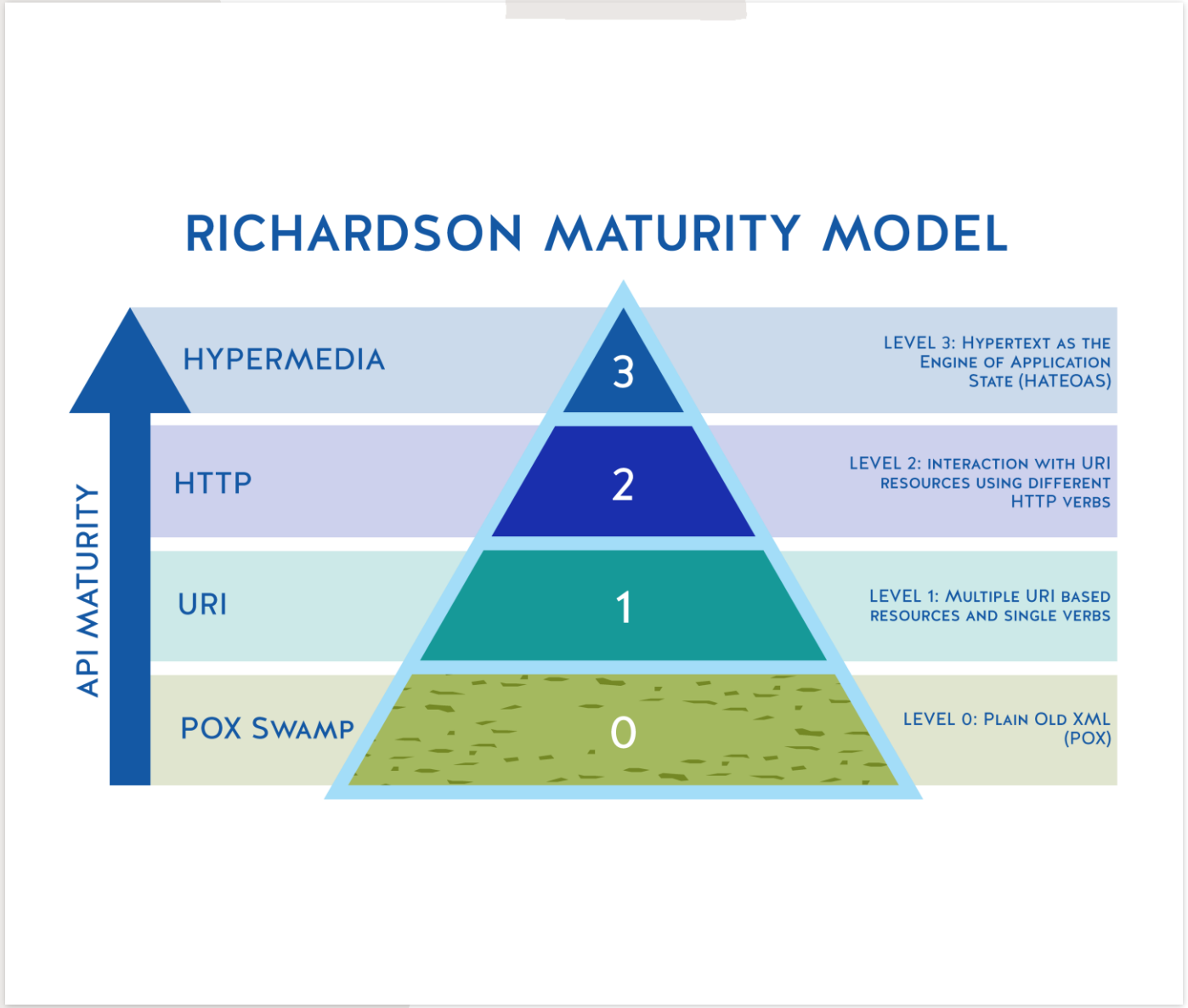
RESTful

- Uma aplicação que segue todas as definições de um serviço REST, é chamada de RESTful

- **Architectural Constraints**
  - Um sistema RESTful deve seguir o modelo Cliente-Servidor
  - Serviços RESTful devem ser stateless
  - Tem um conjunto de operações padronizadas, conforme já explicado pelas requisições do tipo GET, POST, PUT, DELETE.
  - Interface Uniforme
    - Identificação do recurso, Representação do Recurso, Mensagens auto-descritivas, Componente HATEOAS (Hipermidia as the engine of application state)
  - Cacheable
  - Separação em Camadas

Modelo de Maturidade de Richardson

Modelo de maturidade sugerido em 2008 por Leonard Richardson para classificar as APIs Web.



Richardson maturity model with API JSON response type

Lvl	Name	Example request	Response
0	Plain of XML (POX)	method: POST URI: /movie/	JSON
1	Resources with specific URI for action	method: POST URI: /movie/1/delete	JSON
2	Resources + HHTP methods	method: DELETE URI: /movie/1	JSON
3	HATEOAS (level 2 + extra links to navigate through API)	method: DELETE URI: /movie/1	JSON with HAL (extra links)

0 – endpoint único, apenasw utilização de POST e GET

```
{
    "cursos": [
        {
            "id": 1,
            "nome": "C# (C Sharp)",
            "aulas": "api.treinaweb.com.br/cursos/1/aulas"
        },
        {
            "id": 2,
            "nome": "PHP",
            "aulas": "api.treinaweb.com.br/cursos/2/aulas"
        },
        {
            "id": 3,
            "nome": "Java",
            "aulas": "api.treinaweb.com.br/cursos/3/aulas"
        },
    ]
}
```

Especificação do HATEOAS

- **RFC 5988**
- A especificação RFC 5988 da IETF define como links devem ser implementados. De acordo com ela, cada link deve ter as informações:
  - **URI:** Cada link deve conter uma URI, representada no atributo href;
  - **Tipo de relação:** Descreve como a URI se relaciona com o recurso atual, representado pelo atributo rel, devidado de relationship;
  - **Atributos para URI:** Para descrever melhor a URI podem ser adicionados atributos como: hreflang, media, title e type.

```
{
    "cursos": [
        {
            "id": 1,
            "nome": "C# (C Sharp)",
            "links": [
                {
                    "type": "GET",
                    "rel": "self",
                    "uri": "api.treinaweb.com.br/cursos/1"
                },
                {
                    "type": "GET",
                    "rel": "curso_aulas",
                    "uri": "api.treinaweb.com.br/cursos/1/aulas"
                },
                {
                    "type": "PUT",
                    "rel": "curso_atualizacao",
                    "uri": "api.treinaweb.com.br/cursos/1"
                },
                {
                    "type": "DELETE",
                    "rel": "curso_exclusao",
                    "uri": "api.treinaweb.com.br/cursos/1"
                }
            ]
        },
        {
            "id": 2,
            "nome": "PHP",
            "links": [
```

SOAP vs REST

- SOAP segue um protocolo específico
- REST segue um modelo arquitetural
- REST é mais leve
- SOAP possui funcionalidades padronizadas
  - Autenticação, manter seção entre requisições, URLs de Callback (WS Addressing)
- REST possui mais liberdade de implementação
  - O que deixa na responsabilidade dos desenvolvedores a qualidade do serviço
- SOAP já possui um padrão de desenvolvimento, tornando “engessado” o desenvolvimento e limitando a liberdade de customização.

Dúvidas

southiagorm@gmail.com