



# UM CASO DE MIGRAÇÃO PARA MICROSERVIÇOS

# SOBRE A DELIVERY MUCH

- ▶ Marketplace de Delivery
- ▶ Sistema de franquias
- ▶ 70 funcionários
- ▶ 14 estados
- ▶ 125 cidades
- ▶ 4000+ empresas
- ▶ Escritórios em Santa Maria e Florianópolis



# OS PROJETOS

# OS OBJETIVOS

- 🍕 Aplicativo novo
- 🍕 Software de restaurante
- 🍕 Nova API REST utilizando microserviços
- 🍕 Quebrar o monolito

# O DESAFIO

# DESAFIOS

- 🍕 Nova arquitetura
- 🍕 Migrar o que já existia e manter os sistemas legados rodando
- 🍕 Organizar a equipe

# ARQUITETURA

## SERVIÇOS EXISTENTES

- 🍕 BAGUETE (Site)
- 🍕 PANEL (Painel dos Franqueados)
- 🍕 LEMON API (Software dos Restaurantes)

# STACK DE TECNOLOGIA

- 🍕 Docker
- 🍕 NodeSJ + Express + Koa
- 🍕 MongoDB / MariaDB / Redis
- 🍕 Infra AWS
- 🍕 RabbitMQ

# O QUE SÃO MICROSERVIÇOS?

“  
SÃO PEQUENOS SERVIÇOS  
AUTÔNOMOS QUE  
TRABALHAM EM CONJUNTO.

— *Sam Newman, Thoughtworks*

”

“  
UMA ARQUITETURA ORIENTADA A  
SERVIÇOS COM BAIXO  
ACOPLAMENTO E BOUNDED  
CONTEXTS.

— Adrian Cockcroft, *Battery Ventures*

”

# PORQUE MICROSERVIÇOS?

**É MODA  
É COOL  
TODO MUNDO FAZ**



# CARACTERISTICAS

## DONEC QUIS NUNC

- 🍕 Serviços como componentes
- 🍕 Organizado em torno do negócio
- 🍕 Produtos e não projetos
- 🍕 Baixo acoplamento e alta coesão

# CARACTERISTICAS

## DONEC QUIS NUNC

- 🍕 Flexibilidade no uso de tecnologias
- 🍕 Tecnologias de BD diferentes
- 🍕 Automação de infraestrutura
- 🍕 Design para falhas
- 🍕 Design evolucionário

**MAS QUE PROBLEMA  
ESSA ARQUITETURA  
RESOLVE?**



**QUE PROBLEMAS  
EVITAMOS?**

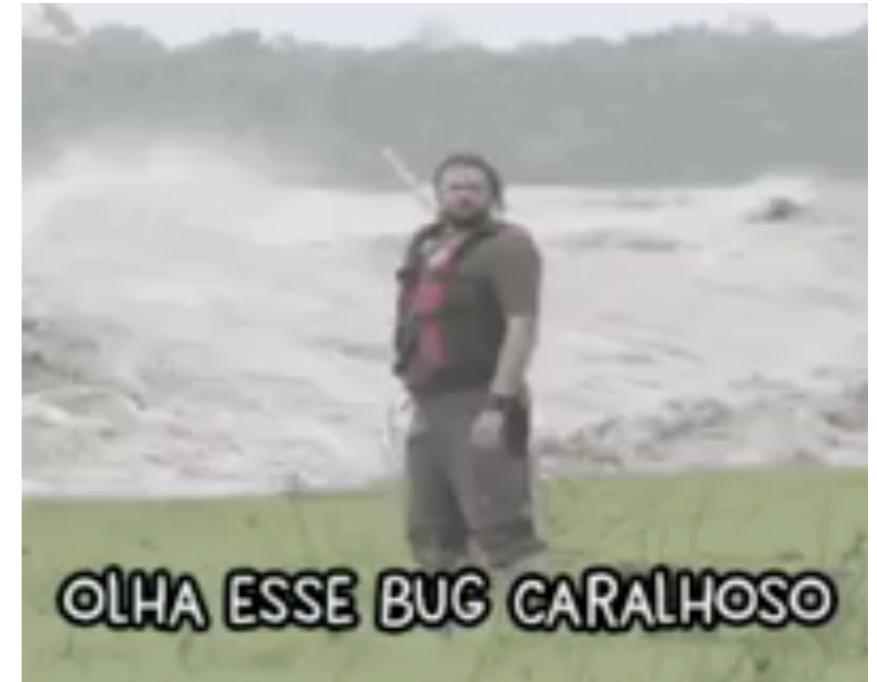
# QUE PROBLEMAS EVITAMOS

- ▶ Alto acoplamento
- ▶ Se algum componente falha, todo o sistema falha
- ▶ Mudanças são custosas
- ▶ Escalabilidade reduzida
- ▶ Deploys pesados
- ▶ Ciclo de desenvolvimento e alterações maiores

E O QUE GANHAMOS  
COM ISSO?

# O QUE GANHAMOS

- ▶ Agilidade
- ▶ Reduzir tamanho das releases
- ▶ Diminuir os impactos ao subir mudanças
- ▶ Facilitar coordenação e trabalho das equipes em bases de código separadas



# ANTES DE QUEBRAR SEU MONOLITO

Não vá querendo arrebentar  
tudo!



# COMO DIVIDIR O MONOLITO?

# **REGRA DO BURACO: PARE DE CAVAR**

Se você estiver em um buraco, pare de cavar. Se uma aplicação monolítica se tornar impossível de gerenciar, pare de torna-la maior.



# DDD - DOMAIN DRIVEN DESIGN

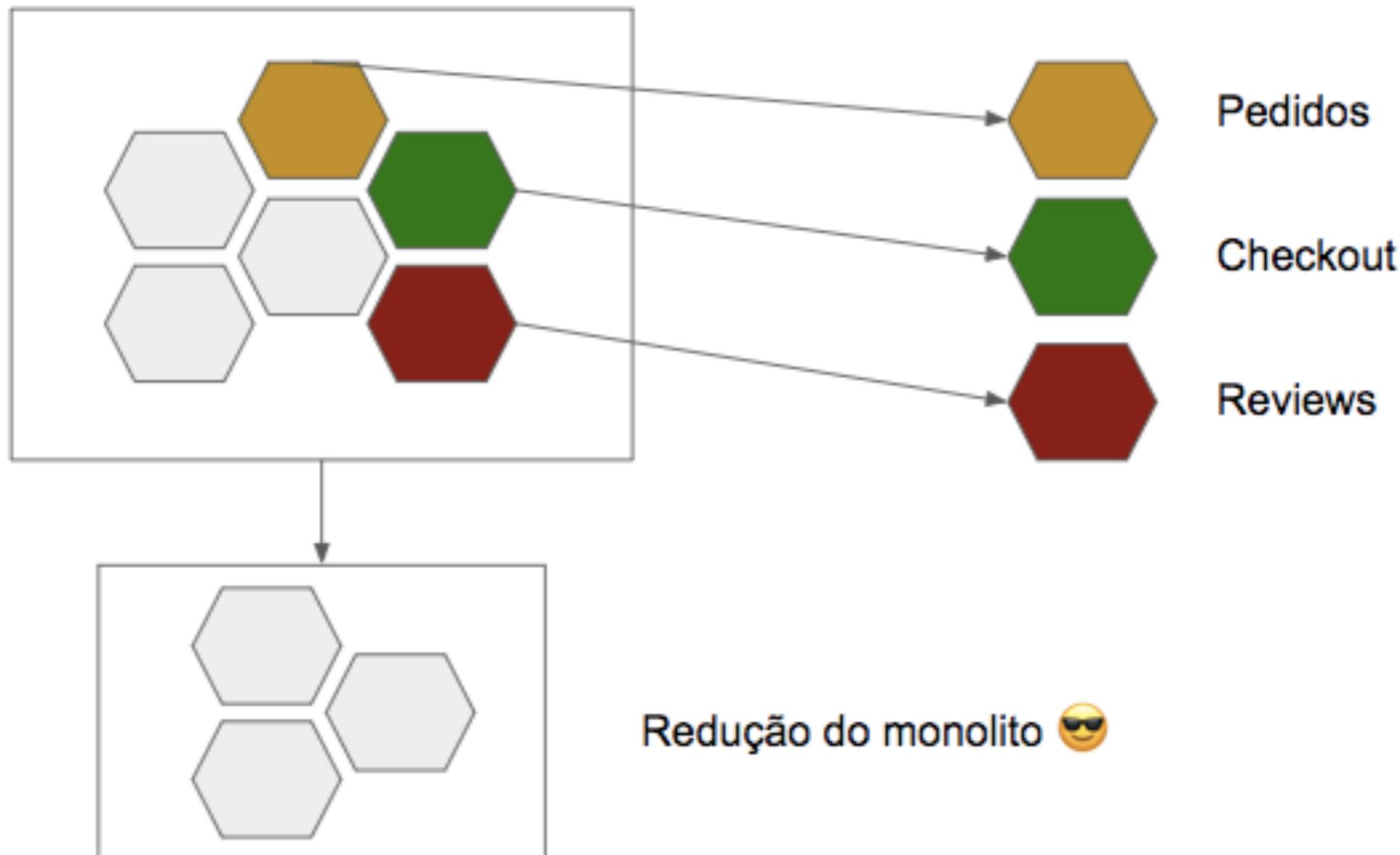
- 🍕 Microserviços são contextos
- 🍕 Dividimos um contexto específico do domínio em um artefato
- 🍕 Precisa de tempo pra evoluir
- 🍕 Bounded-contexts

# PADRÃO STRANGLER

Uma figueira que nasce na copa da árvore e cria raízes no chão, ela se desenvolve então ao ponto de estrangular a árvore hospedeira.



# APLICAÇÃO MONOLÍTICA EM MICROSERVIÇOS





## PRIMEIRO A LÓGICA

Antes de mover os dados, primeiro o código relacionado a regra de negócio.

## DEPOIS OS DADOS

Os dados devem ser a última coisa, fazer isso de forma precoce pode acarretar em retrabalho.



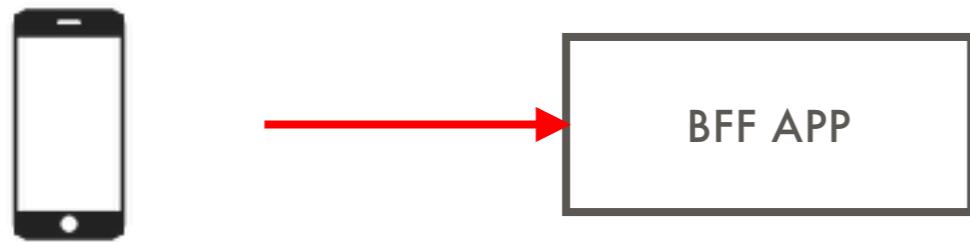
**SERVINDO OS  
CLIENTES DA API**

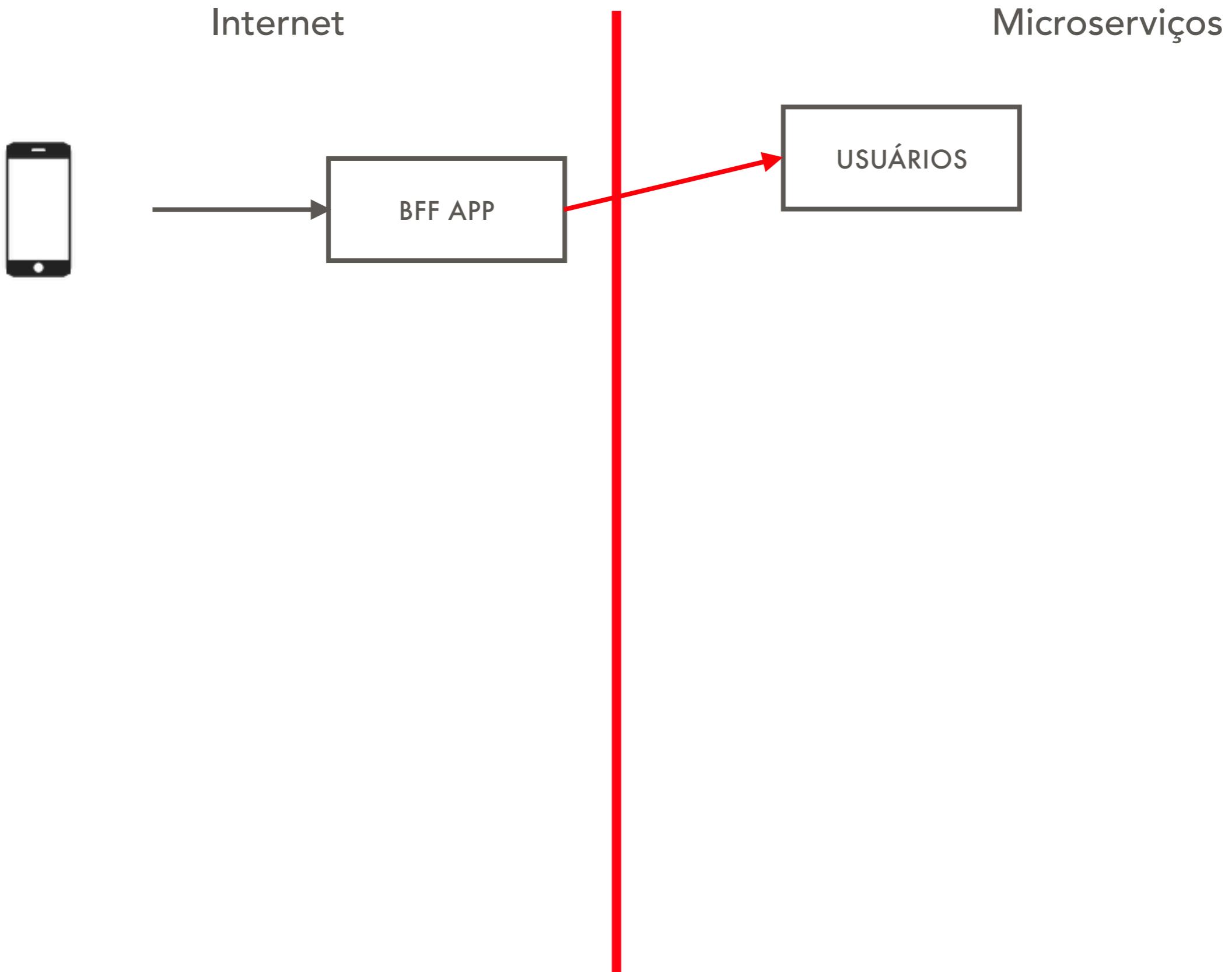


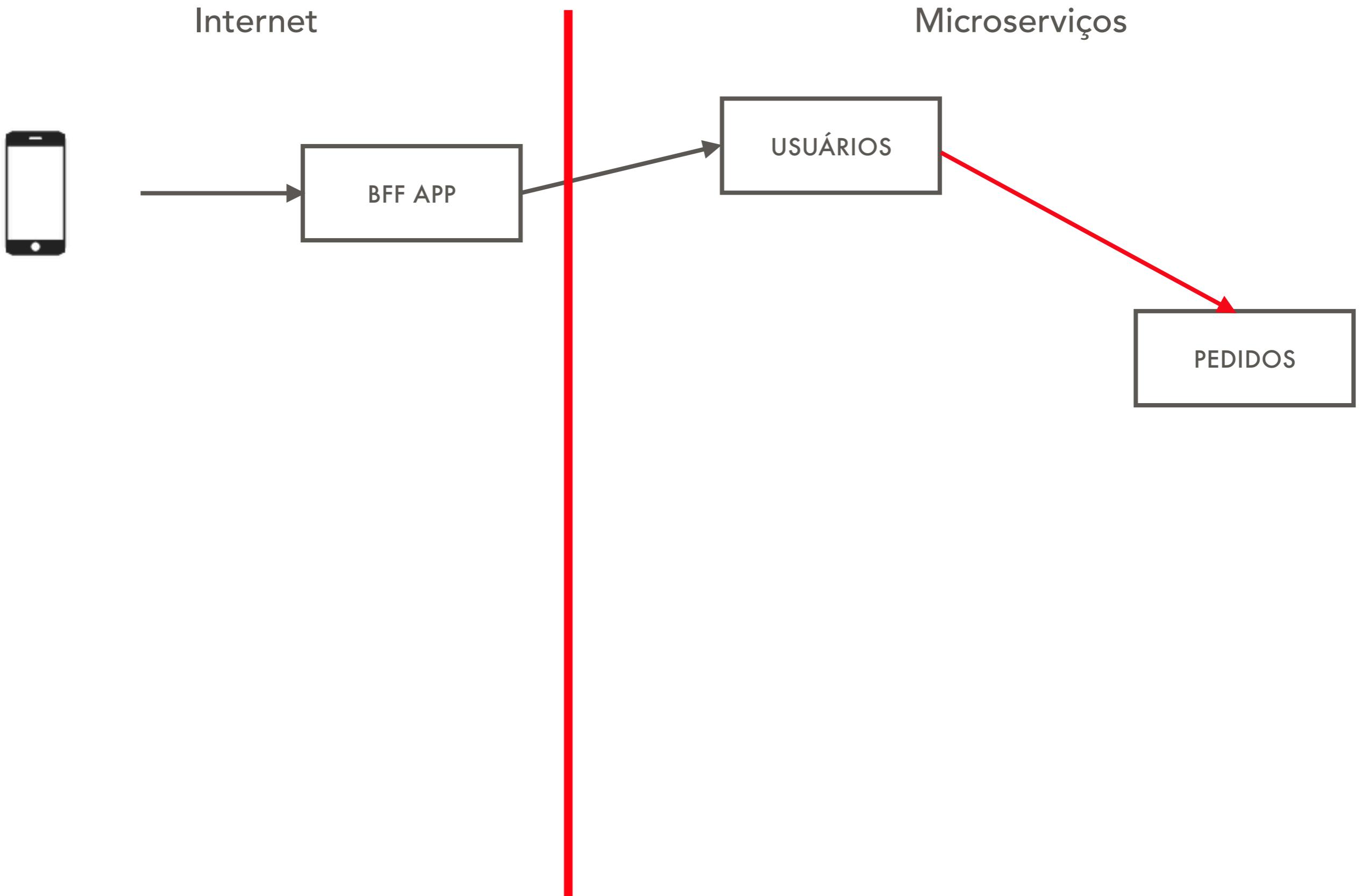
# BACKEND FOR FRONTENDS



Internet

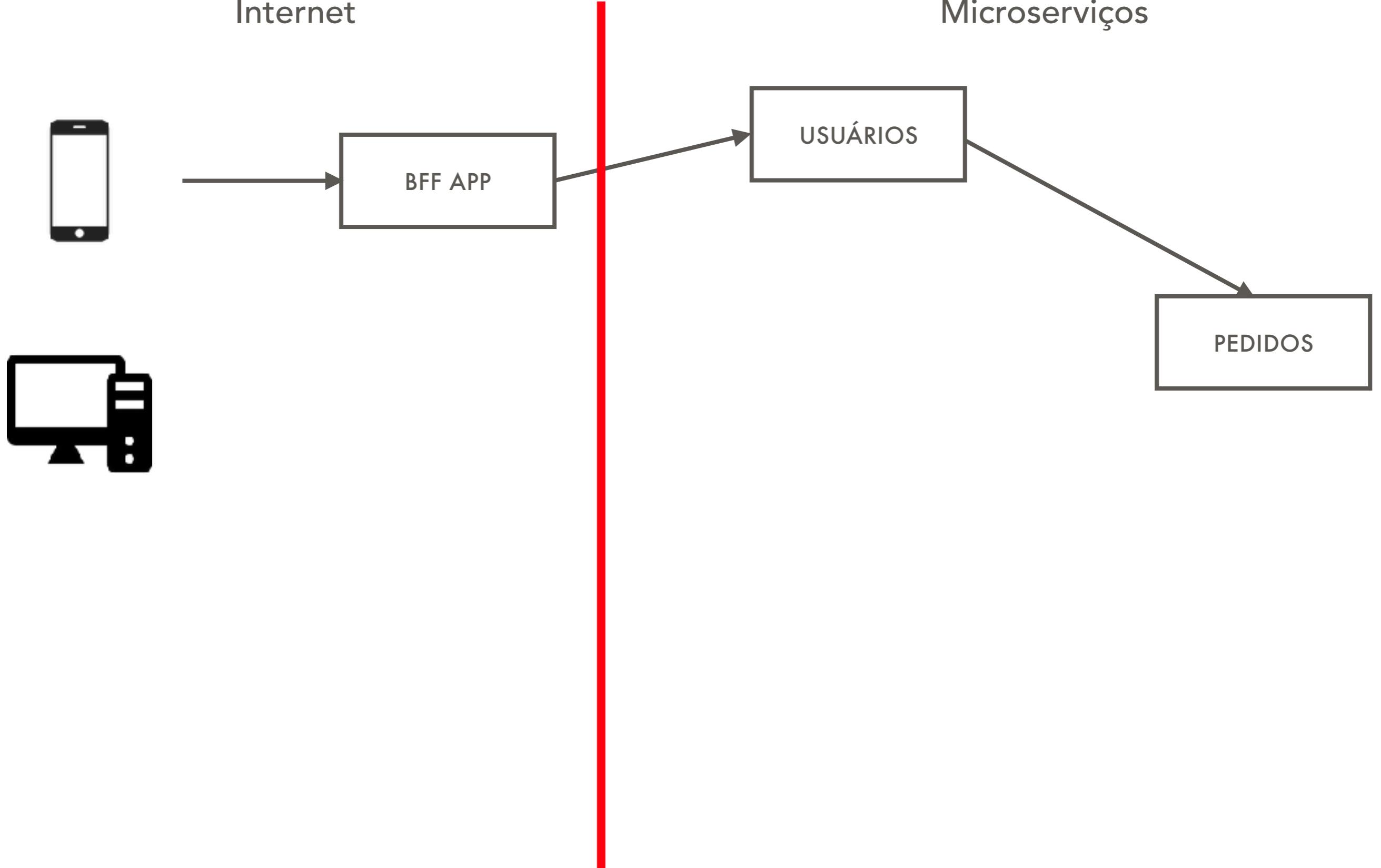






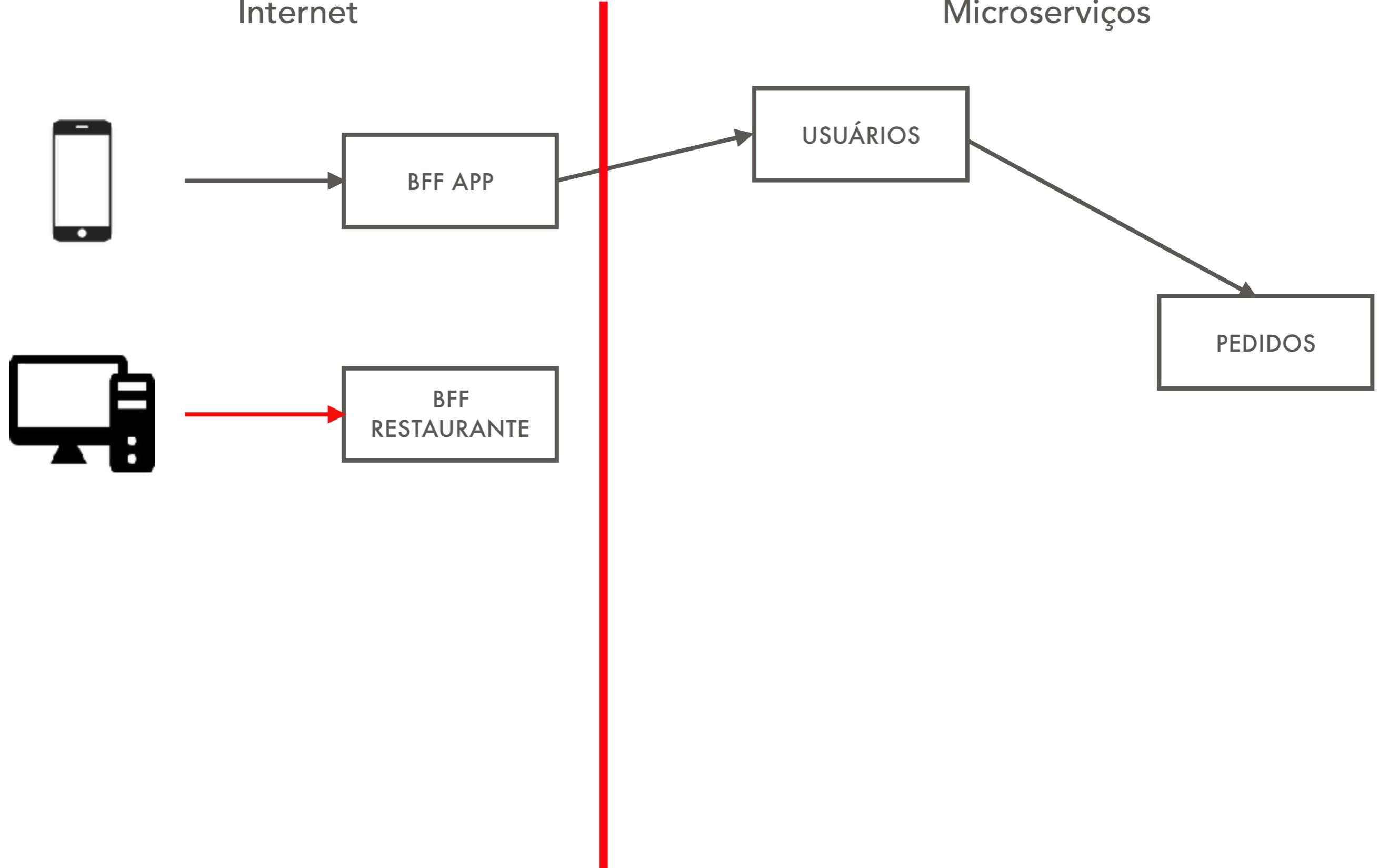
Internet

Microserviços



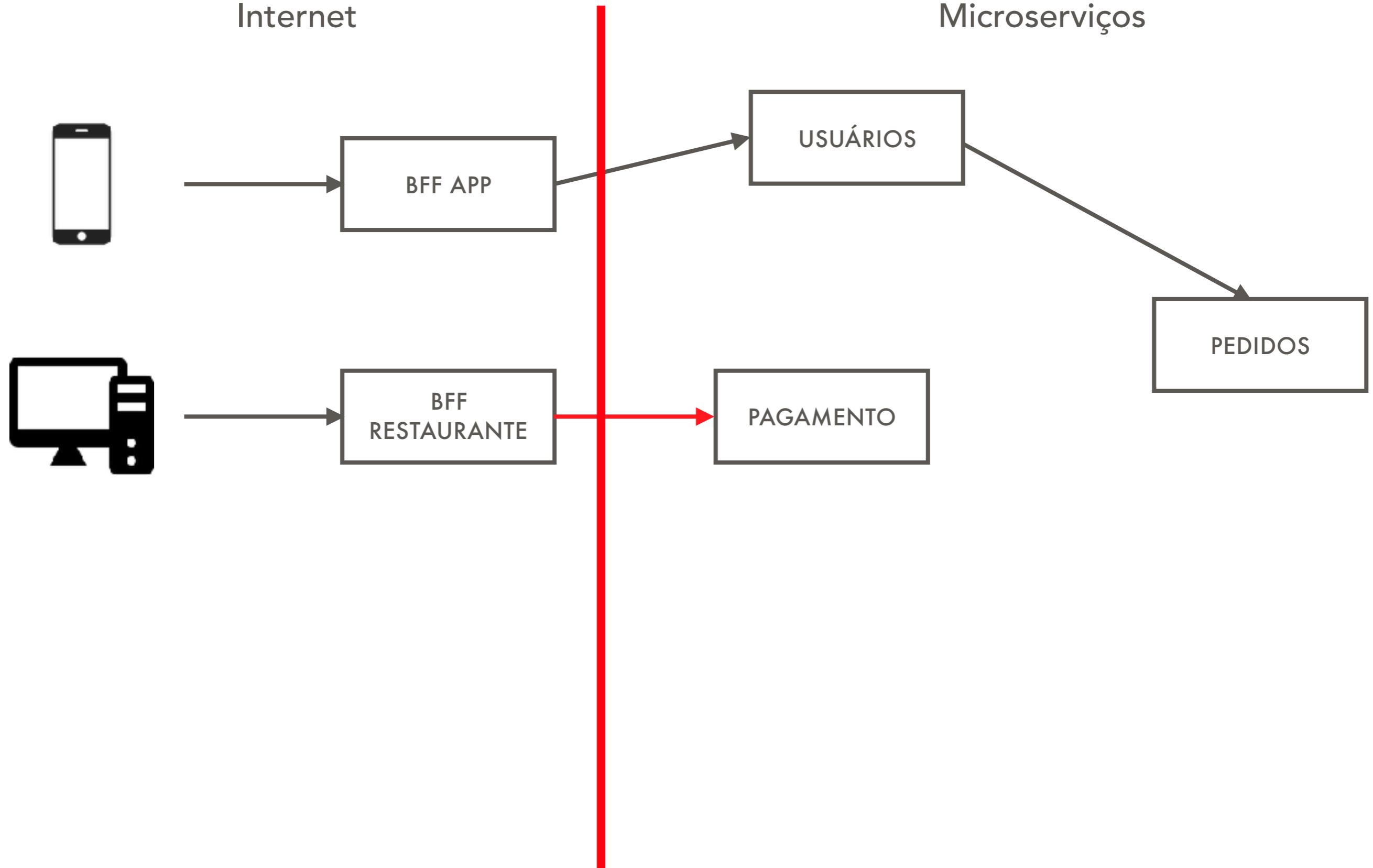
Internet

Microserviços



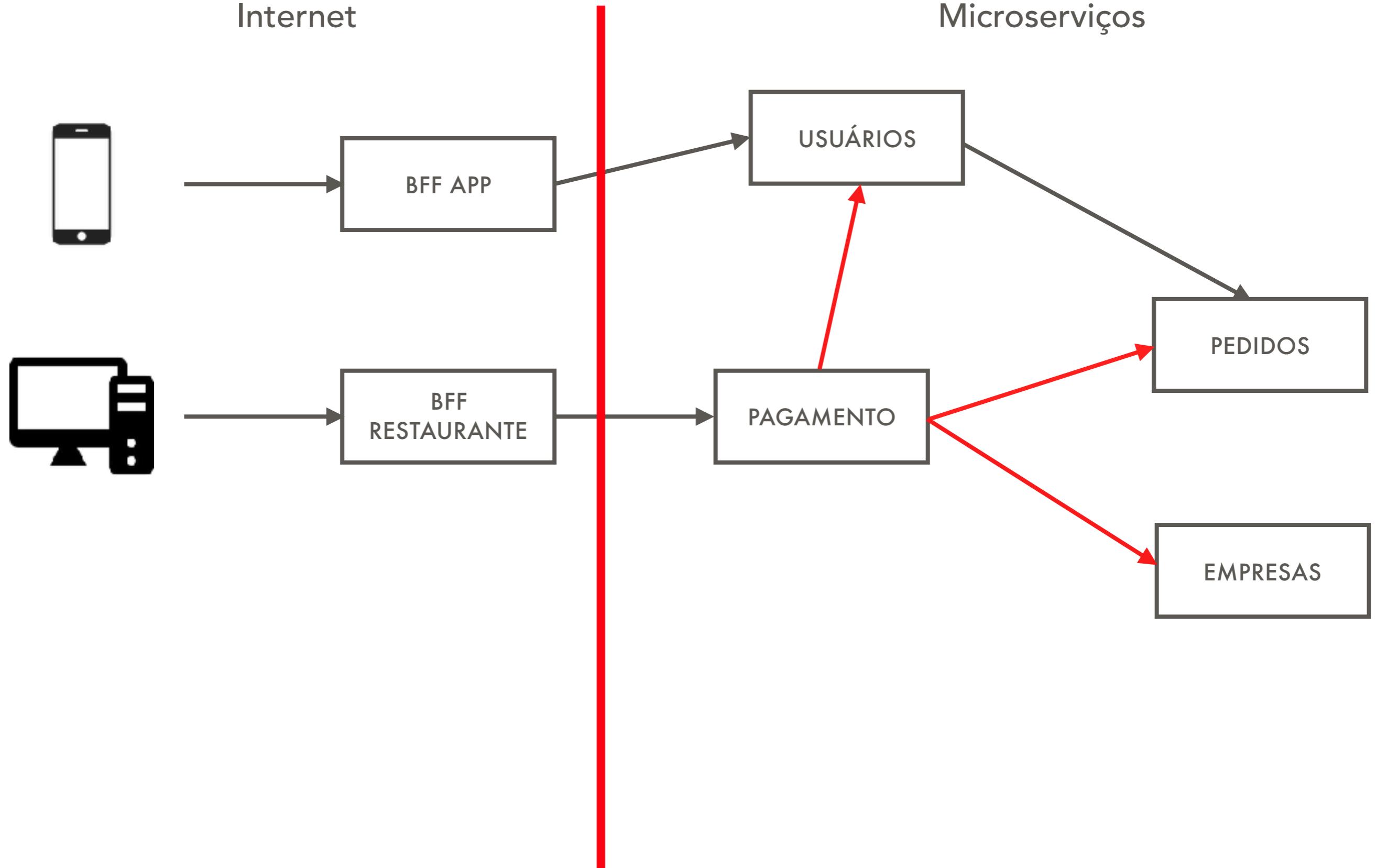
Internet

Microserviços



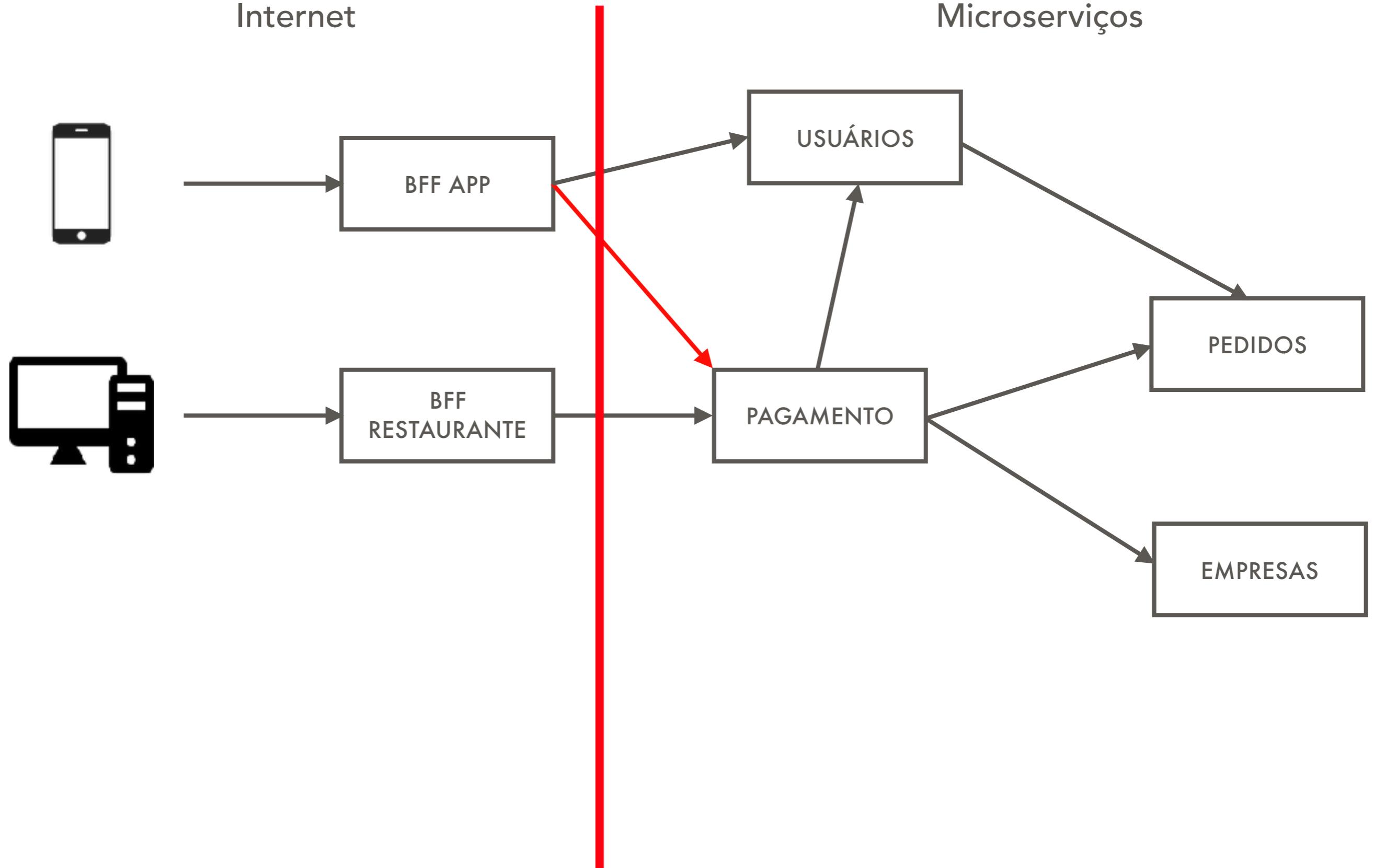
Internet

Microserviços



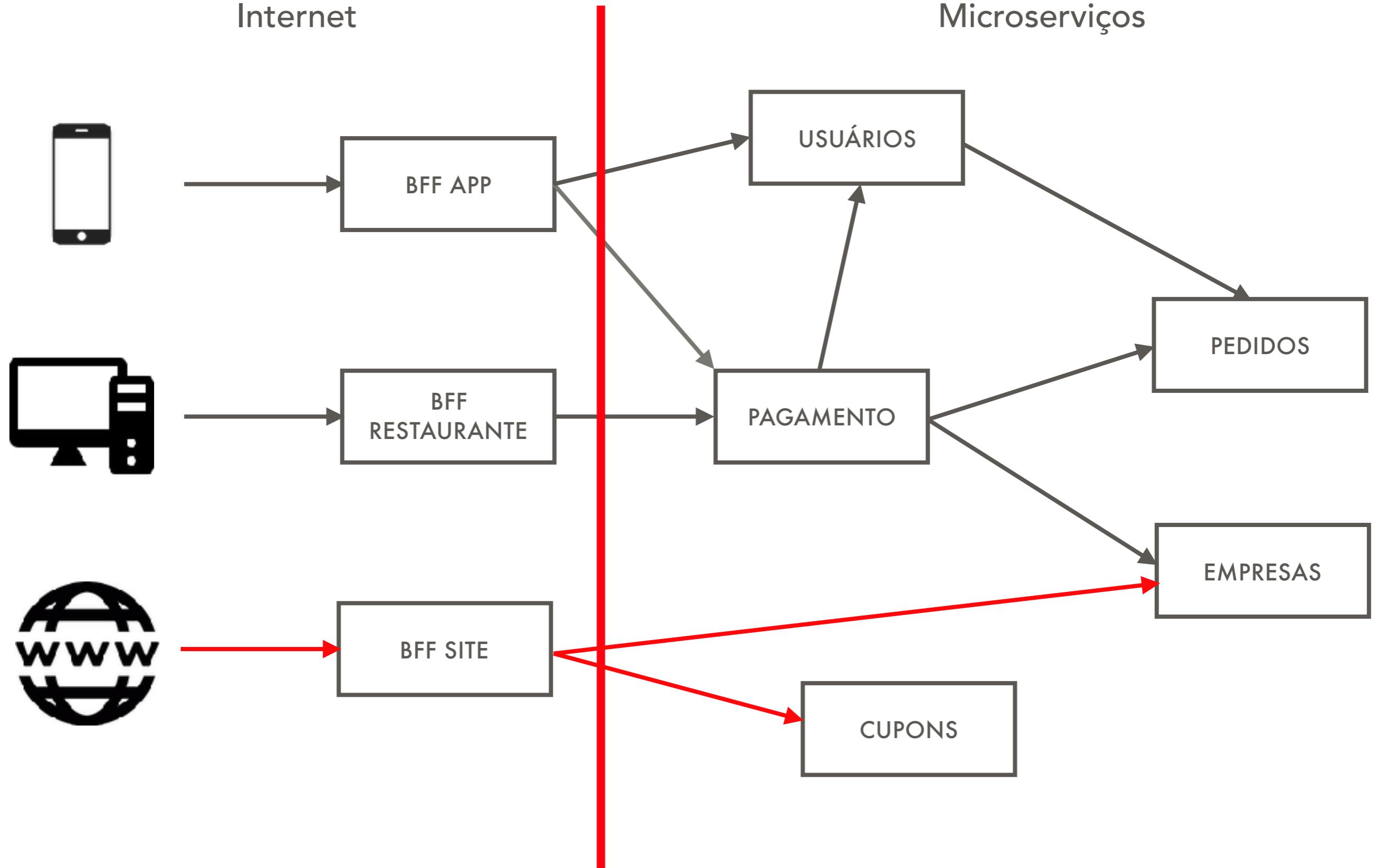
Internet

Microserviços

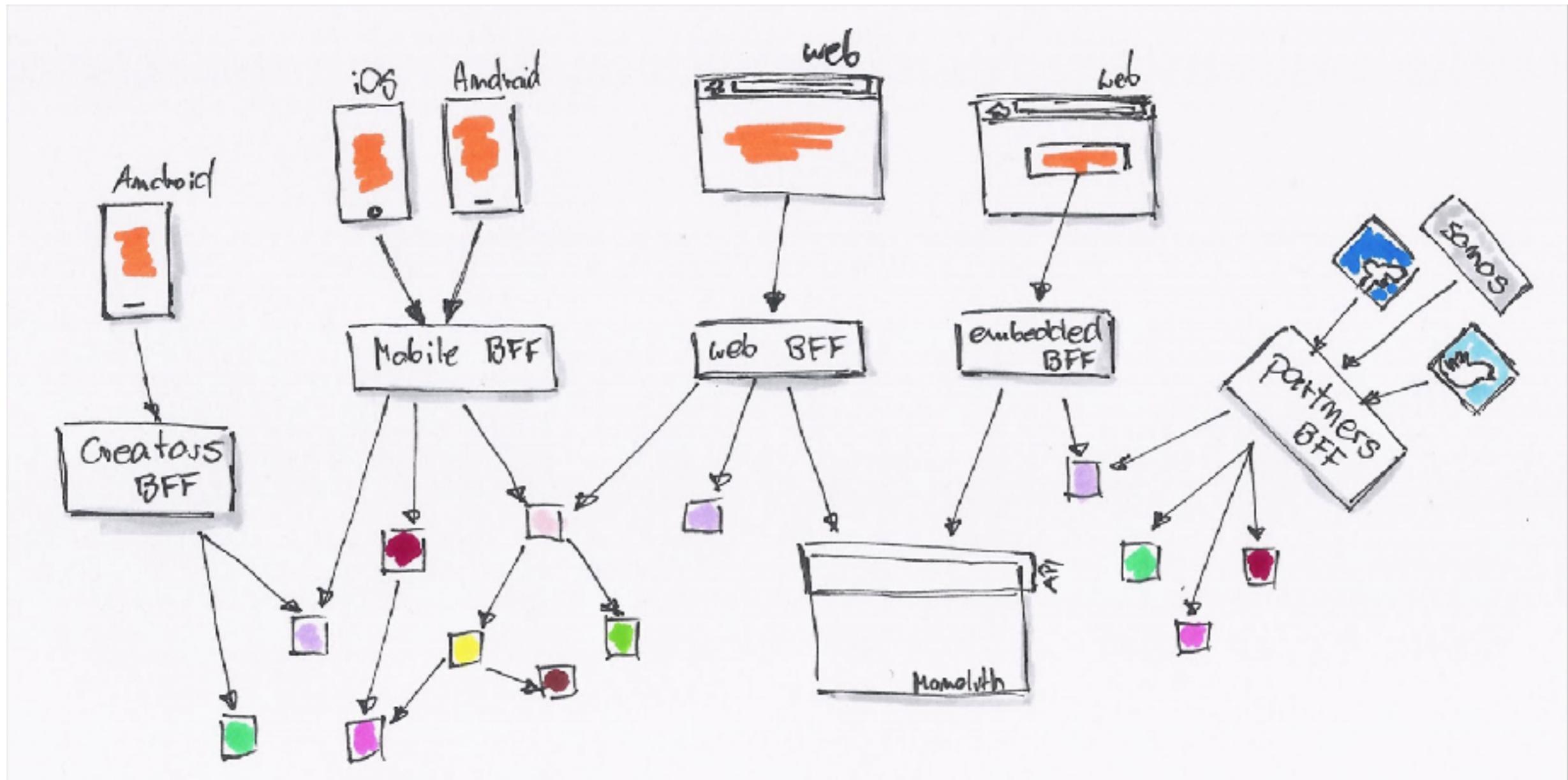


Internet

Microserviços



# BACKEND FOR FRONTENDS



Créditos: <https://www.thoughtworks.com/insights/blog/bff-soundcloud>

O QUE ESTAMOS  
PLANEJANDO

# O QUE ESTAMOS FAZENDO

- 🍕 Testes com Continuous Integration no GitLab
- 🍕 Implementando boas práticas:
  - 🍕 Execução de testes unitários
  - 🍕 Integrações após os testes passarem

# ARQUITETURA ORIENTADA A EVENTOS

# MOTIVOS

- 🍕 Desacoplar cliente dos serviços
- 🍕 Diminuição do impacto das mudanças
- 🍕 Adicionar novas features com mais agilidade
- 🍕 Maior resiliência

# MOTIVOS

- ◀ Buffer de mensagens
- ◀ Adiciona uma camada de back-pressure
- ◀ Permite escalar consumidores sem causar indisponibilidade

# TRADEOFF DA ARQUITETURA

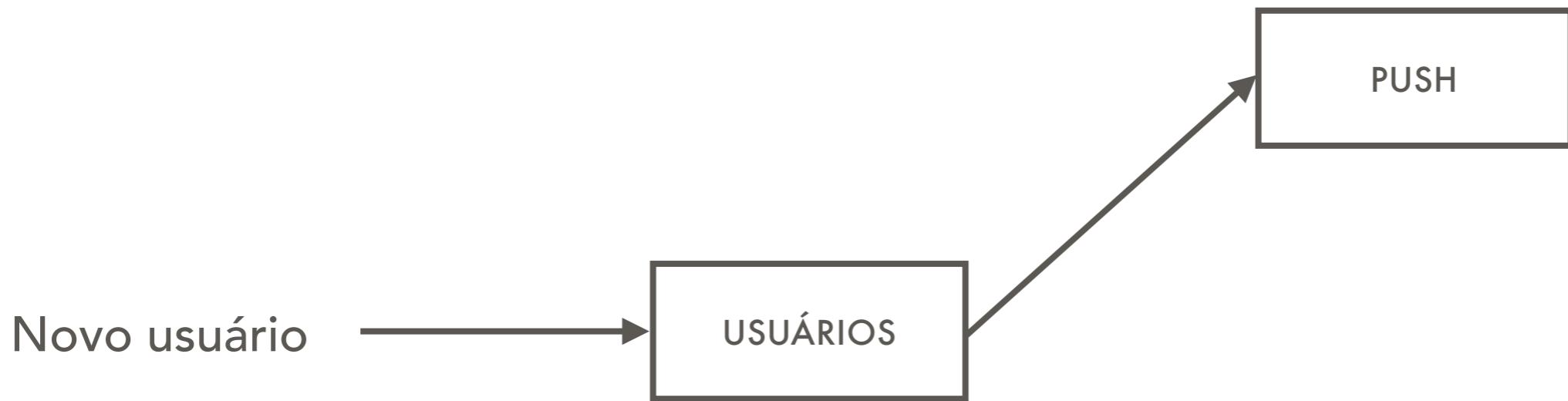
- 🍕 Complexidade operacional
- 🍕 Tratamento de erros
- 🍕 Ações compensatórias
- 🍕 Garantias transacionais
- 🍕 Consistência eventual

# ORQUESTRAÇÃO vs COREOGRAFIA

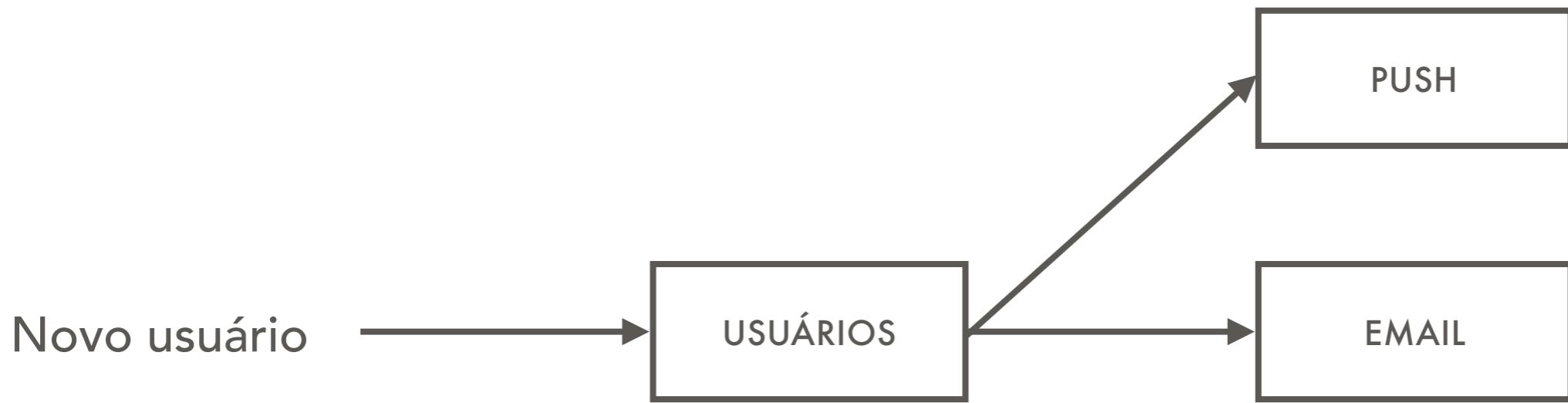
# ORQUESTRAÇÃO



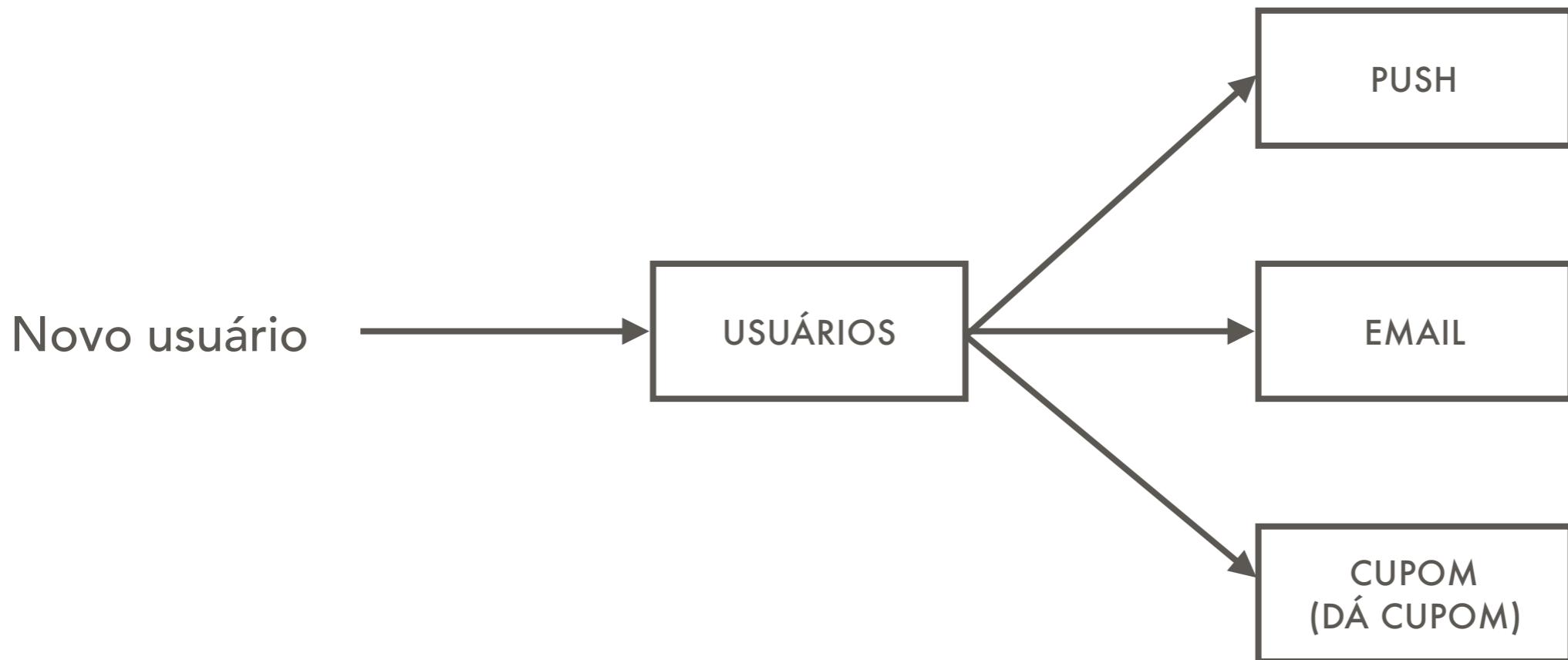
# ORQUESTRAÇÃO



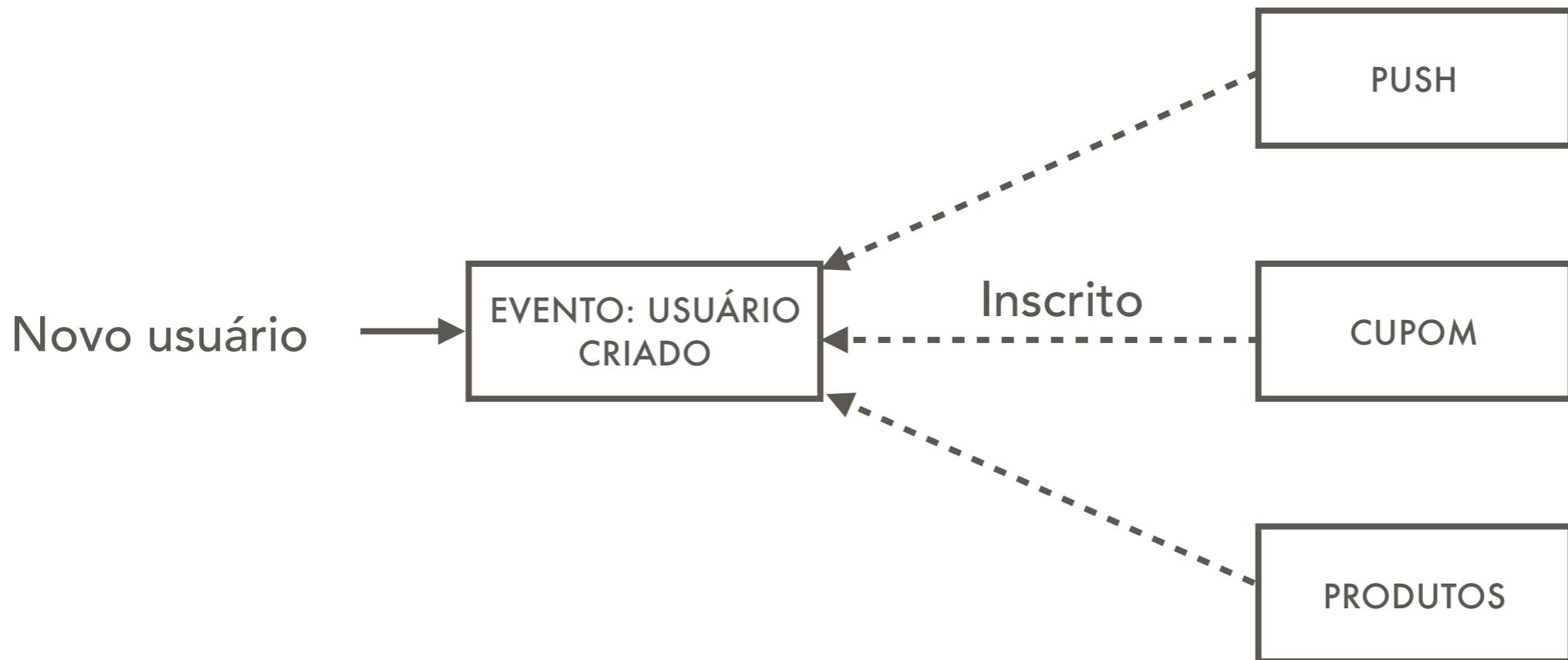
# ORQUESTRAÇÃO



# ORQUESTRAÇÃO



# COREOGRAFIA



## DONEC QUIS NUNC

- ◀ Using distributed transactions is usually not an option, and not only because of the CAP theorem. They simply are not supported by many of today's highly scalable NoSQL databases and messaging brokers. You end up having to use an eventual consistency-based approach, which is more challenging for developers.

# O QUE APRENDEMOS

- 🍕 Lei de Conway
- 🍕 Primeiro lógica, depois dados
- 🍕 Automatize trabalho repetitivo
- 🍕 Não aumente o monolito!!!!!!!!!!

“  
“QUALQUER EMPRESA QUE PROJETA UM  
SISTEMA, INEVITAVELMENTE PRODUZ UM  
PROJETO CUJA ESTRUTURA É UMA CÓPIA DA  
ESTRUTURA DE COMUNICAÇÃO DA  
ORGANIZAÇÃO.”

— *Melvin Conway*

”



CUPOM  
**#VALEUTIAGO**

OBRIGADO

ESTAMOS CONTRATANDO!!!

[tiago.lammers@deliverymuch.com.br](mailto:tiago.lammers@deliverymuch.com.br)