Eventos (Parte II) + Padrões Para APIs externas

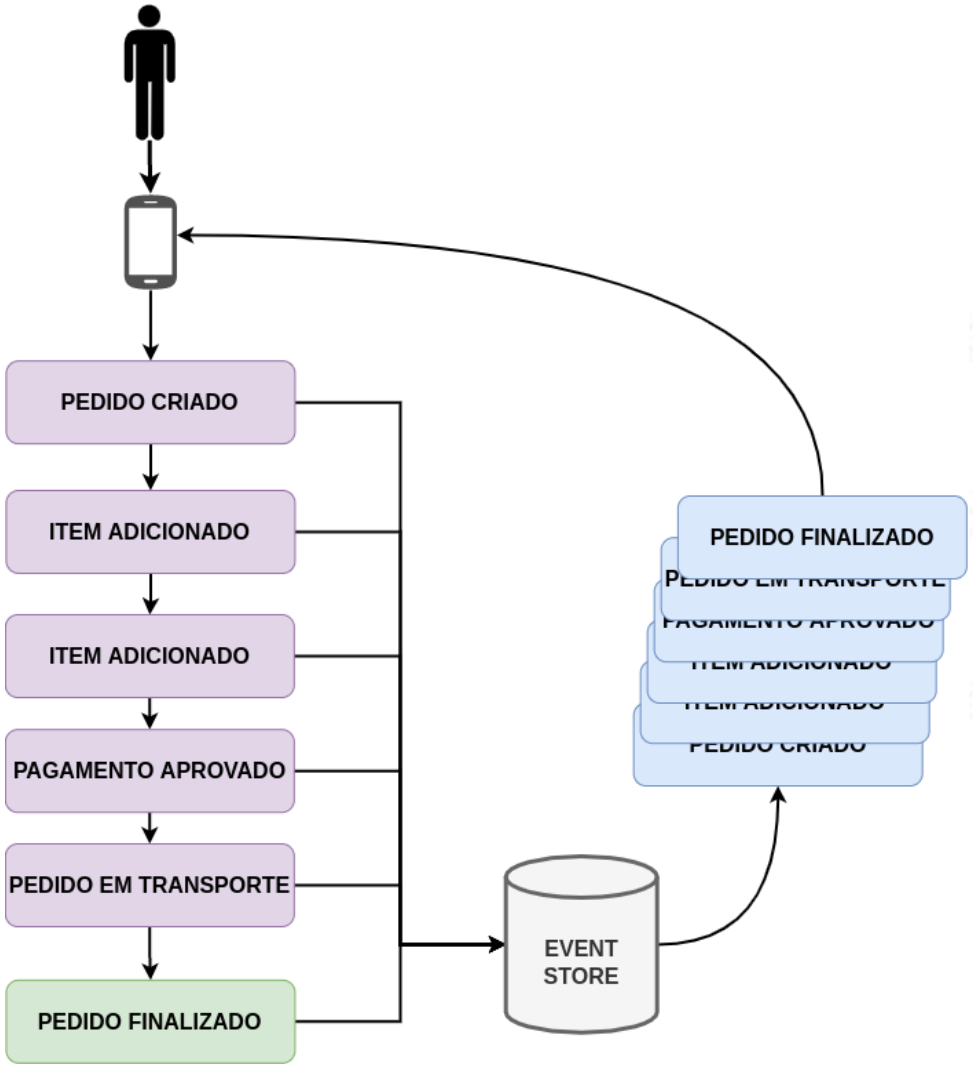
7COMm
Serviços e Soluções em TI

Event Sourcing (Parte II)

7COMm
Serviços e Soluções em TI

Event Sourcing

- Maneira de estruturar regras de negócio e persistência.
- Cada evento representa uma mudança de estado.
- O estado atual pode ser recriado.
- http://microservices.io/patterns/data/event-sourcing.html


PEDIDO CRIADO
ITEM ADICIONADO
ITEM ADICIONADO
PAGAMENTO APROVADO
PEDIDO EM TRANSPORTE
PEDIDO FINALIZADO
EVENT STORE
PEDIDO FINALIZADO

Event Store

- Híbrido entre um banco de dados e um message broker
  - Possui uma API para inserir e recuperar eventos por chave
  - É permitido subscribe em uma API de eventos
- Pode-se utilizar um RDBMS e implementar a estratégia de pooling.
- Pode-se utilizar um framework de mercado.

Event Store

- Event Store (https://eventstore.org)
- Lagom (http://www.lightbend.com/lagom-framework)
- Axon (http://www.axonframework.org)

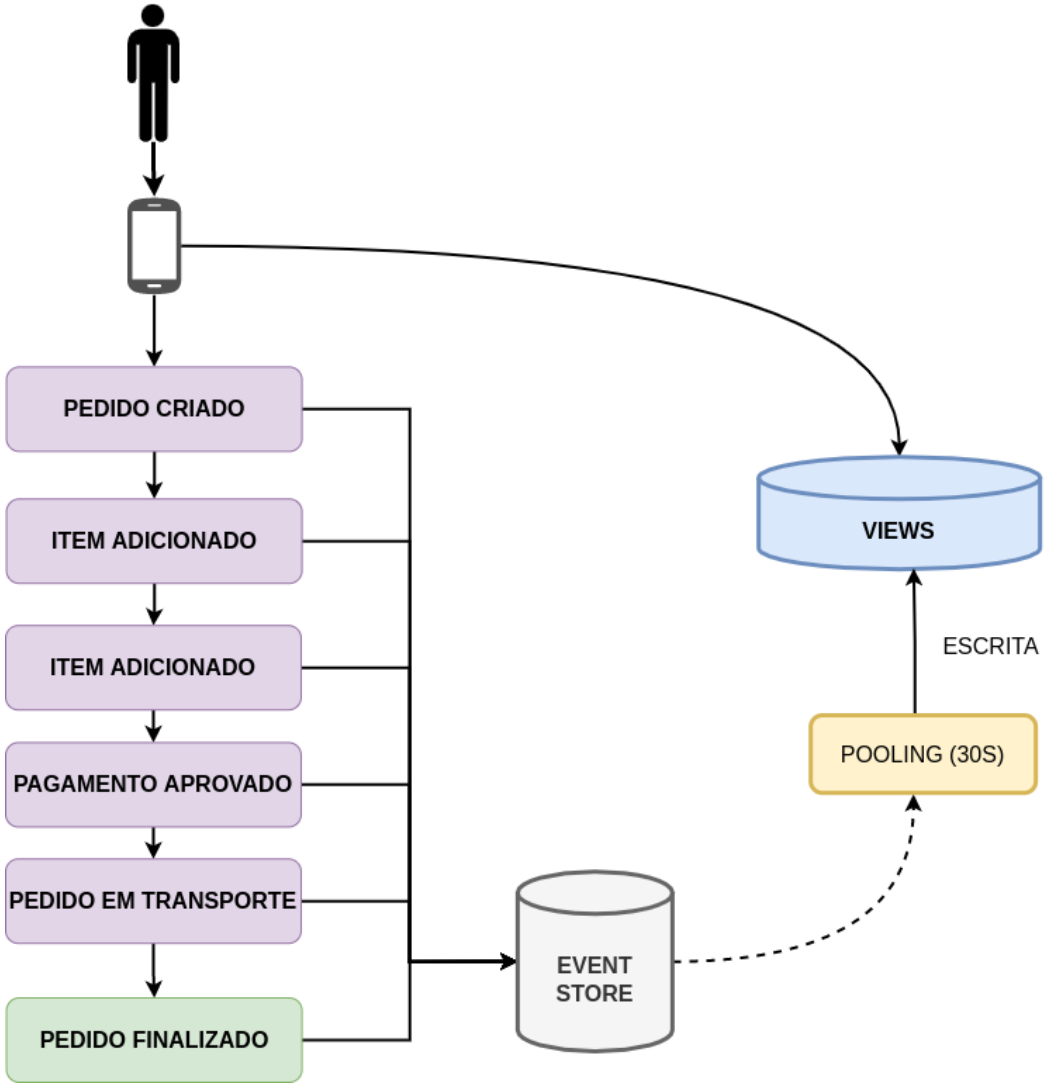
Event Sourcing e publicação de eventos

- Já verificamos em aulas anteriores maneiras de publicação de eventos
  - Pooling
  - Transaction logs

Pooling

- Pode-se salvar os eventos em uma tabela.
- O elemento interessado, deve, então, realizar buscas regularmente nesta tabela.
- SELECT * FROM EVENTS where event_id > ? ORDER BY event_id ASC.
- Problema dessa abordagem é que as transações podem ser comitadas em uma ordem diferente da ordem em que os eventos são gerados.

Pooling


PEDIDO CRIADO
ITEM ADICIONADO
ITEM ADICIONADO
PAGAMENTO APROVADO
PEDIDO EM TRANSPORTE
PEDIDO FINALIZADO
EVENT STORE
VIEWS
ESCRITA
POOLING (30S)

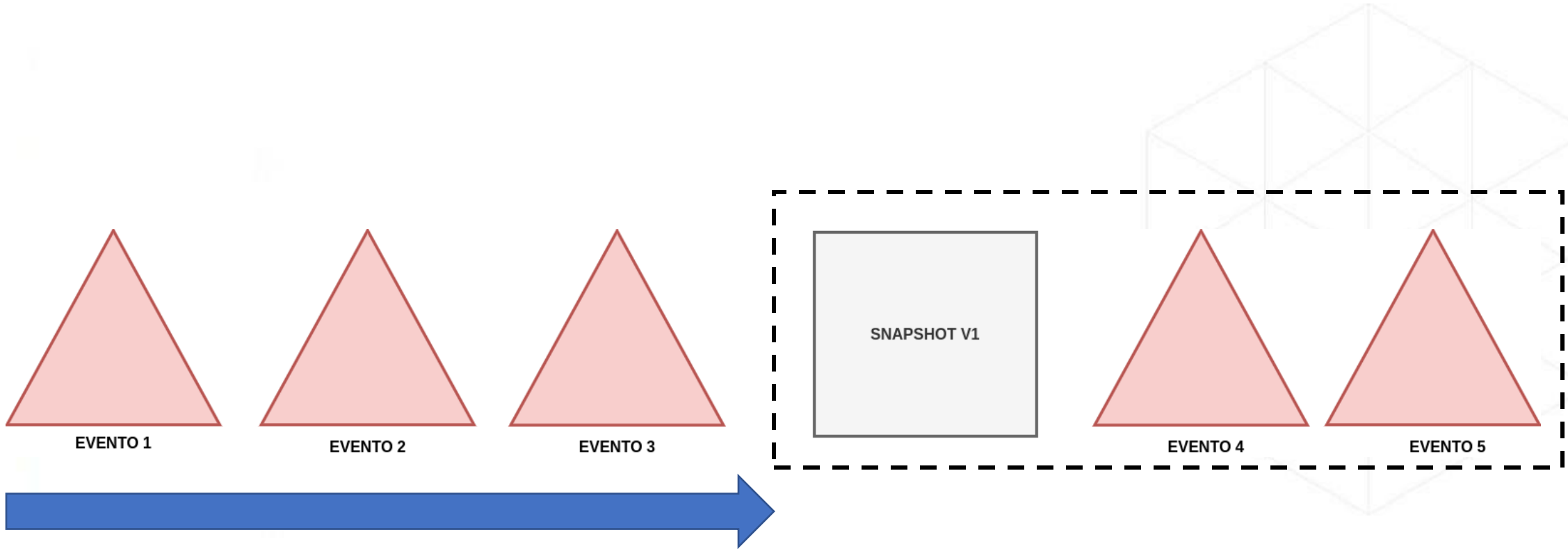
Transaction Log Tailing

- Lê os eventos da tabela e os publica em um message broker.
- Garante que eventos vão ser publicados e também tem mais performance e é mais escalável.

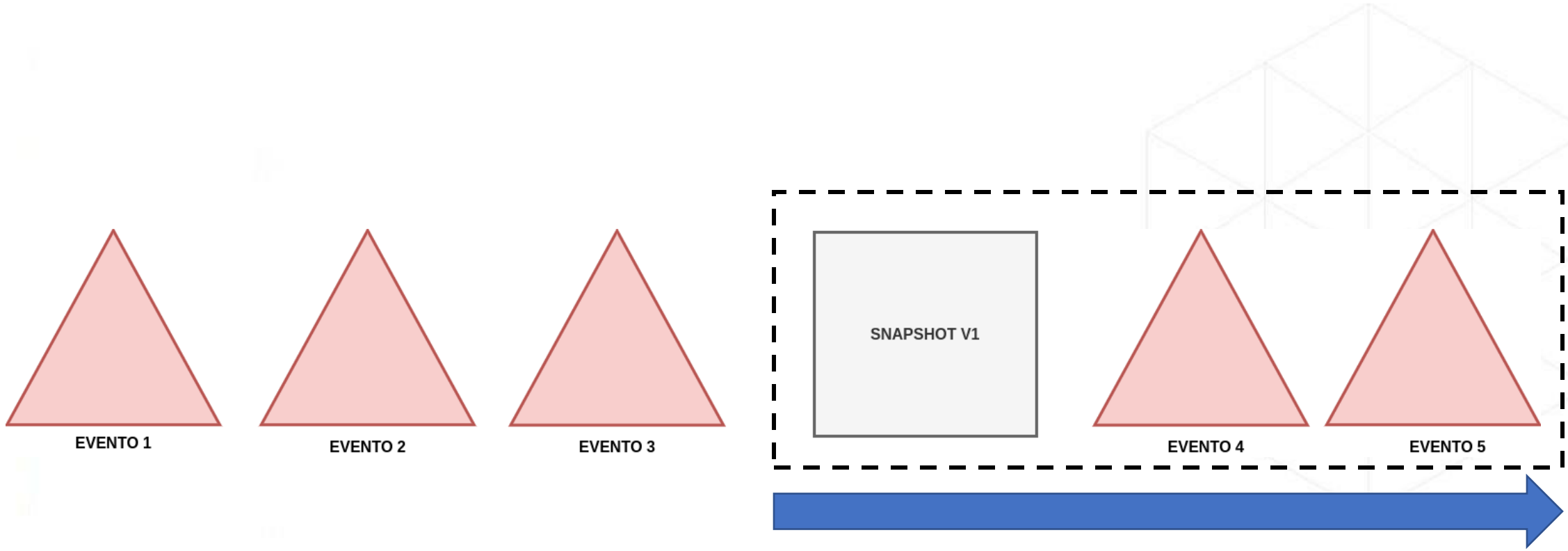
Snapshots

- Agregados com poucas transições de estados são restaurados de maneira simples.
- Agregados com vida longa e muitas transições podem ser problemáticos para restaurar através de eventos.
- Com o tempo seria extremamente ineficiente.
- Uma solução é o uso de snapshots.

Snapshots



Snapshots



Snapshots

- No exemplo foi criado um snapshot.
- É uma escrita de todos os eventos até então.
- Para se restaurar o estado atual, utilizaria este snapshot + a aplicação de dois eventos.

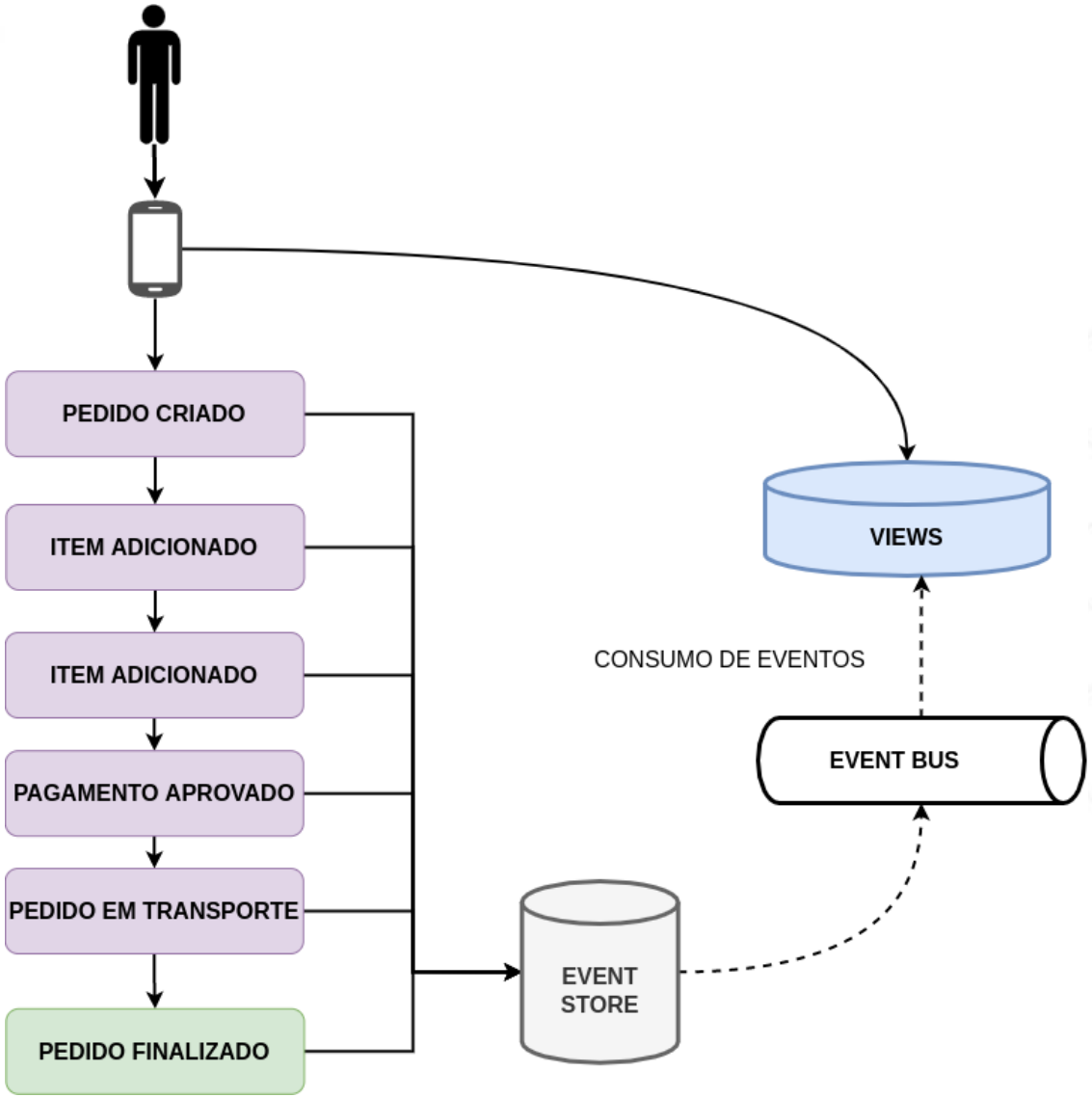
Snapshots

EVENTS

event_id	event_type	entity_type	entity_id	event_data
...	...	...	...	...
103	...	Customer	101	{...}
104	Credit Reserved	Customer	101	{...}
105	Address Changed	Customer	101	{...}
106	Credit Reserved	Customer	101	{...}

SNAPSHOTS

event_id	entity_type	event_id	snapshot_data
...	...	...	...
103	Customer	101	{name: "..." , ...}
...	...	...	...
...	...	...	...


PEDIDO CRIADO
ITEM ADICIONADO
ITEM ADICIONADO
PAGAMENTO APROVADO
PEDIDO EM TRANSPORTE
PEDIDO FINALIZADO
VIEWS
CONSUMO DE EVENTOS
EVENT BUS
EVENT STORE

Benefícios de Event Sourcing

- Publica eventos confiáveis.
- Preserva a história dos agregados.
- Provê uma *máquina do tempo*.
- Evita o problema de incompatibilidade de impedância objeto-relacional.

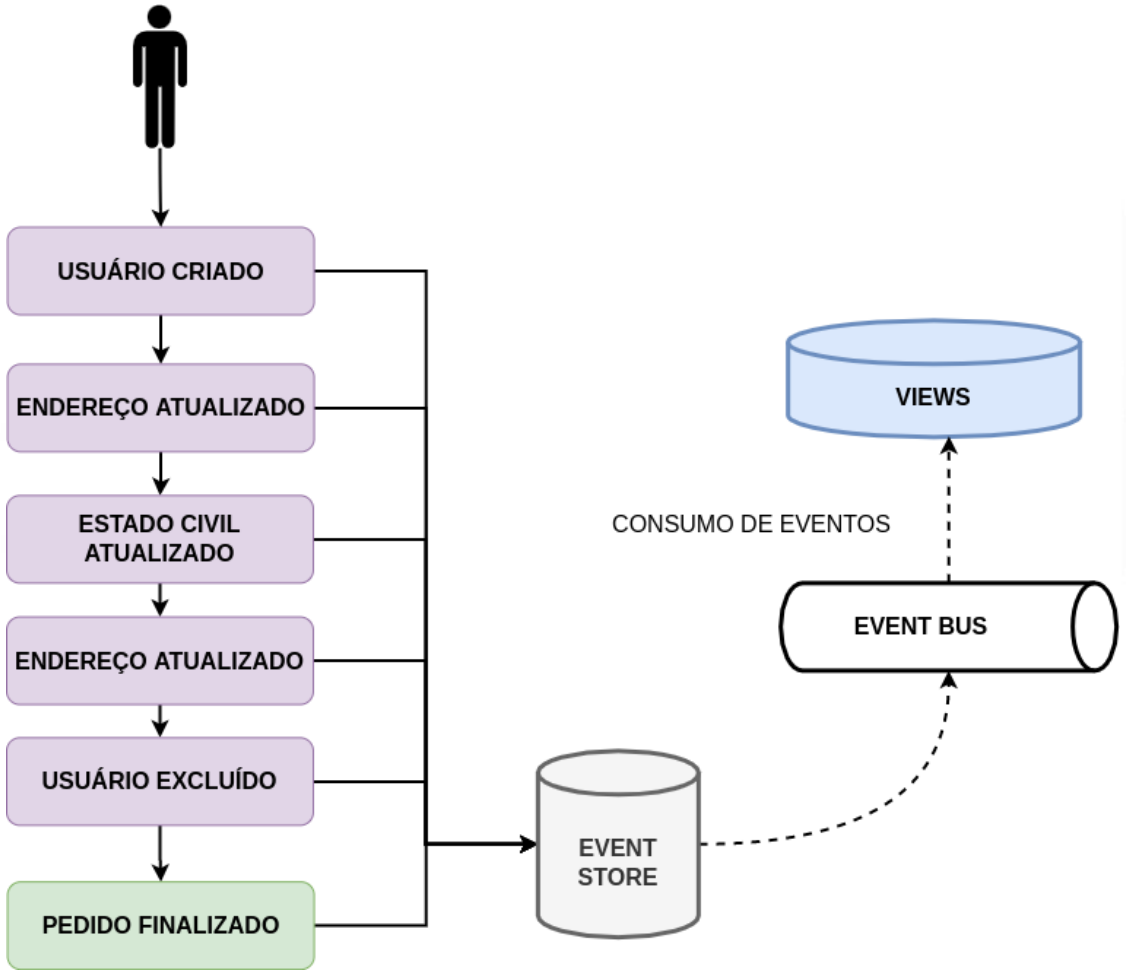
Pontos negativos de Event Sourcing

- Paradigma de linguagem diferente do convencional
  - Pode ser utilizado durante a conversão da aplicação em microsserviços
- Curva de aprendizado maior.
- Evolução dos eventos pode ser problemática de tratar.
- Buscas na tabela de eventos é complicado, uma vez que os agregados devem ser *reconstruídos* no select.

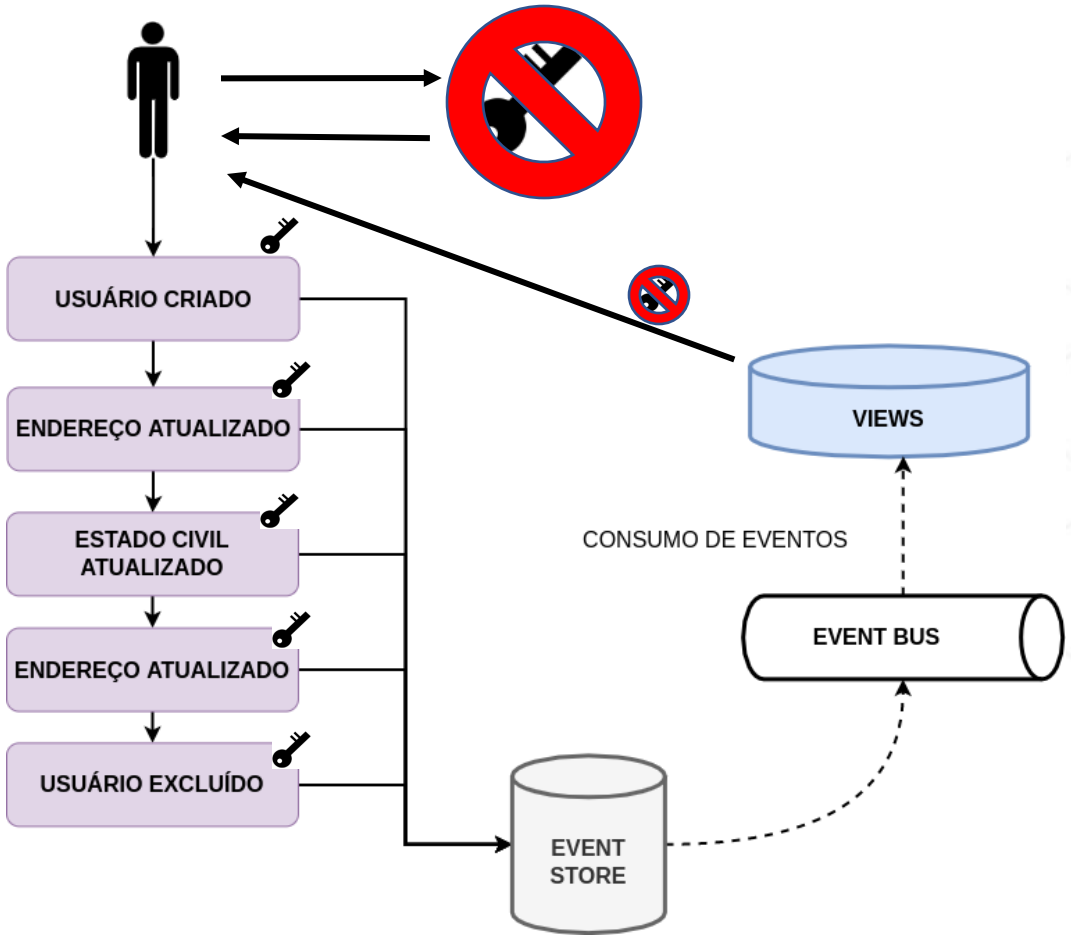
Exclusão de dados com Event sourcing

- Deve ser tratado de maneira especial.
- Um dos objetivos de Event Sourcing é gravar dados **pra sempre**.
- Pode-se utilizar *soft delete*, através de flags

Exclusão de dados com Event sourcing



Exclusão de dados SENSÍVEIS com Event sourcing



Padrões para APIs Externas

7COMm
Serviços e Soluções em TI

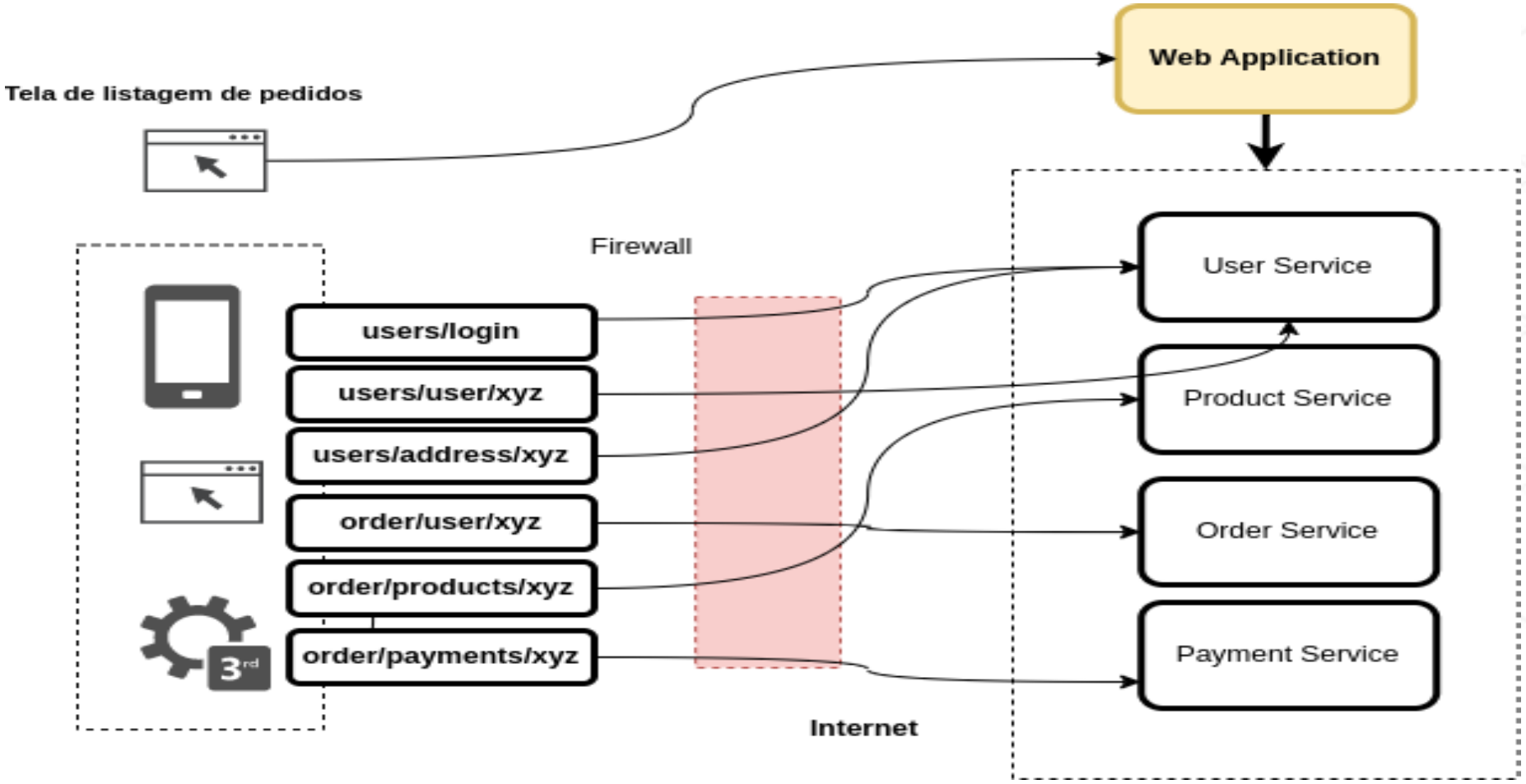
Padrões para APIs externas

- Comunicação de serviços com o mundo externo
- Como abrir suas APIs externamente (para terceiros, navegadores, mobiles, etc)?
- Deve-se expor cada um dos serviços individualmente? Deve-se unificá-los?

Tipos de APIs externas

- Frontend
- Javascript
- Mobile
- Terceiros

Podemos acessar os serviços diretamente?



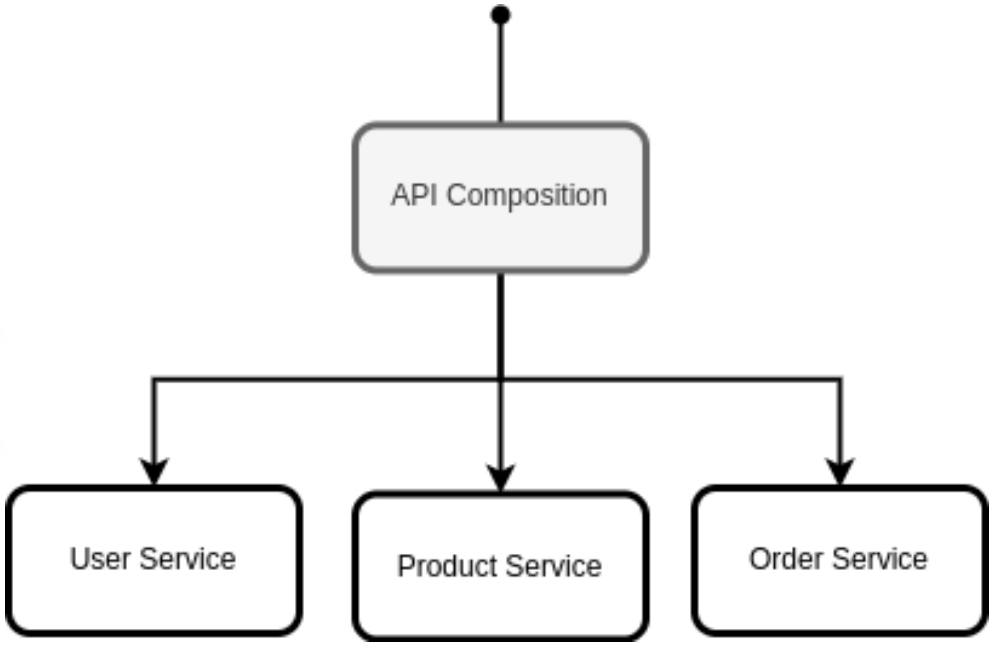
Desenhando APIs para clientes mobile

- Devemos sempre ter em mente o perfil do consumidor da API
- Sempre que possível, entregar **EXATAMENTE** o que o client precisa
- Não é ideal obrigar o cliente a fazer diversos *Requests*

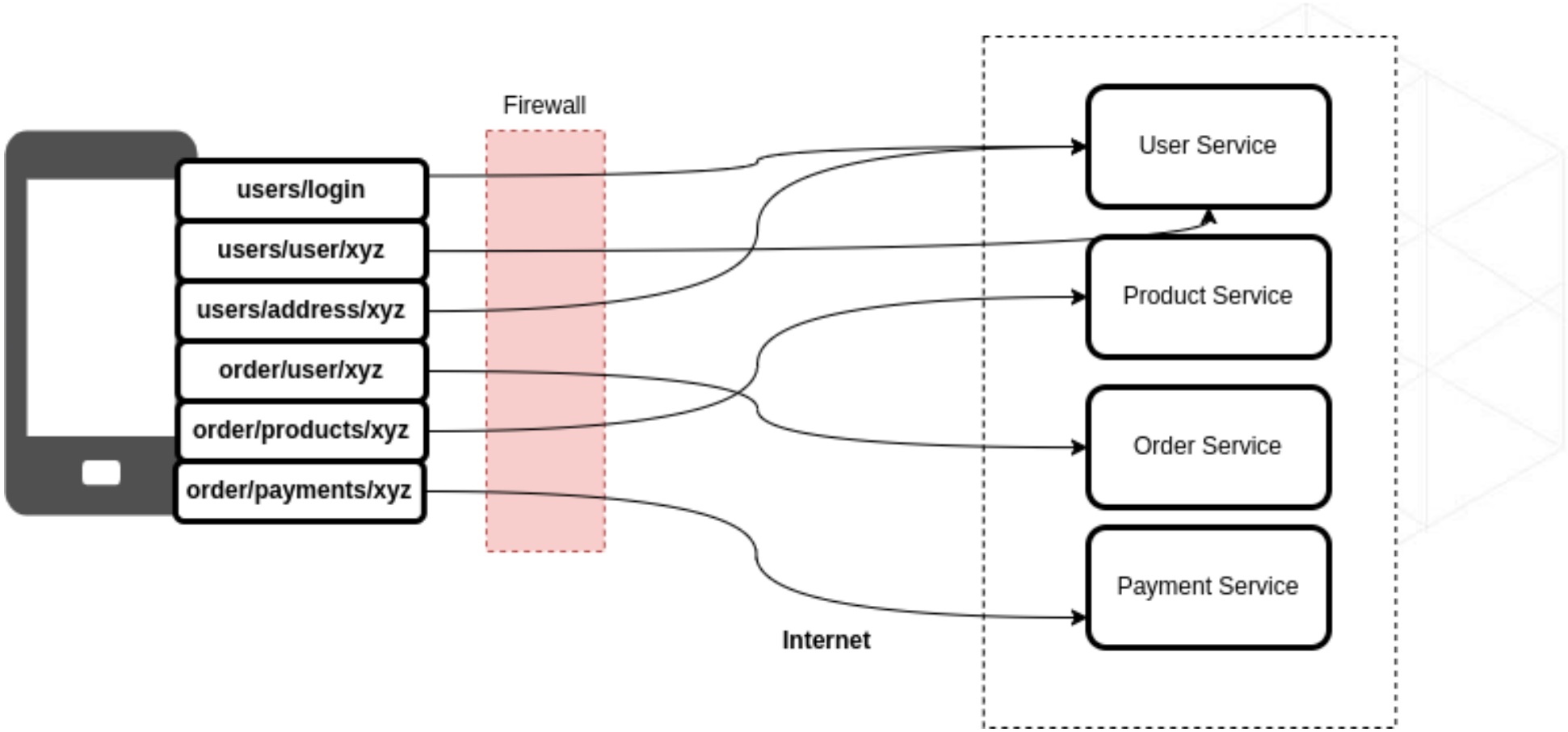
API Composition

- https://microservices.io/patterns/data/api-composition.html
- Faz os chamados nos serviços correspondentes
- Centraliza as informações
- Faz o *merge* em memória

API Composition



Desenhando APIs para clientes mobile



Desenhando APIs para clientes mobile

- Na imagem temos o papel de API Composer sendo realizado pelo próprio Client
- Este papel é o responsável por diversos requests e pelos *merges* de cada request

Experiência Ruim com múltiplos Requests

- Pode parecer que a aplicação é *lenta*
- O desenvolvedor mobile investe um esforço grande no papel de API Composer, ao invés da experiência do usuário
- Desenvolvedor *frontend* mais próximo do *backend*
- Serviços podem utilizar enlaces diferentes (REST, mensageria, gRPC e outros)
- Alto acoplamento entre *frontend* e *backend*

Desenhando APIs para clientes externos

- Não se sabe a natureza da utilização
- APIs **estáveis** são a regra
- Clientes externos **não possuem a disponibilidade** para alterar sua conexão como em um time próprio
- É comum manter várias versões da API em produção ao mesmo tempo

Desenhando APIs para clientes externos

- Deve-se evitar o acesso a API dos serviços diretamente
- Um time deve ser designado para desenvolver uma API pública, responsável por coordenar os acessos aos serviços

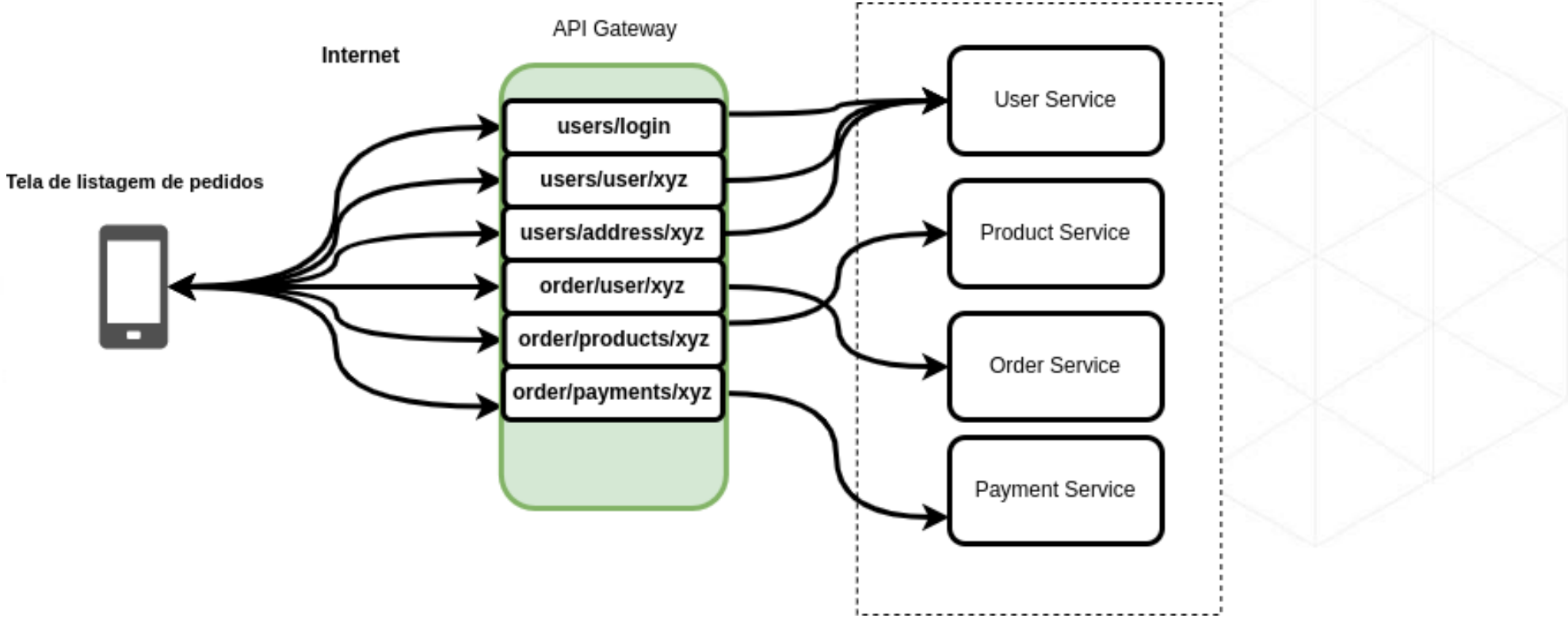
API Gateway

7COMm
Serviços e Soluções em TI

API Gateway

- http://microservices.io/patterns/apigateway.html
- Serviço responsável por ser o ponto de entrada dos clients do mundo externo
- É responsável por rotear os requests
- Autenticação e autorização
- Rate Limit
- Request Logging
- ...

API Gateway


API Gateway
Internet
Tela de listagem de pedidos
users/login
users/user/xyz
users/address/xyz
order/user/xyz
order/products/xyz
order/payments/xyz
User Service
Product Service
Order Service
Payment Service

API Gateway

- A abordagem anterior resolve um dos problemas
- O problema com a quantidade de informações, merge e orquestração de requests continua

API Gateway – Edge Functions

- Funções periféricas da requisição
- Autenticação
- Autorização
- Rate Limit
- Cache
- Coleta de métricas
- Log de Requests
- ...

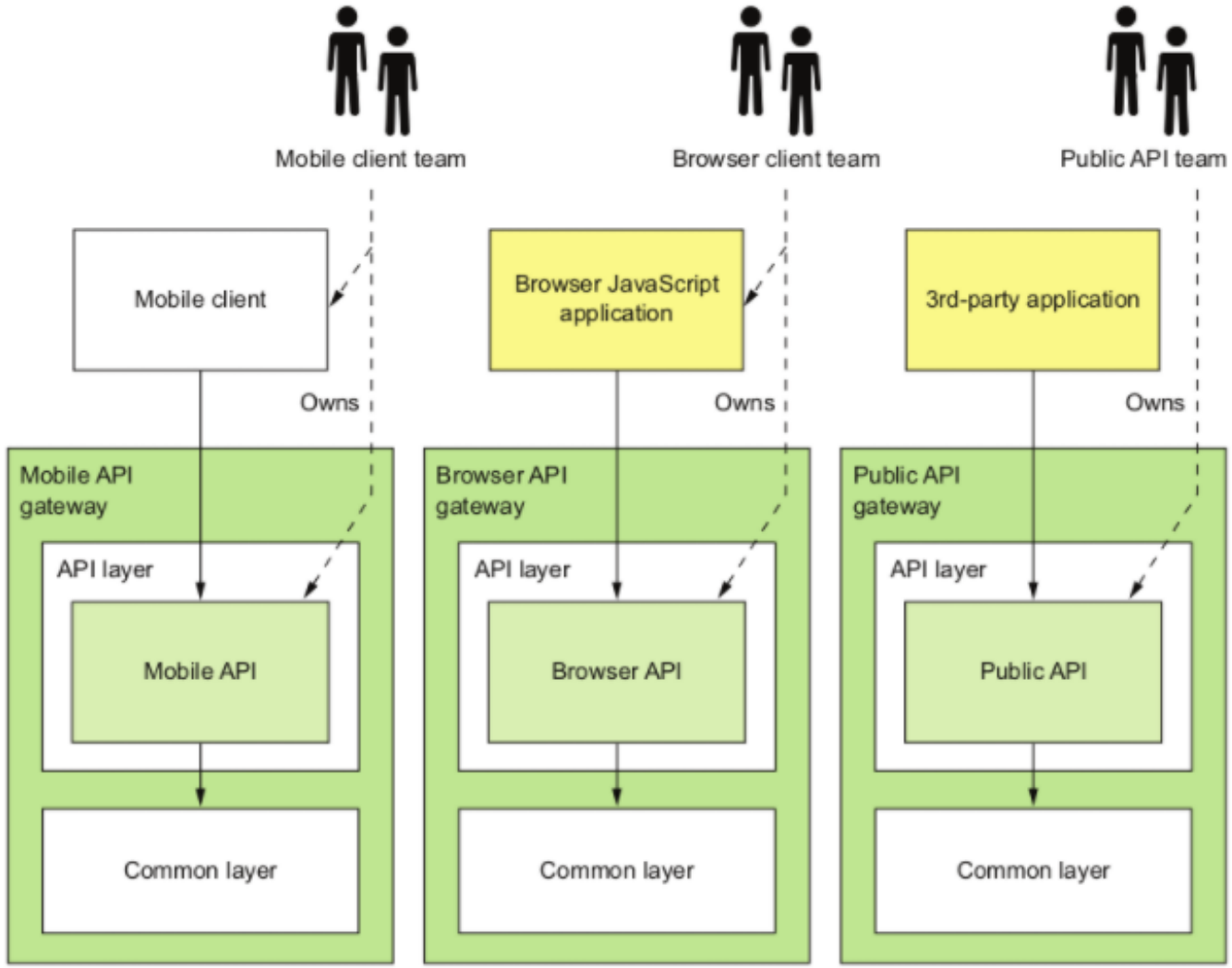
API Gateway – Edge Functions

- Pode ser utilizado um novo serviço para estas funções
- Nesse caso, deve ser a primeira chamada do API Gateway

BFF

- *Backend For Frontend*
- Mecanismo criado pelo *SoundCloud*
- Cada *frontend* deve ter um *backend* específico que acessa os serviços
- Variação do API Gateway

BFF


Mobile client team
Browser client team
Public API team
Mobile client
Browser JavaScript application
3rd-party application
Owns
Owns
Owns
Mobile API gateway
Browser API gateway
Public API gateway
API layer
API layer
API layer
Mobile API
Browser API
Public API
Common layer
Common layer
Common layer

BFF

- Em teoria, o BFF pode ter sua própria stack
- O recomendado é utilizar a mesma, pois duplicação de código pode ser um problema
- É importante que estes elementos sejam o mais leves possíveis

Quem deve ser o responsável pelo API Gateway?

- O ideal é que os próprios *clients* sejam donos do projeto
  - Mantém o princípio de times autônomos
  - Alteração disponível na entrega, pelo próprio time
- Evita a burocracia de uma nova *feature* na fila do backend
  - Replanejamento de Sprint do Backend
  - Priorização pode não ser compatível com a necessidade do time de frontend

API Gateways de mercado

- Requer pouco ou nenhum esforço de desenvolvimento
- Menos flexível
- Normalmente não suportam API Composition

API Gateways de mercado

- AWS API Gateway
- AWS Load Balancer
- Kong
- Traefik
- WSO2
- Gravitee

Implementando o API Gateway

- Aplicação WEB que funciona como Proxy
- Alguns pontos devem ser considerados
  - Roteamento com um mínimo esforço
  - Implementar corretamente o protocolo HTTP (incluindo seus headers)
- Existem frameworks para facilitar este trabalho

Netflix Zuul

- https://github.com/Netflix/zuul
- Utilizam uma série de filters, que se comportam como interceptors
  - Roteamento
  - Autenticação e autorização
  - Monitoramento e métricas
- Permite o API Composition, através de controllers MVC
- Configuração baseada em Paths
  - Não é permitido, por exemplo, um GET e um POST ser roteados para serviços distintos, caso tenham o mesmo path

Próximos Passos

- CQRS

OBRIGADO!

Centro
Rua Formosa, 367 - 29º andar Centro, São Paulo - SP, 01049-000

Alphaville
Avenida Ipanema, 165 - Conj. 113/114 Alphaville, São Paulo - SP,06472-002

+55 (11) 3358-7700
contact@7comm.com.br

7COMm
Serviços e Soluções em TI