INSTITUTO FEDERAL
Paraíba

Campus Cajazeiras

PROGRAMAÇÃO P/ WEB 2

3

3. COMUNICAÇÃO ENTRE MICROSSERVIÇOS

PROF. DIEGO PESSOA

DIEGO.PESSOA@IFPB.EDU.BR

@DIEGOEP

CST em Análise e Desenvolvimento de Sistemas

RESUMO

- Aplicando padrões de comunicação (RPC, Circuit Breaker, Client-side Discovery, self registration, server-side discovery, etc)
- A importância da comunicação remota na arquitetura de microsserviços
- Definindo e evoluindo APIs
- Opções e trade-offs
- Confiabilidade do envio de mensagens como transações de banco de dados

VISÃO GERAL DO PROCESSO DE COMUNICAÇÃO NUMA ARQUITETURA DE MICROSSERVIÇOS

- Serviços podem usar requisições síncronas baseadas em mecanismos como HTTP-based REST ou gRPC.
- Alternativamente, podem ser usados mecanismos assíncronos como AMQP or STOMP.
- Também há vários formatos de mensagem, como os baseados em texto JSON, XML ou os baseados em binário como Avro ou Protocol Buffers.

ESTILOS DE INTERAÇÃO

	one-to-one	**one-to-many**
synchronous	Request/ Response	-
asynchronous	Request/Async response One-	Publish/ Subscribe

DEFININDO APIS NUMA ARQUITETURA DE MICROSSERVIÇOS

- API: define as operações de cada módulo que os clientes podem chamar
- Deve expor apenas funcionalidades e abstrair a implementação
- Como melhorar a definição de APIs?
  - API-first approach:

API FIRST APPROACH

- 1. Escrever a definição da interface dos serviços
- 2. Definir comunicação entre serviços
- 3. Revisar com desenvolvedores e clientes
- Só então, implementar os serviços
- Obs.: Uma API raramente será fixa. Tipicamente, estará sempre em evolução

EVOLUINDO APIS

- Em arquiteturas MS, mudanças em APIs podem impactar e impedir o funcionamento de outros serviços (desenvolvidos por outras equipes)
- Geralmente não dá pra forçar a mudança/atualização
- Como aplicações modernas nunca devem parar para manutenção, versões novas e antigas devem estar aptas para rodar ao mesmo tempo.

USANDO VERSIONAMENTO SEMÂNTICO

- Conjunto de regras que especificam como os números de versão devem ser incrementados

MAJOR	Quando a mudança gera incompatibilidade na API
MINOR	Quando são feitas melhorias retrocompatíveis na API
PATCH	Quando são feitas correções de bugs retrocompatíveis

USANDO VERSIONAMENTO SEMÂNTICO NAS APIS

- Adicionar a major version nas URLs das APIs
- Incluir o número da versão como parte da mensagem que é publicada pela API
- Vantagem: evitar duplicidade de APIs e evoluir de maneira controlada

USANDO VERSIONAMENTO SEMÂNTICO NAS APIS

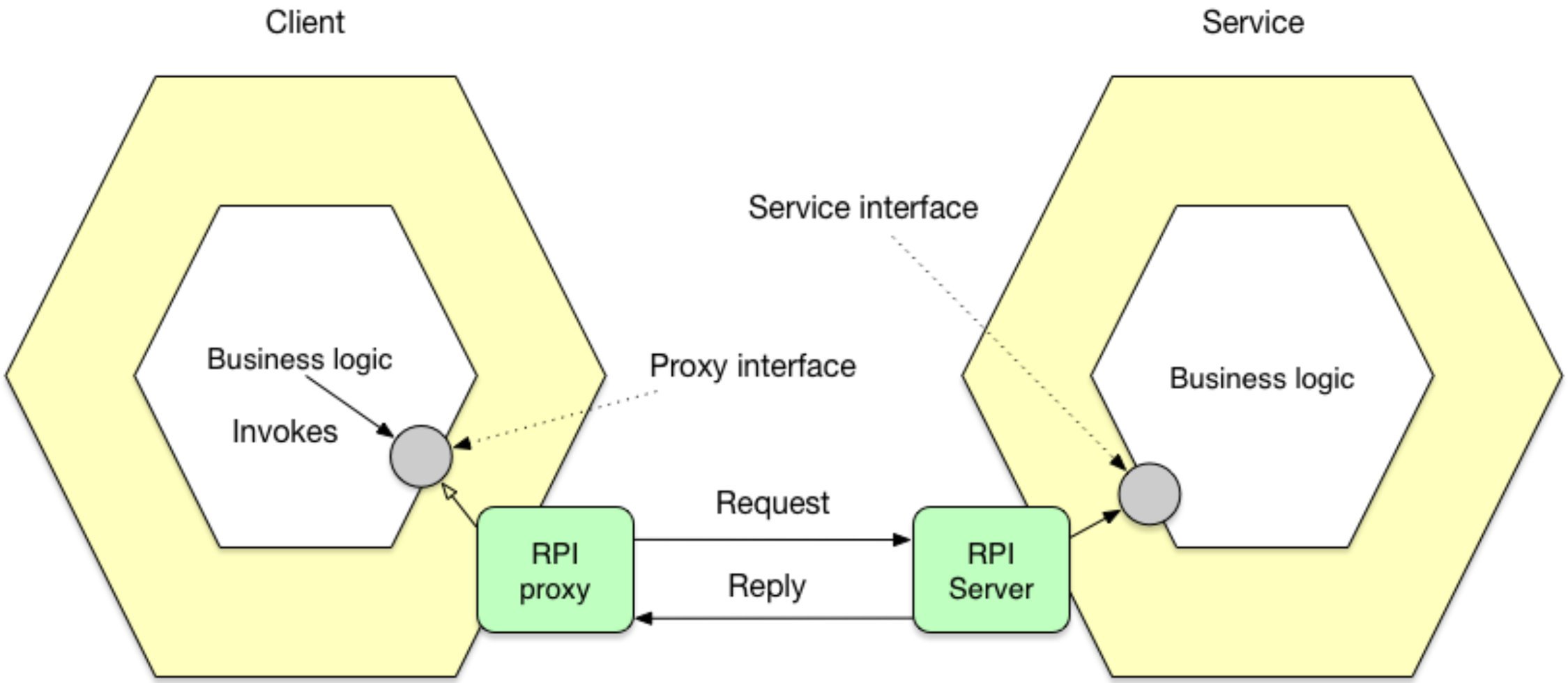
- Opção 1: adicionar as versões aos paths:
  - /v1/..., /v2/...
- Opção 2: Usar mecanismo de negociação de conteúdo HTTP:

```
GET /orders/xyz HTTP/1.1

Accept: application/resource+json; version=1

...
```

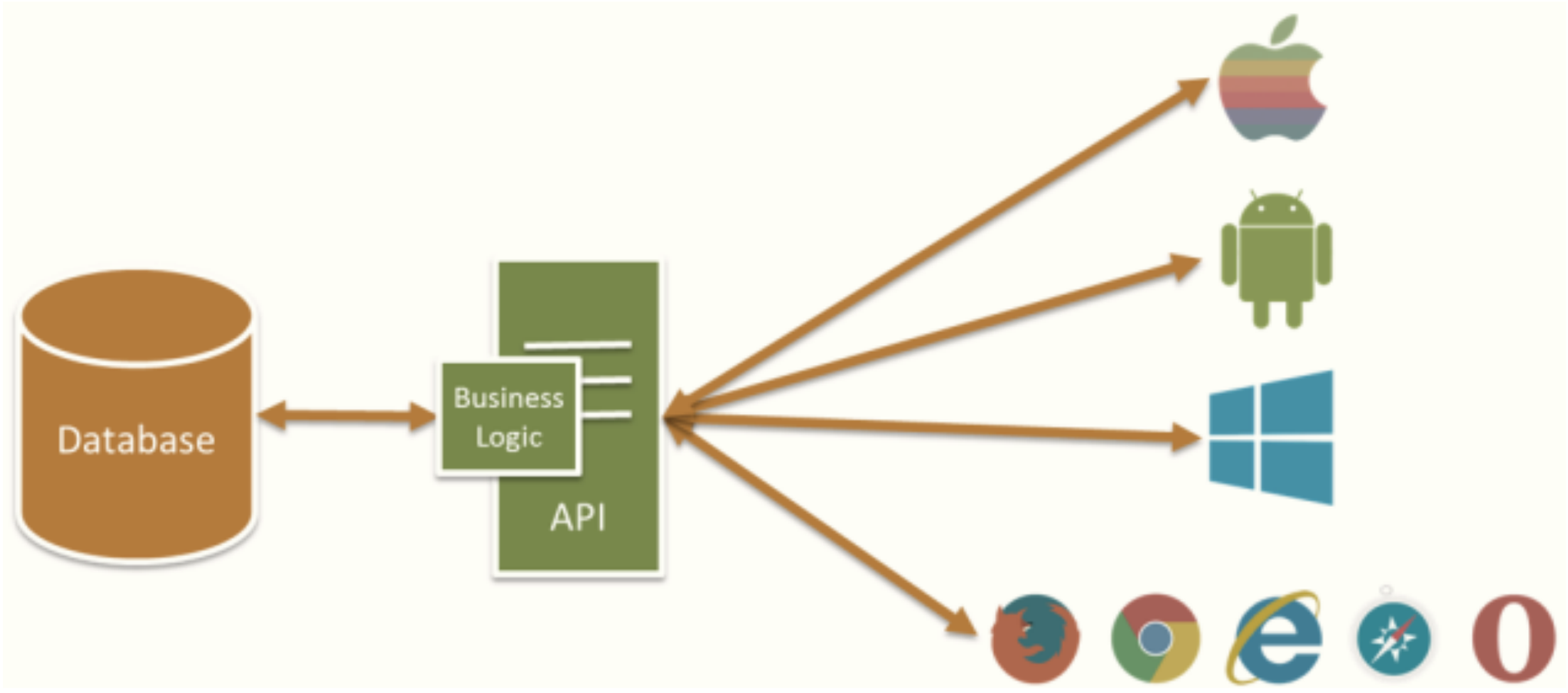
FORMATOS DE MENSAGEM

- Baseados em texto: JSON, XML
- Formatos Binários: Protocol Buffers, Avro.
- Proveem uma IDL (Interactive Data Language) para estruturar mensagens.
- Compilador gera o código que serializa/deserializa mensagens

COMUNICAÇÃO SÍNCRONA USANDO RPC



USANDO REST



MODELO DE MATURIDADE REST

LEVEL 0	Clientes invocam o serviço fazendo requisições HTTP POST a apenas um endpoint para todas as entidades.
LEVEL 1	Suporta a ideia de recursos. Cada entidade possui uma API que é invocada via método HTTP POST.
LEVEL 2	Serviços suportam a utilização de verbos HTTP para realizar ações: GET, POST, PUT, DELETE
LEVEL 3	Baseado no princípio HATEOAS (Hypertext As The Engine Of Application State). Recurso contém links para relacionados.

ESPECIFICANDO APIS

- O desafio de recuperar múltiplos recursos numa requisição única. (Exemplo: recuperar pedido e cliente)

```
/orders/order-id-1345?expand=consumer
```

- O desafio de mapear operações para verbos HTTP

```
POST /orders/{orderId}/cancel
```

```
POST /orders/{orderId}/revise
```

VANTAGENS E DESVANTAGENS DO REST

- **Vantagens:**
- Simples e familiar
- Pode ser testado a partir do browser com um plugin (Postman) ou pela linha de comando com CURL (caso um formato de mensagem baseado em texto seja usado)
- Suporta diretamente a comunicação no estilo requisição/resposta
- HTTP é firewall friendly
- Não requer um broker intermediário, o que simplifica a arquitetura do sistema

VANTAGENS E DESVANTAGENS DO REST

- **Desvantagens:**
- Só suporta o estilo de comunicação requisição/resposta
- Disponibilidade reduzida - visto que o cliente e serviço se comunicam diretamente sem um intermediário para armazenar as mensagens.
- Clientes precisam conhecer as URLs das instâncias dos serviços, o que não é trivial em aplicações modernas. Com isso, clientes precisam usar o mecanismo conhecido como "Service Discovery" para localizar instâncias.
- Recuperar múltiplos recursos numa única requisição é desafiador
- Algumas vezes'e complicado mapear múltiplas operações de atualização para verbos HTTP

USANDO GRPC

```
service OrderService {
  rpc createOrder(CreateOrderRequest) returns (CreateOrderReply) {}
  rpc cancelOrder(CancelOrderRequest) returns (CancelOrderReply) {}
  rpc reviseOrder(ReviseOrderRequest) returns (ReviseOrderReply) {}
  ...
}

message CreateOrderRequest {
  int64 restaurantId = 1;
  int64 consumerId = 2;
  repeated LineItem lineItems = 3;
  ...
}

message LineItem {
  string menuItemId = 1;
  int32 quantity = 2;
}


message CreateOrderReply {
  int64 orderId = 1;
}
...
```

VANTAGENS E DESVANTAGENS DO GRPC

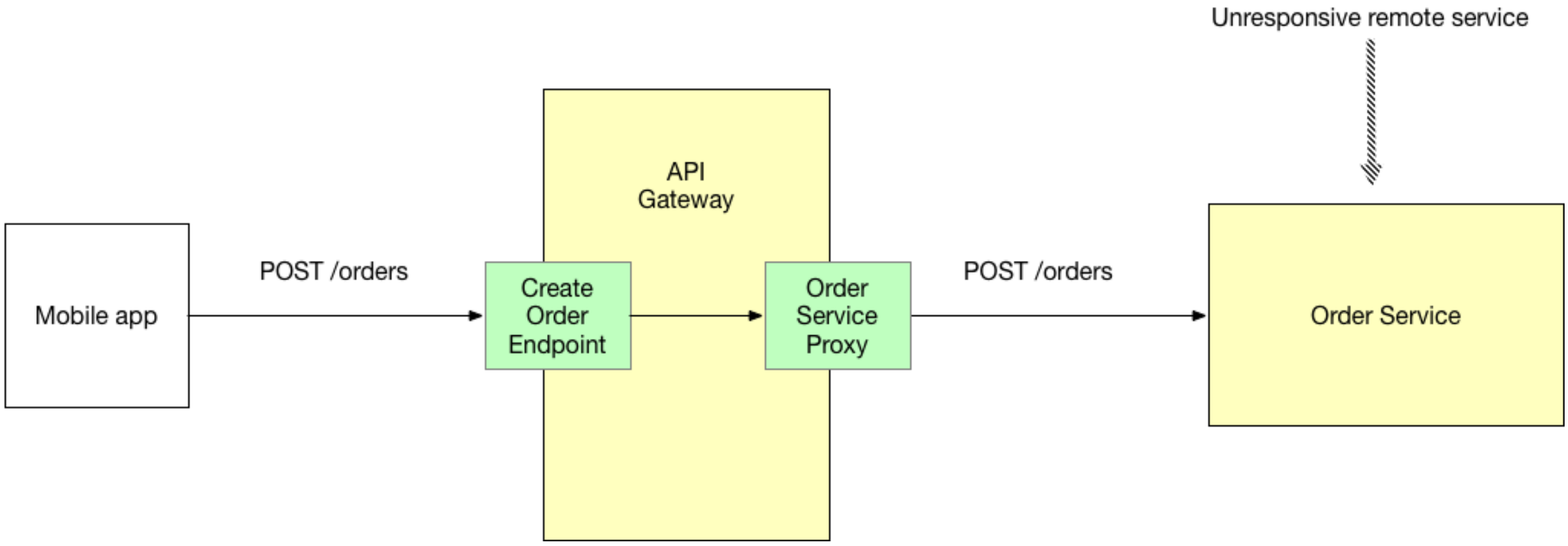
- **Vantagens:**
- Facilidade de especificação da API, que possui várias operações de atualização
- Mecanismo eficiente de compactação, o que é uma vantagem quando são trocadas mensagens grandes
- Streaming bidirecional que permite outros estilos de comunicação
- Permite interoperabilidade entre clientes e serviços escritos em uma ampla gama de linguagens

VANTAGENS E DESVANTAGENS DO GRPC

- **Desvantagens:**
- Mais trabalho para clientes Javascript para consumir APIs baseadas em gRPC do que REST.
- Firewalls antigos podem não suportar HTTP/2 (base do gRPC)
-

PADRÃO CIRCUIT BREAKER

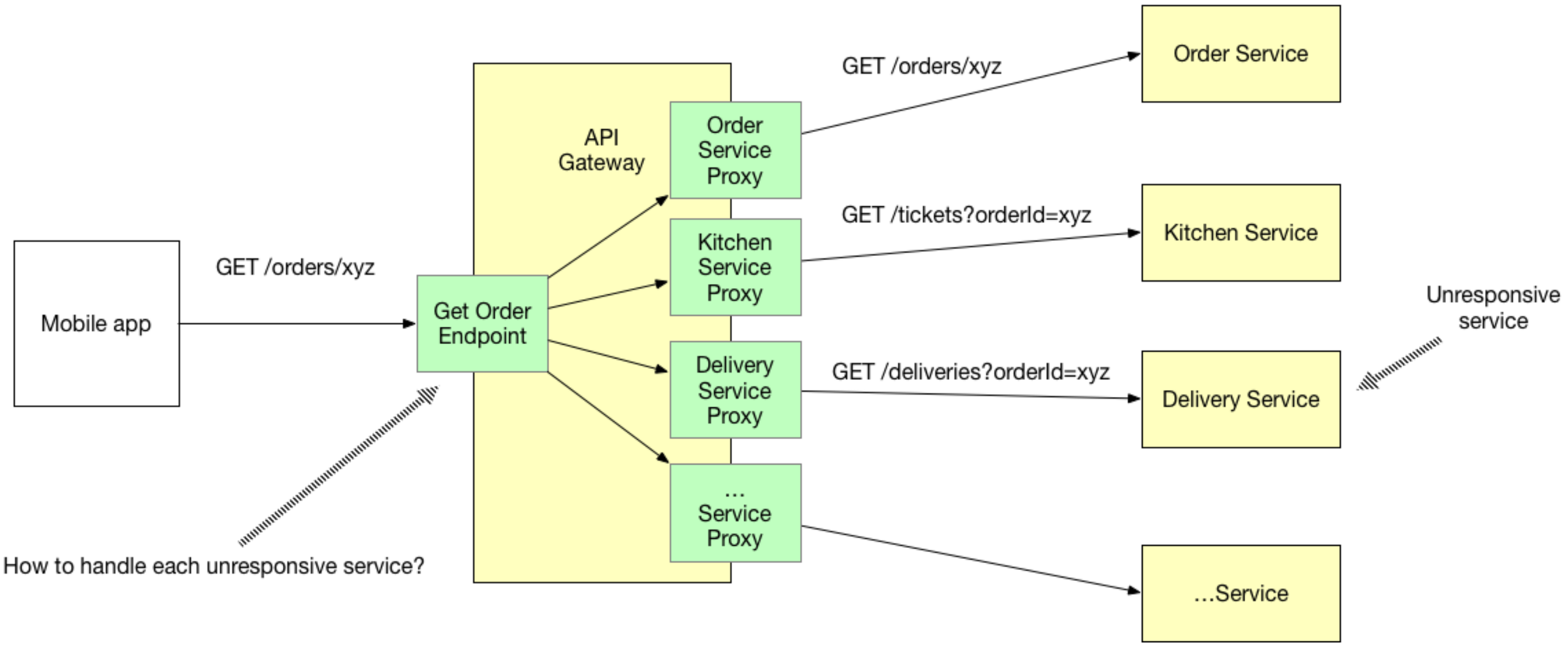
- "An RPI proxy that immediately rejects invocations for a timeout period after the number of consecutive failures exceeds a specified threshold."



PADRÃO CIRCUIT BREAKER

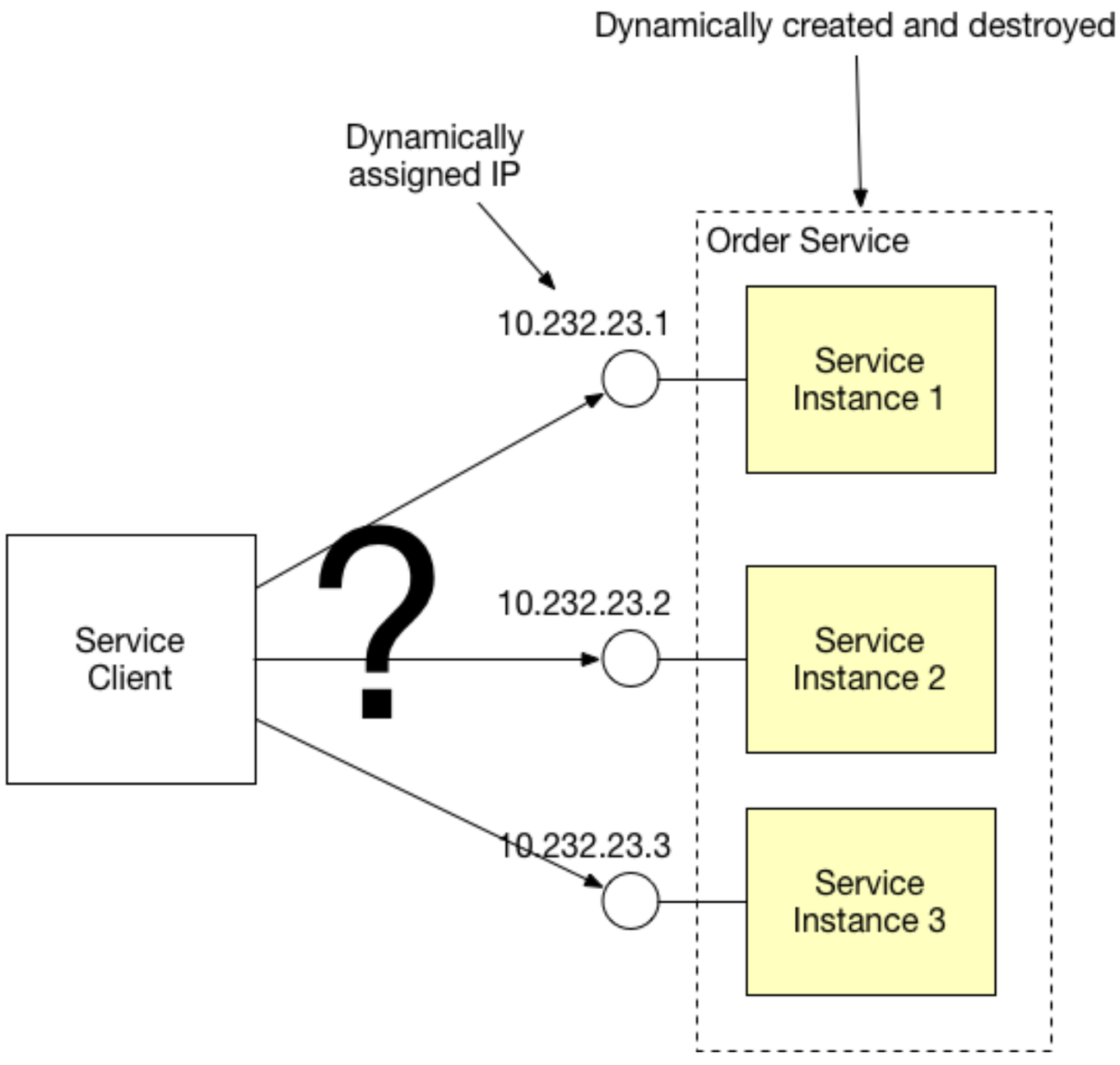
- https://github.com/Netflix/Hystrix é uma biblioteca open source que implementa esse e outros padrões de comunicação.

RECUPERANDO DE UM SERVIÇO INDISPONÍVEL

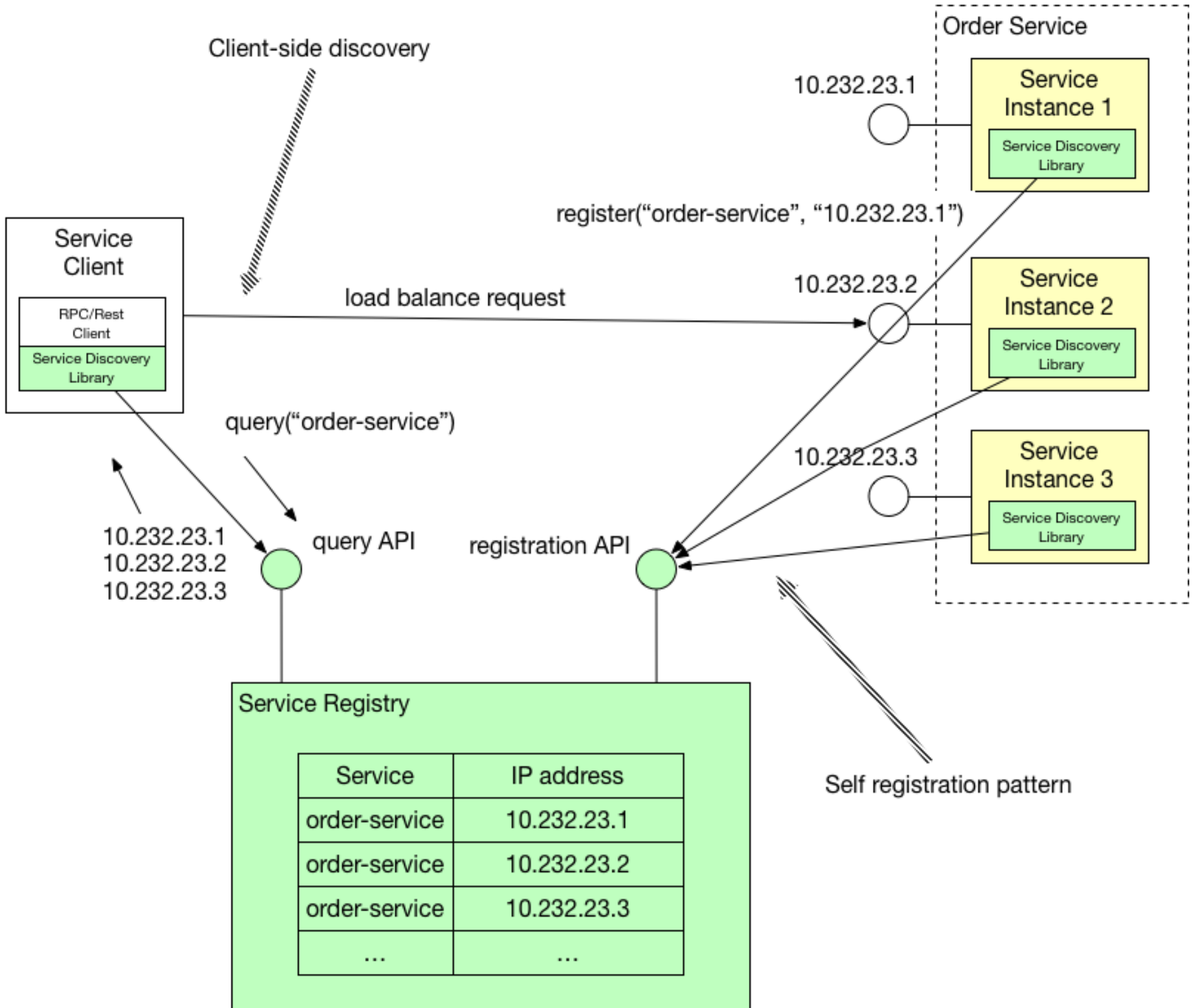


O gateway deve rodar uma versão em cache ou omitir o serviço indisponível da resposta.

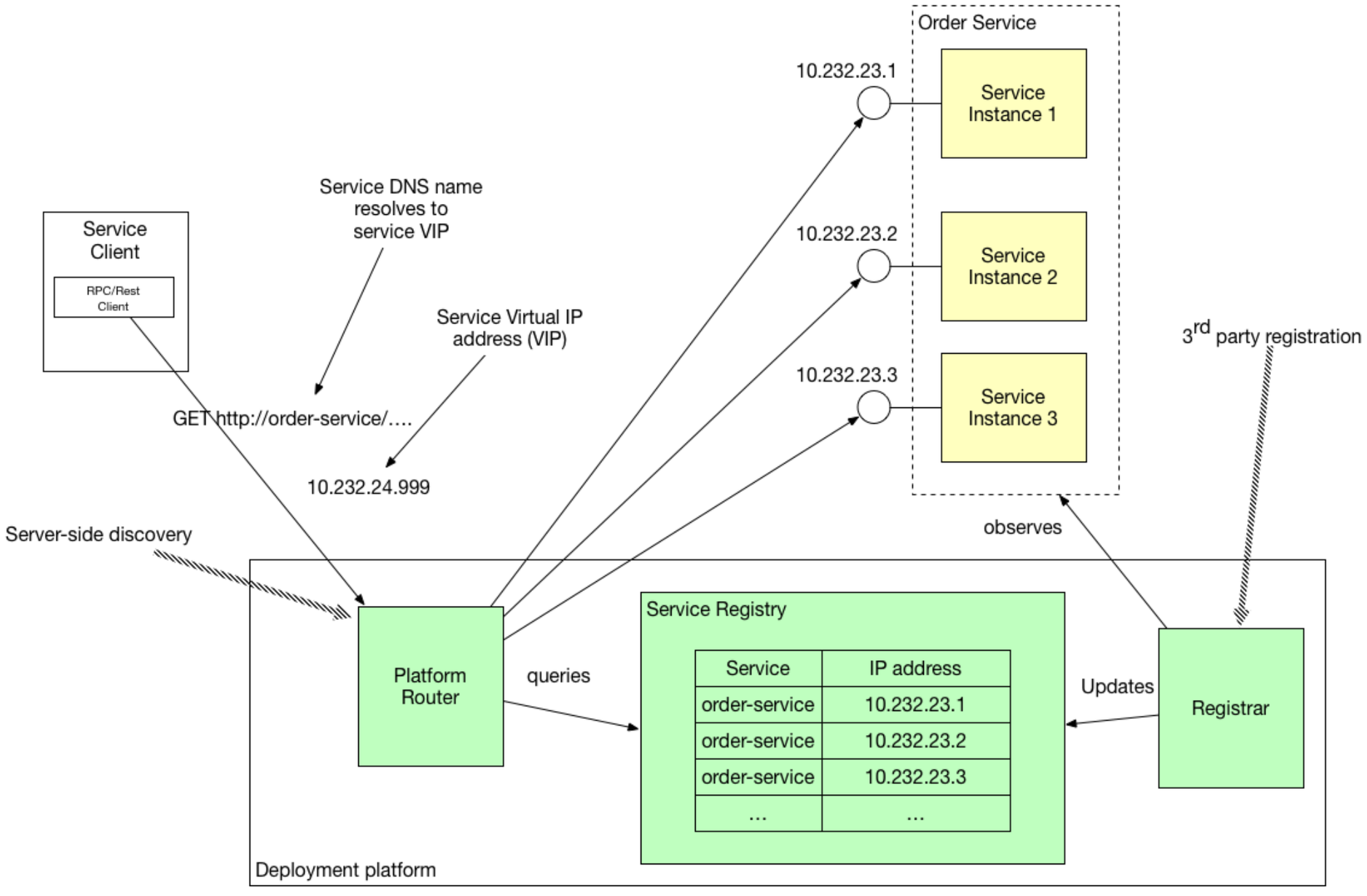
USANDO SERVICE DISCOVERY



APLICANDO SERVIÇOS NO NÍVEL DE APLICAÇÃO PARA PADRÕES DE SERVICE DISCOVERY



APLICANDO SERVIÇOS PARA PLATAFORMA DE SERVICE DISCOVERY



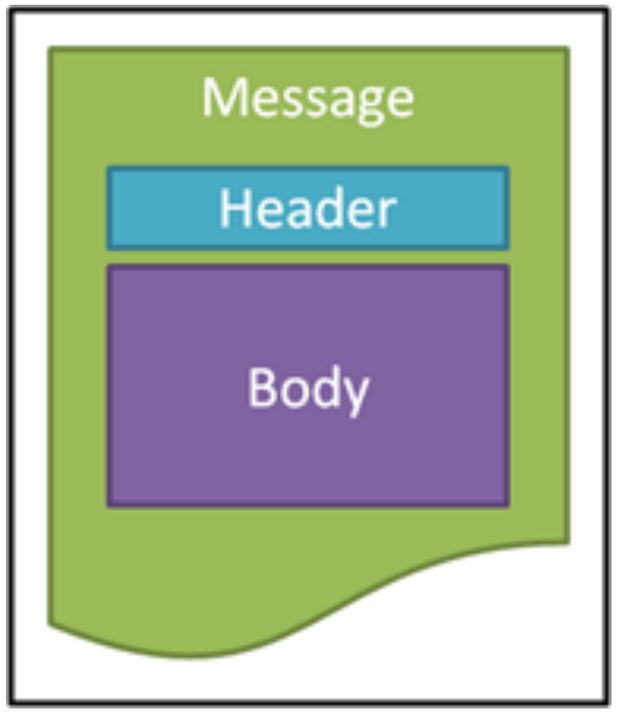
COMUNICAÇÃO ASSÍNCRONA

ENVIO DE MENSAGENS ASSÍNCRONAS

- Pode haver a presença de um *Message Broker* intermediando a requisição ou não (envio direto)
- Fluxo:
  - 1. Cliente envia mensagem para servidor
  - 2. Se a mensagem possui retorno, o servidor envia uma nova mensagem com a resposta para o cliente
- Como a comunicação é assíncrona, o cliente não fica bloqueado aguardando a resposta. Ele assume que a resposta não será imediata.

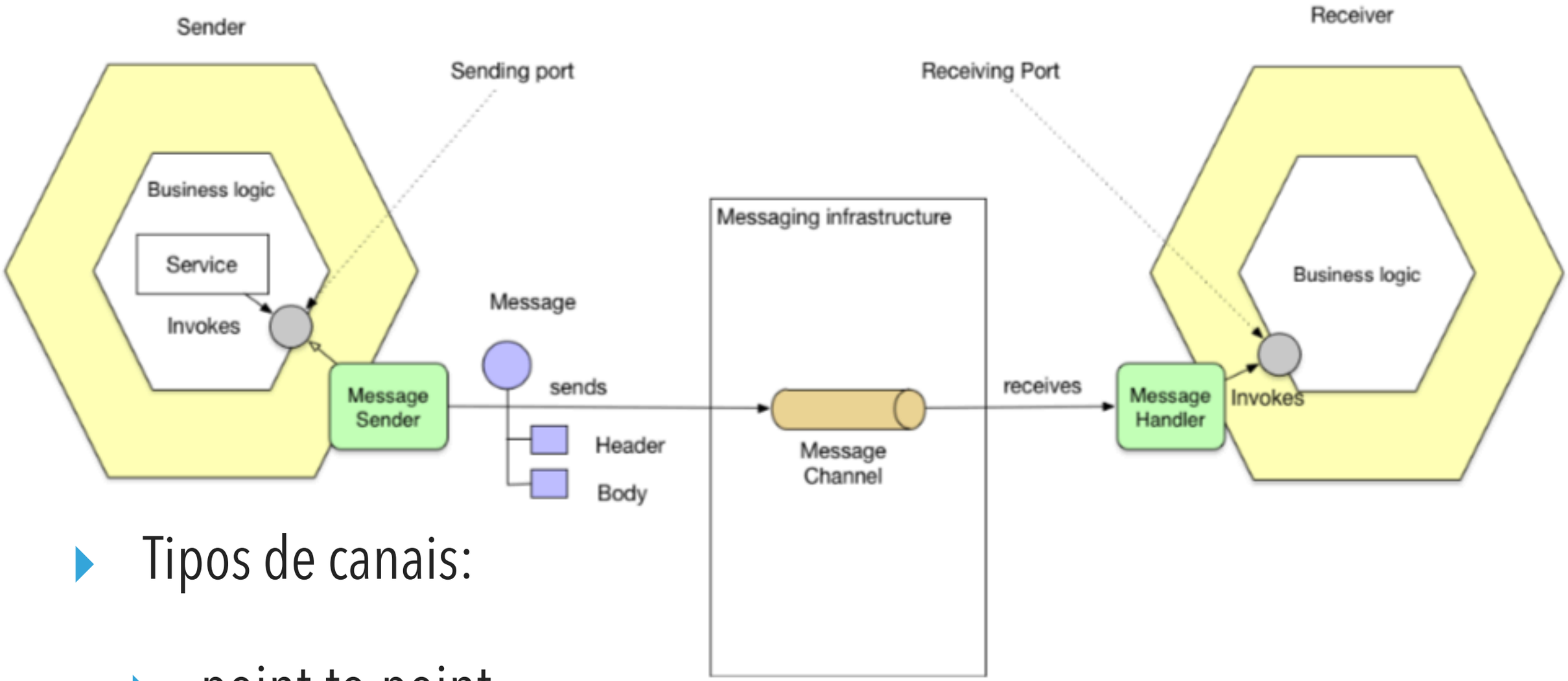
MENSAGENS

- Estrutura



- Tipos de mensagem:
  - Documento
  - Comando
  - Evento

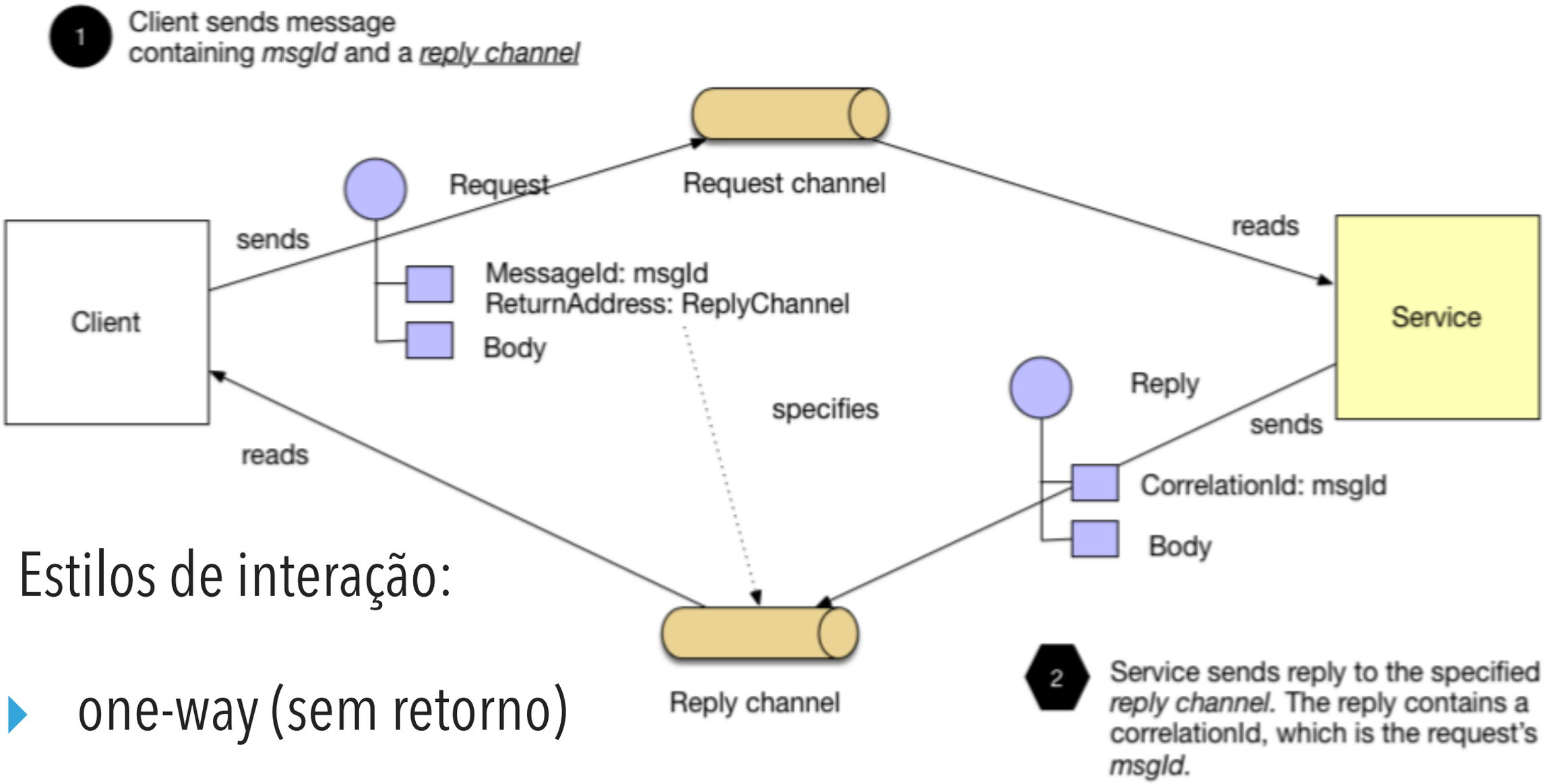
MESSAGE CHANNEL



- Tipos de canais:
  - point-to-point
  - publish-subscribe

Modelo proposto no livro "Enterprise Integration Patterns" por Gregor Hohpe

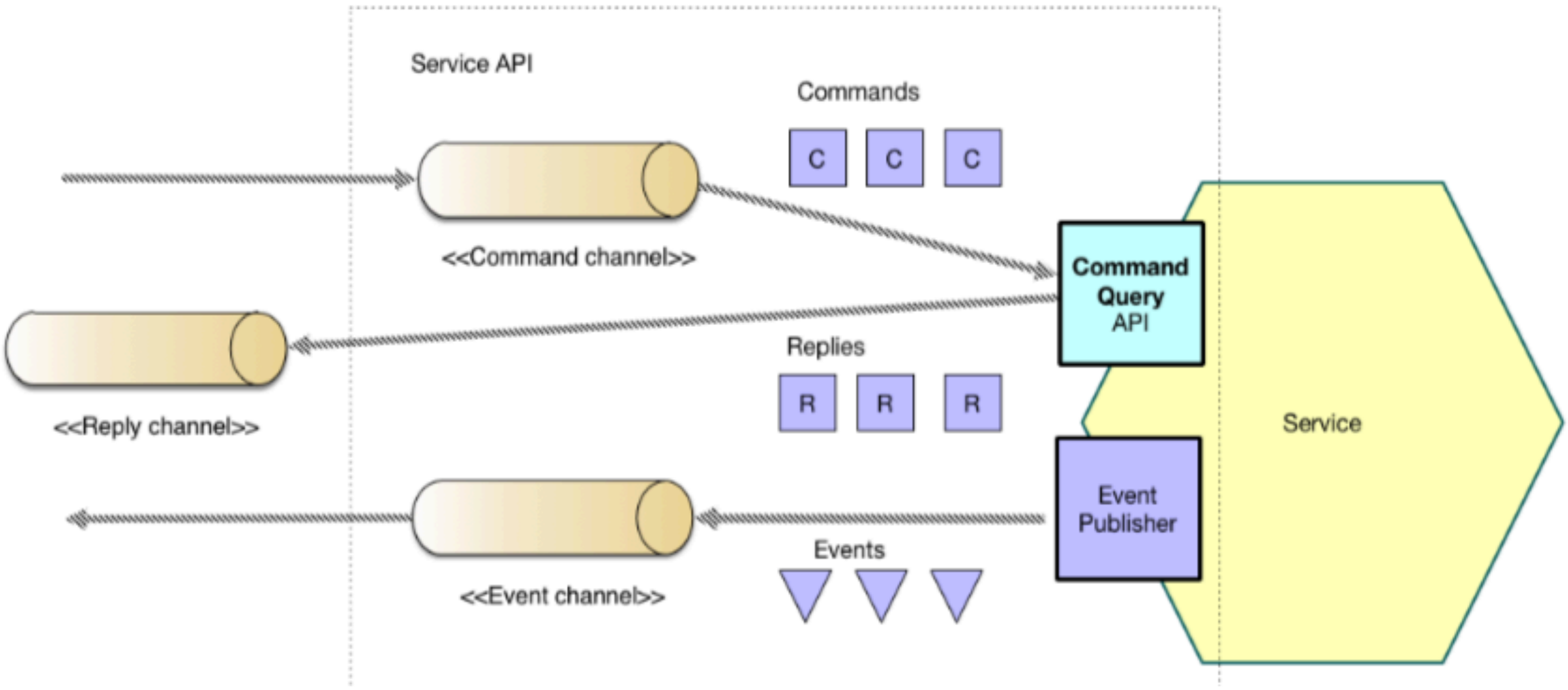
IMPLEMENTANDO OS ESTILOS DE INTERAÇÃO ATRAVÉS DE MENSAGENS



- Estilos de interação:
  - one-way (sem retorno)
  - publish/subscribe (sem resposta ou com resposta assíncrona)

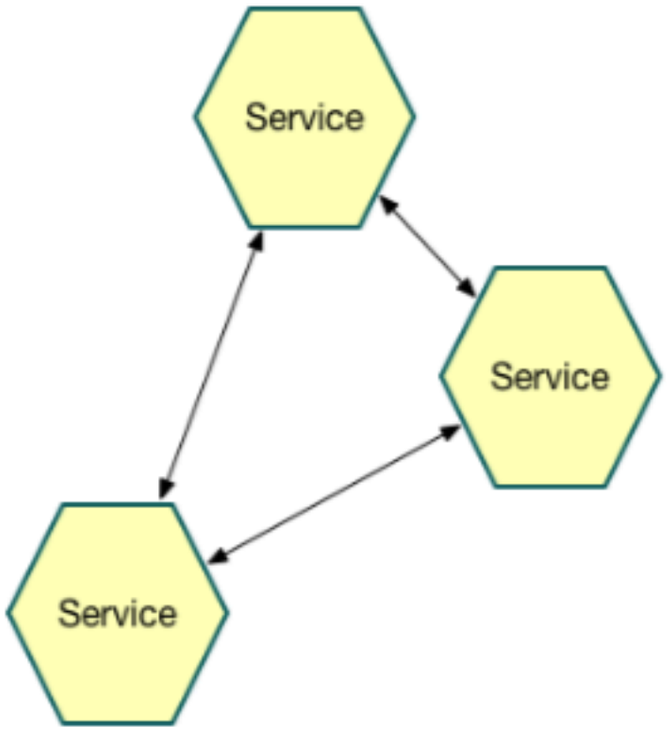
CRIANDO UMA API PARA UM SERVIÇO BASEADO EM MENSAGENS

- Documentar operações e documentar eventos



USANDO UM MESSAGE BROKER

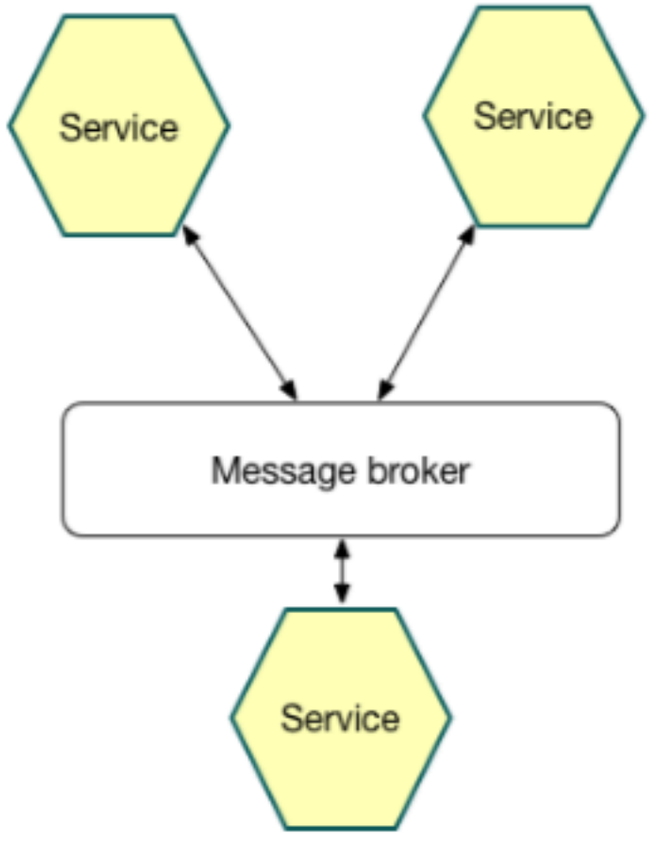
Brokerless architecture



Ex.: ZeroMQ

Vs.

Broker-based architecture



Ex.:
ActiveMQ, RabbitMQ (local)
AWS Kinesis, AWS SQS (nuvem)

BROKERLESS (SEM BROKER)

- Vantagens:
  - Menos tráfego na rede e melhor latência
  - Elimina o ponto único de falha (broker)
  - Menor complexidade (não há broker para manter)
- Desvantagens:
  - Serviços precisam conhecer um ao outro e usar mecanismos de descoberta (ex.: Service Discovery)
  - Disponibilidade reduzida (ambos os pontos precisam estar on-line)
  - Dificuldade para implementar mecanismos como a garantia de entrega

BROKER-BASED

- Vantagens:
  - Menor acoplamento (um serviço não precisa conhecer o destino)
  - Buffering de mensagens (os serviços não precisam estar disponíveis)
  - Comunicação flexível (suporta todos os estilos de interação)
- Desvantagens:
  - Gargalo de desempenho no broker (tudo passa por ele)
  - Broker precisa ter alta disponibilidade (ponto único de falha)
  - Complexidade operacional adicional

TECNOLOGIAS BROKED-BASED

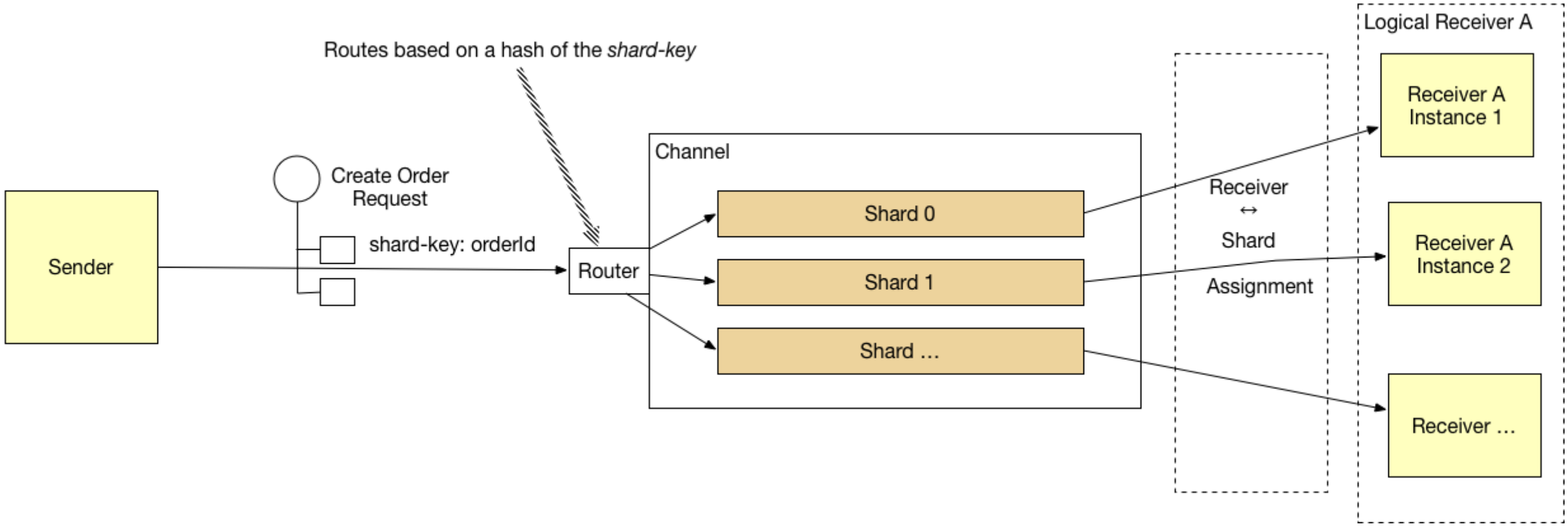
	Point-to-point channel	**Publish-subscribe channel**
JMS	Queue	Topic
Apache Kafka	Topic	Topic
AMQP-based brokers, such as RabbitMQ	Exchange + Queue	Fanout exchange and a queue per consumer
AWS Kinesis	Stream	Stream
AWS SQS	Queue	-

TECNOLOGIAS BROKED-BASED – QUAL ESCOLHER?

- **Fatores a considerar:**
- Linguagens de programação suportadas
- Suporte a protocolos padrões, como AMQP e STOMP ou é proprietário
- Ordenação de mensagens - preserva a ordem?
- Garantia de entrega - que tipo de garantia de entrega é oferecida?
- Persistência - as mensagens são persistidas no disco e são mantidas caso o broker quebre?
- Durabilidade - se um consumidor se reconecta ao broker, ele receberá as mensagens enviadas enquanto estava desconectado?
- Escalabilidade - o quão escalável é um message broker?
  Latência - qual é a latência fim a fim
- Consumidores Concorrentes - Como o broker trata a concorrência entre consumidores?

BROKER-BASED – DESAFIOS:

- Concorrência e ordenação de mensagens



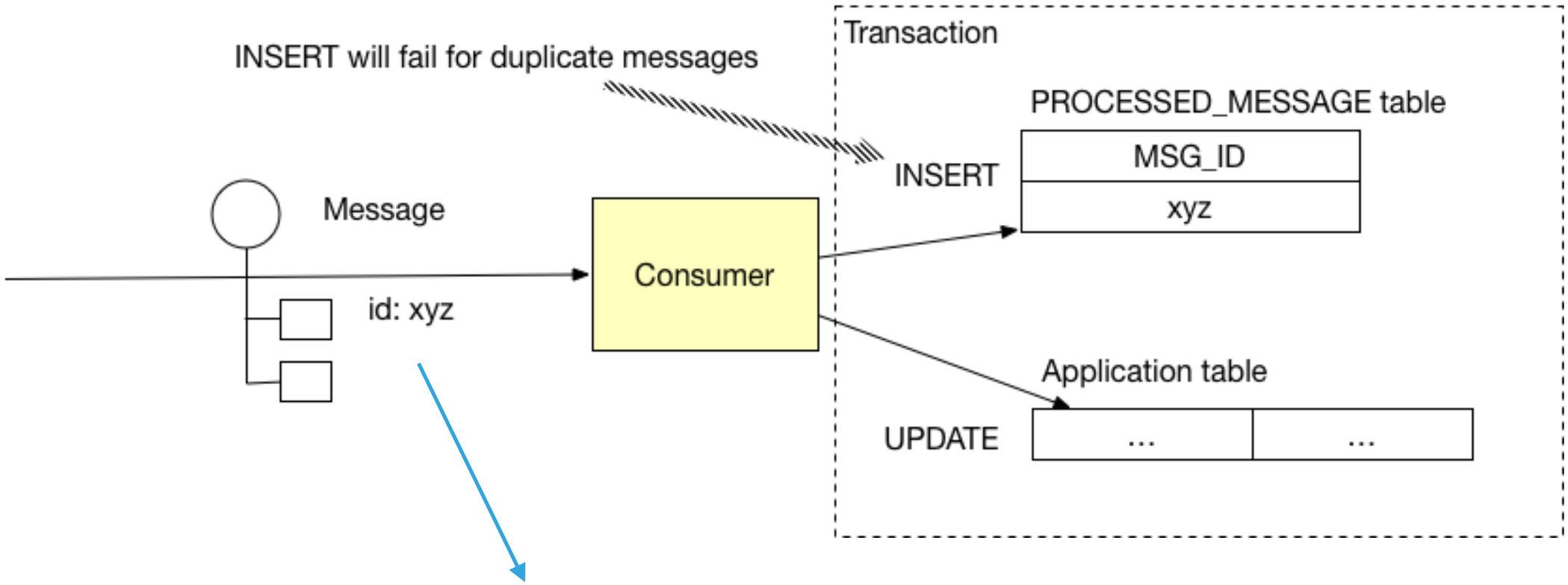
- Shards= canal particionado

BROKER-BASED – DESAFIOS:

- Tratando mensagens duplicadas
  - Broker deve entregar a mensagem apenas uma vez
  - Mas isso é custoso, então os brokers prometem entregar ao menos uma vez
- Estratégias para lidar com mensagens duplicadas:
  - Escrever tratadores de mensagens "idempotentes" (pode enviar quantas mensagens duplicadas for que ele não irá quebrar)
  - Monitorar mensagens e descartar duplicidades

BROKER-BASED – DESAFIOS:

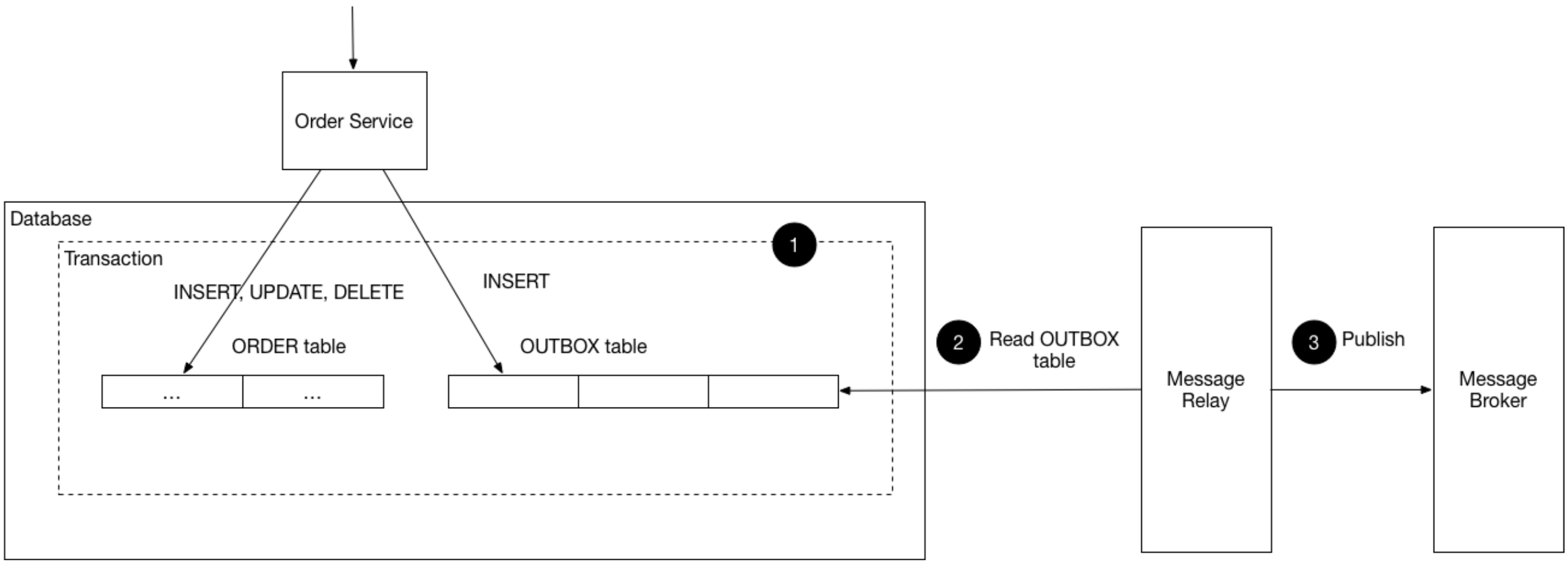
- Monitorar mensagens e descartar duplicatas



Mensagem com o ID da entidade

MENSAGENS TRANSACIONAIS

- Usando banco de dados como fila de mensagens.



MENSAGENS TRANSACIONAIS

PADRÃO TRANSACTIONAL OUTBOX
Publish an event or message as part of a database transaction by saving it in an *outbox* in the database. See http://microservices.io/patterns/data/transactional-outbox.html

- 1. Ler periodicamente a OUTBOX

  SELECT * FROM OUTBOX ORDERED BY ... ASC

- 2. Publicar as mensagens em um Message Broker

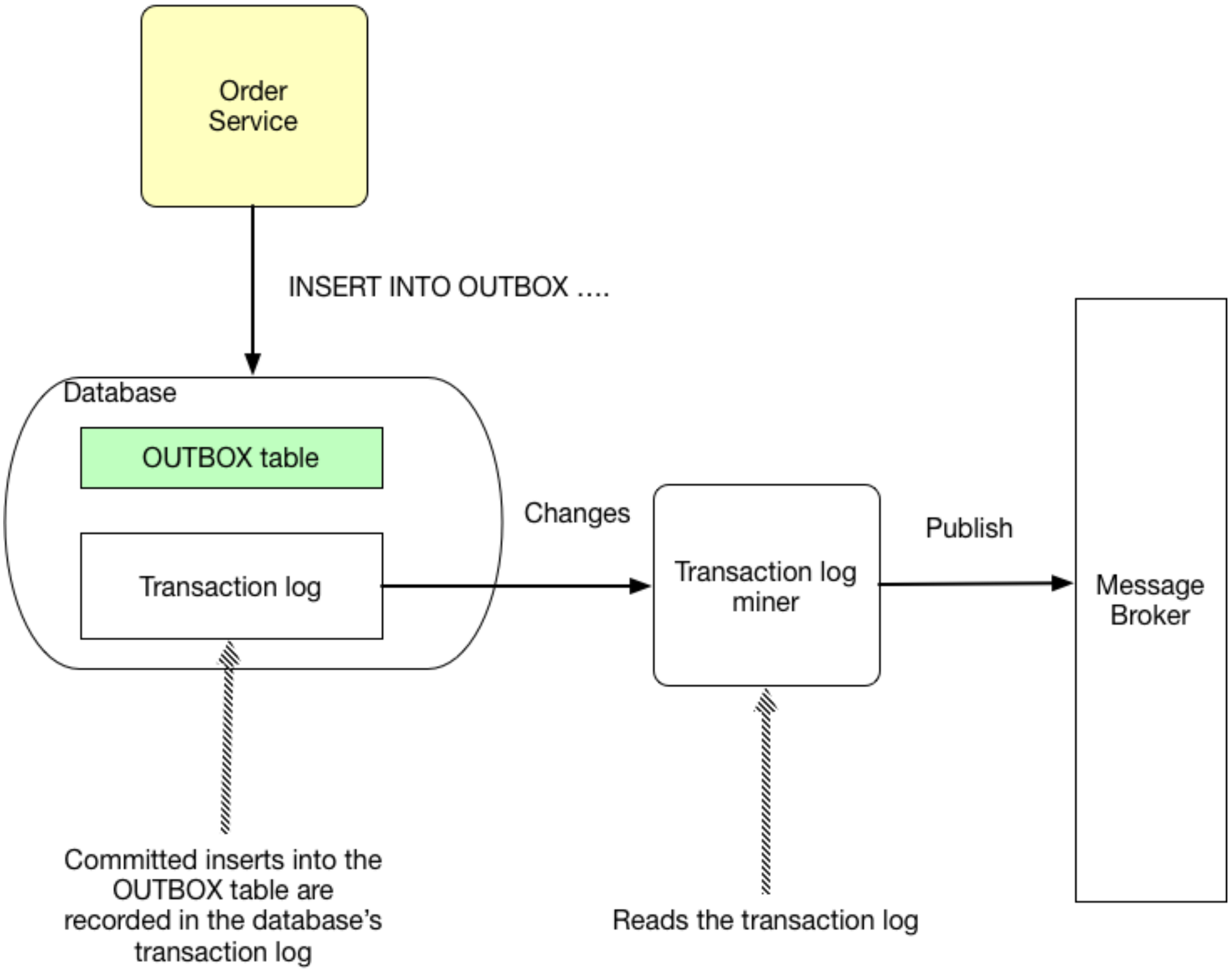
- 3. Limpar a OUTBOX

```
BEGIN
 DELETE FROM OUTBOX WHERE ID in (....)
COMMIT
```

ALTERNATIVA:

PATTERN: POLLING PUBLISHER
Publish messages by polling the outbox in the database. See http://microservices.io/patterns/data/polling-publisher.html

PUBLICANDO EVENTOS APLICANDO O PADRÃO LOG TAILING



EXEMPLOS DE TECNOLOGIAS QUE APLICAM LOG TAILING

- Debezium
- LinkedIn Databus
- DynamoDB streams
- Eventuate Tram