Computação Distribuída

Douglas Pereira Luiz
Odorico Machado Mendizabal

Universidade Federal de Santa Catarina – UFSC
Departamento de Informática e Estatística – INE

Padrões de Projeto para Computação Distribuída

Padrões de Projeto

Soluções gerais que facilitam o desenvolvimento oferecendo:

- Reusabilidade
- Manutenabilidade
- Comunicação

Padrões de Projeto para Computação Distribuída

Soluções específicas para o contexto distribuído

- Comunicação
- Cooperação
- Desempenho

Exemplos

- *Sidecar*
- *Circuit Breaker*
- *Sharding*
- Balanceamento de Carga
- *Scatter/Gather*
- *Publisher/Subscriber*
- Eleição de Líder

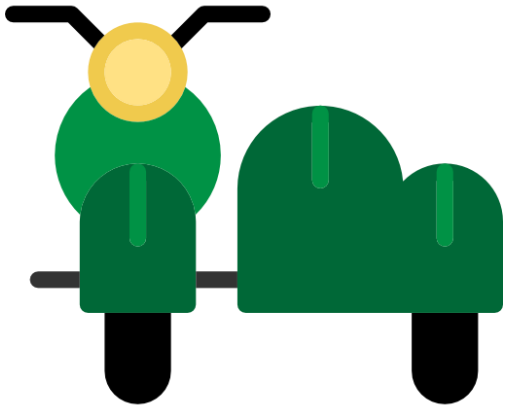
Sidecar

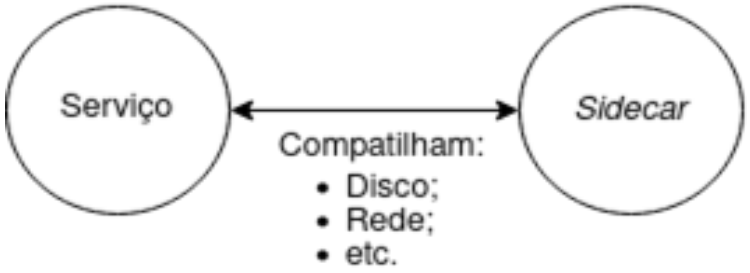
Problema enfrentado

Serviço precisa de uma funcionalidade nova. Exemplo:

- Serviço *web* legado
- Atende requisições com HTTP
- Precisamos melhorar a segurança com HTTPS
- Não queremos investir muito no código legado
- O que fazer?

Adicionamos um *sidecar* ao serviço





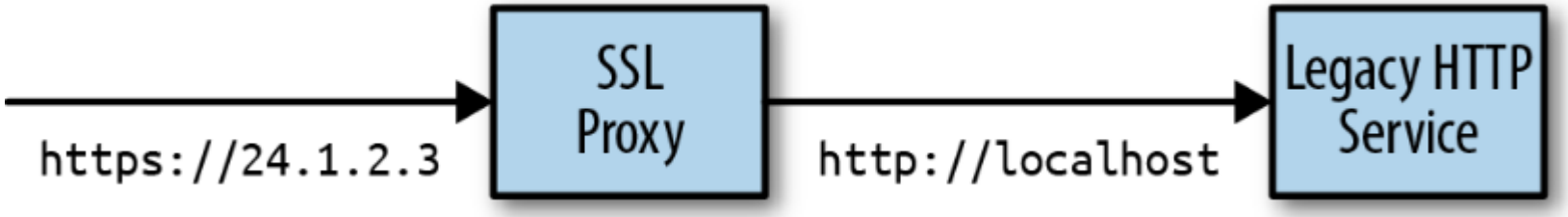
Benefícios do *sidecar*

O *sidecar* serve para:

- Adicionar ou melhorar uma funcionalidade
- Não modifica o serviço internamente
- Pode muitas vezes ser usado sem o conhecimento do serviço

Exemplo: Adicionar HTTPS a um serviço HTTP legado

- O *sidecar* será um serviço SSL na rede local
- Recebe as requisições e repassa ao serviço legado
- O serviço legado atende somente requisições do *sidecar* na rede local



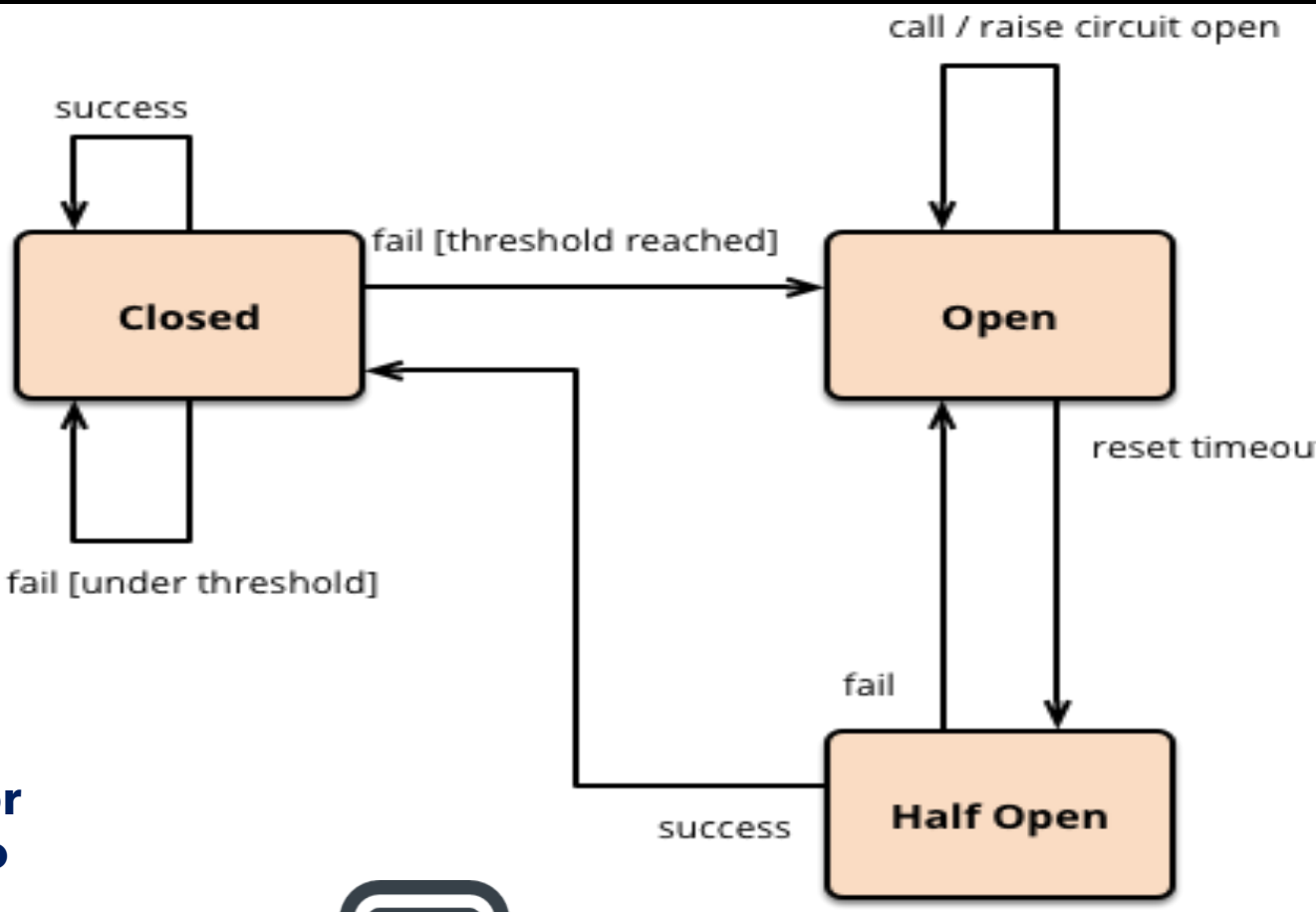
Disjuntor (*Circuit Breaker*)

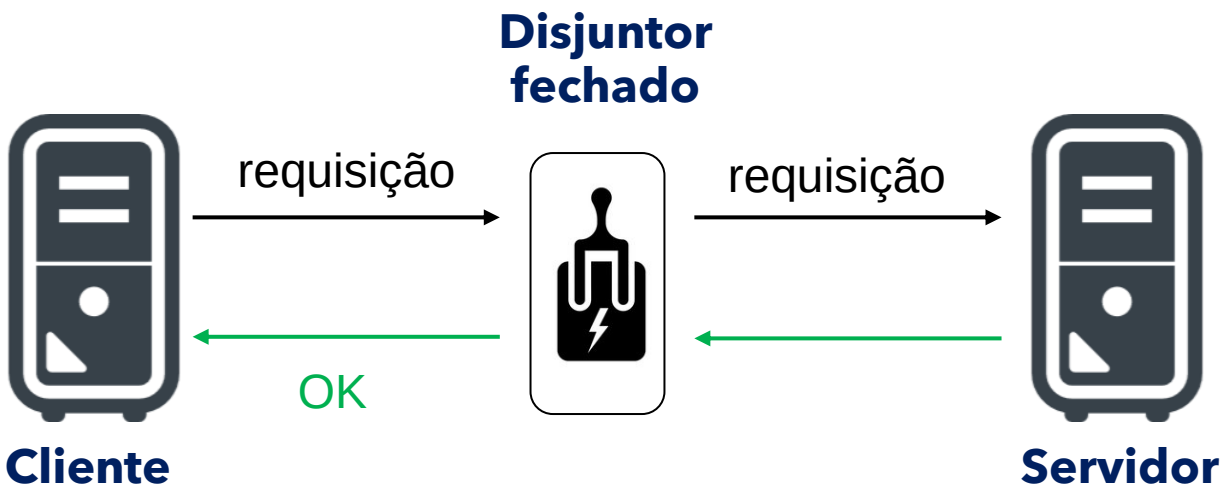
Problema enfrentado

- Requisições à um servidor podem falhar
- Problemas de conexão podem levar os clientes a esperar repetitivamente *timeouts*
- Recursos críticos do cliente e do servidor podem ficar ocupados (ex. *threads*, memória, …)

Disjuntor (*Circuit Breaker*)

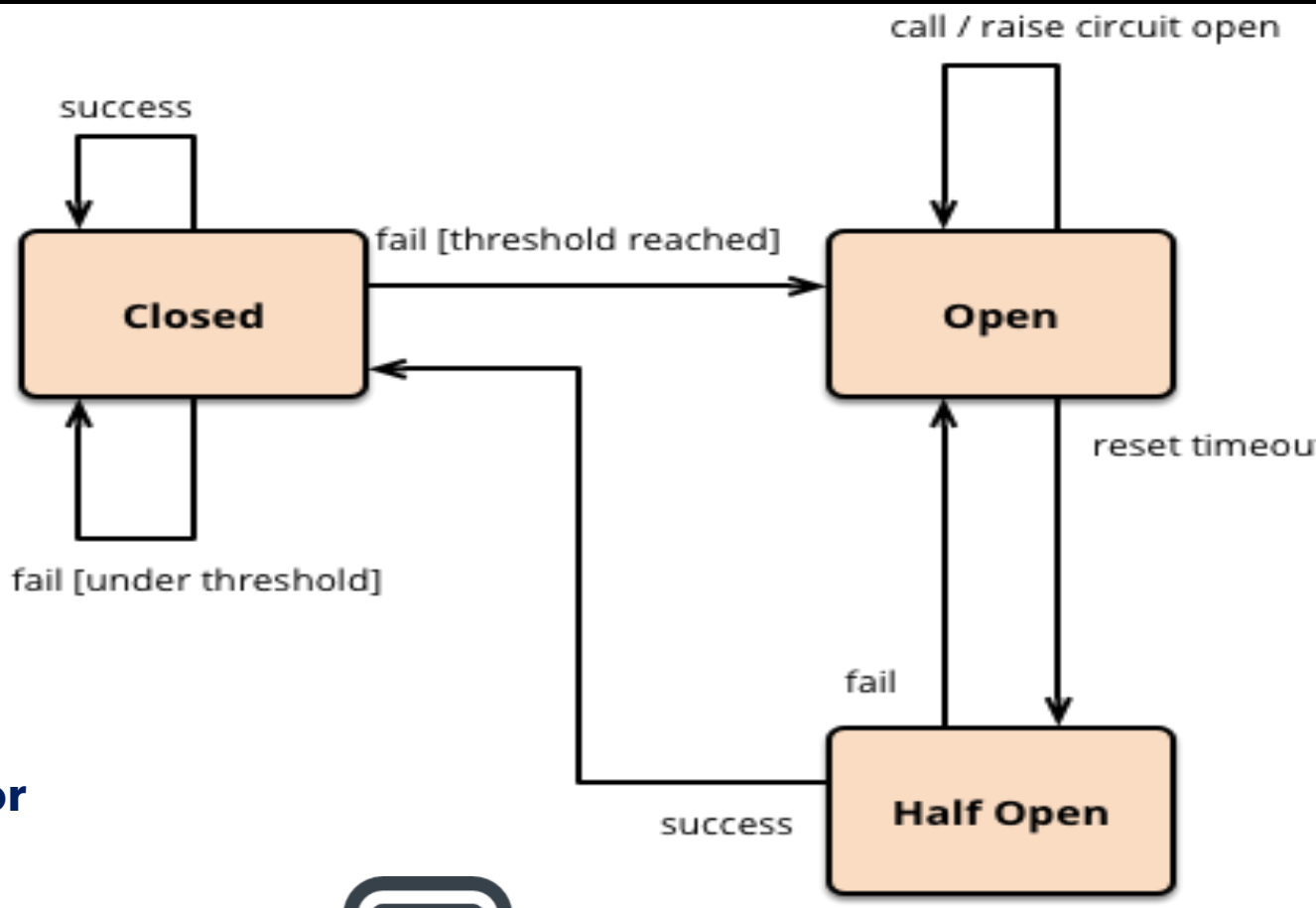
Estados do disjuntor

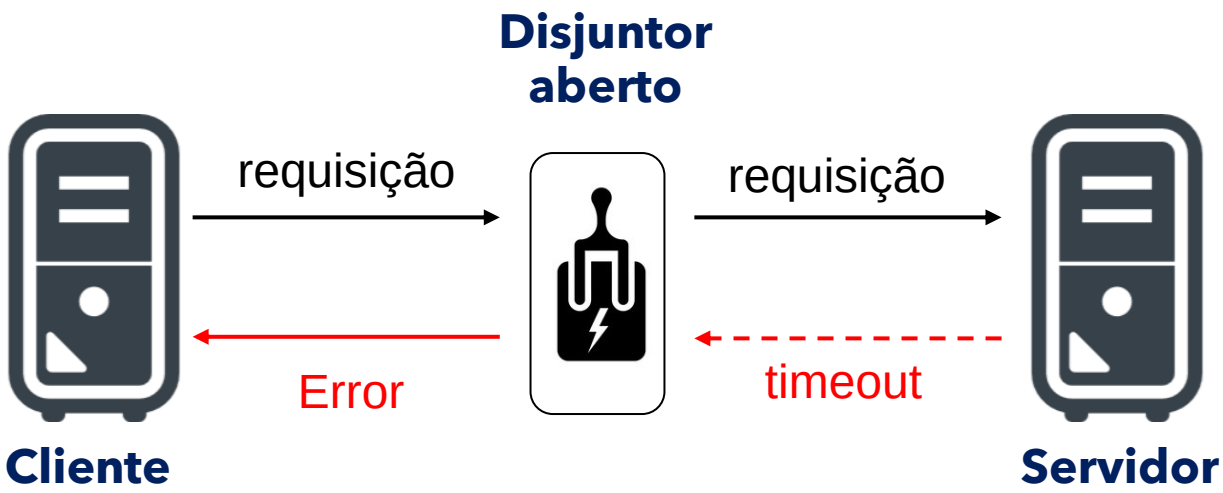




Disjuntor (*Circuit Breaker*)

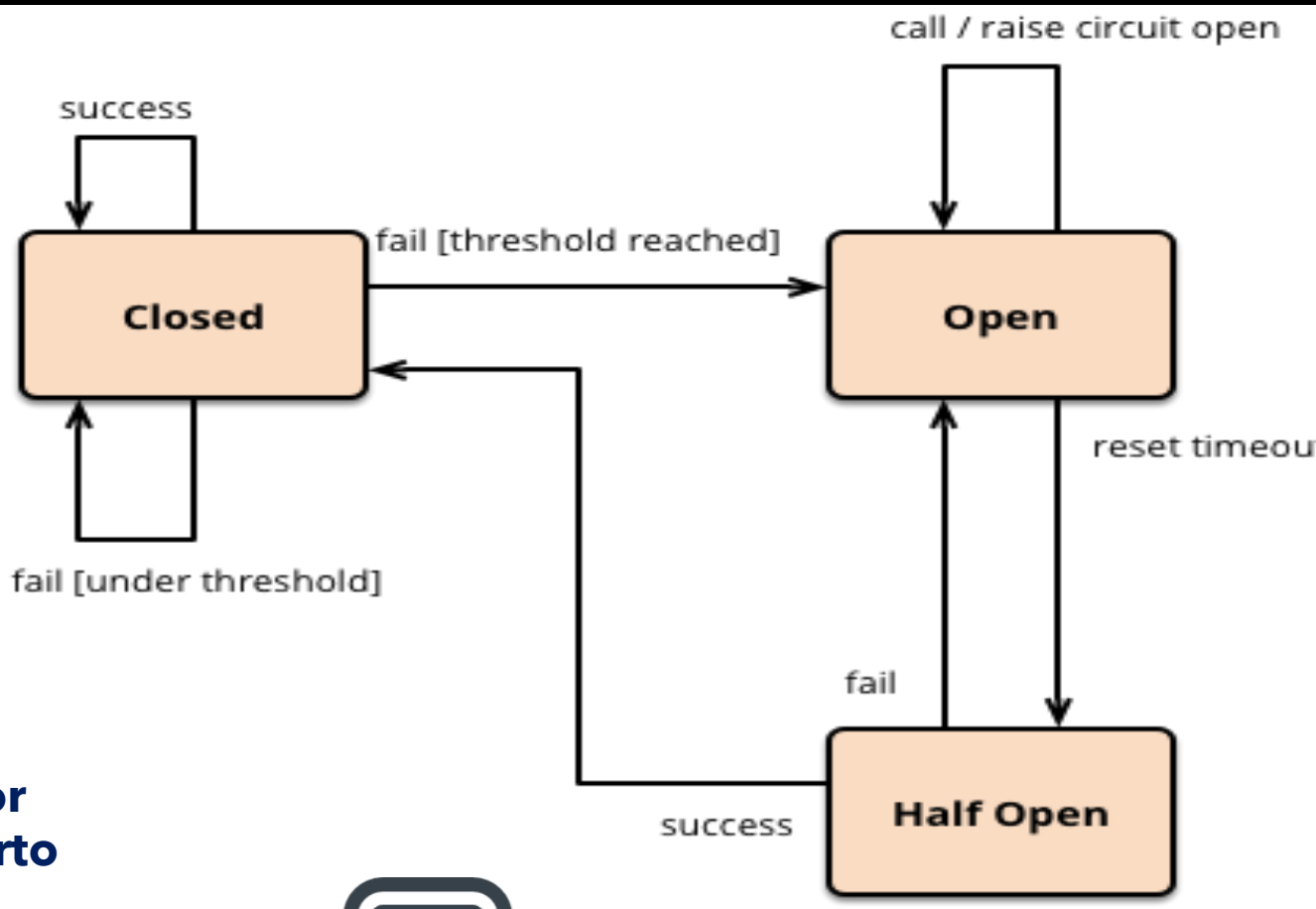
Estados do disjuntor

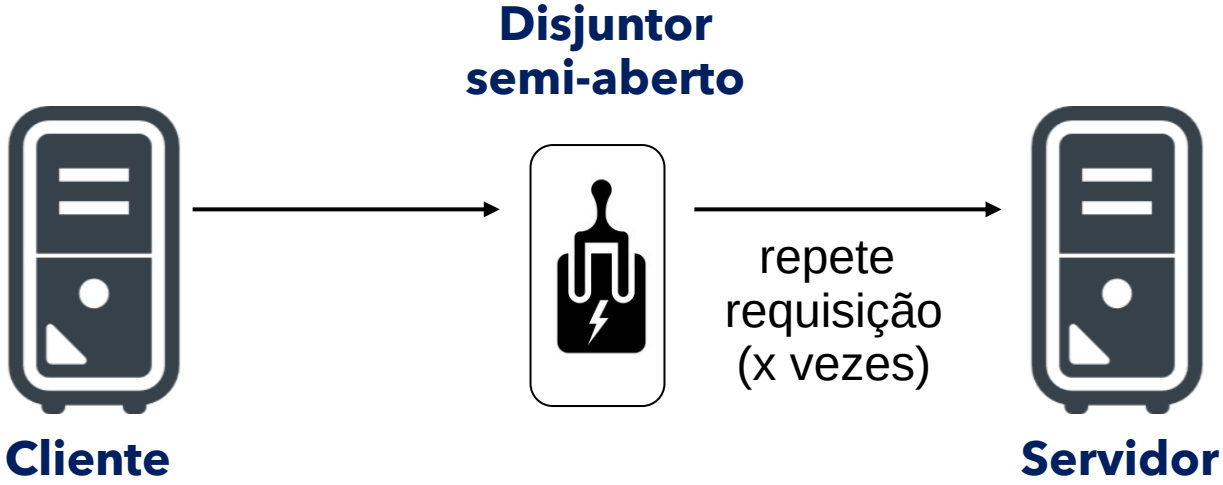




Disjuntor (*Circuit Breaker*)

Estados do disjuntor





Benefícios do disjuntor

- Prevenção de falhas em cascata
- Melhoria da disponibilidade
- Gestão de recursos mais eficiente

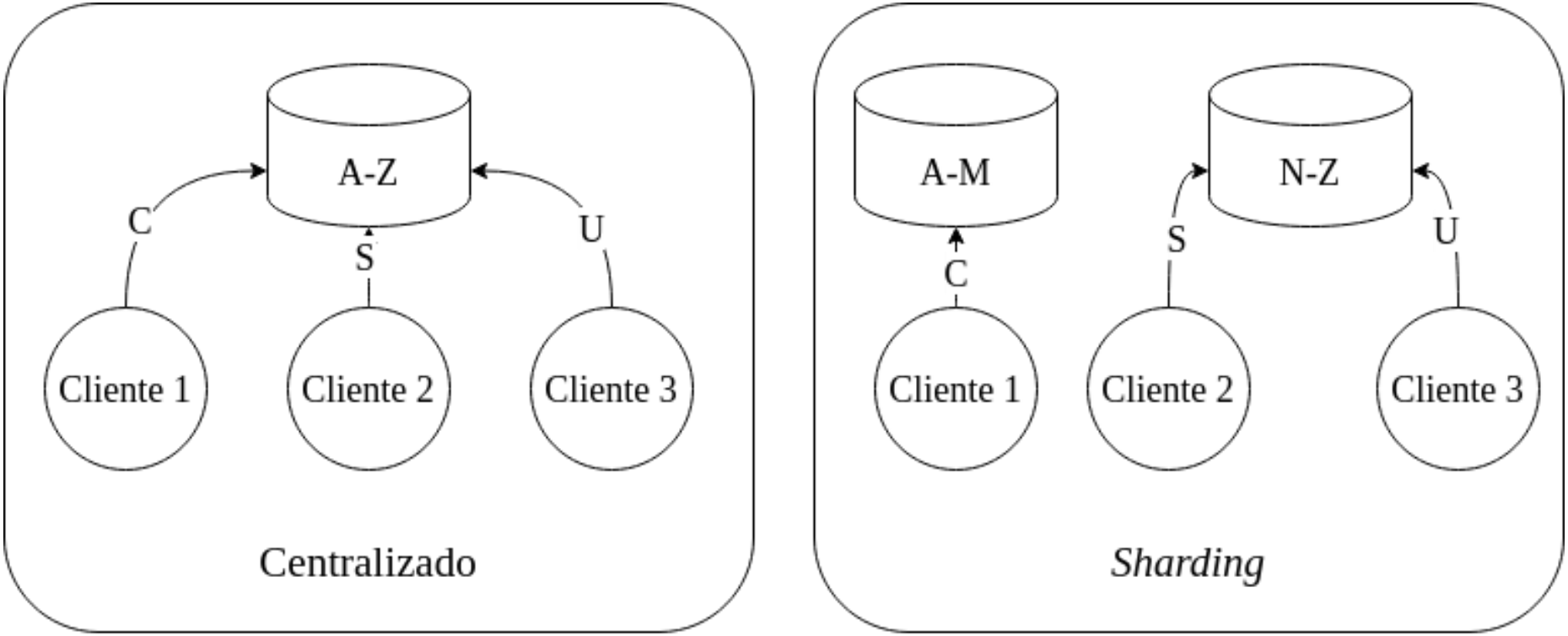
Particionamento (*sharding*)

Situações recorrentes

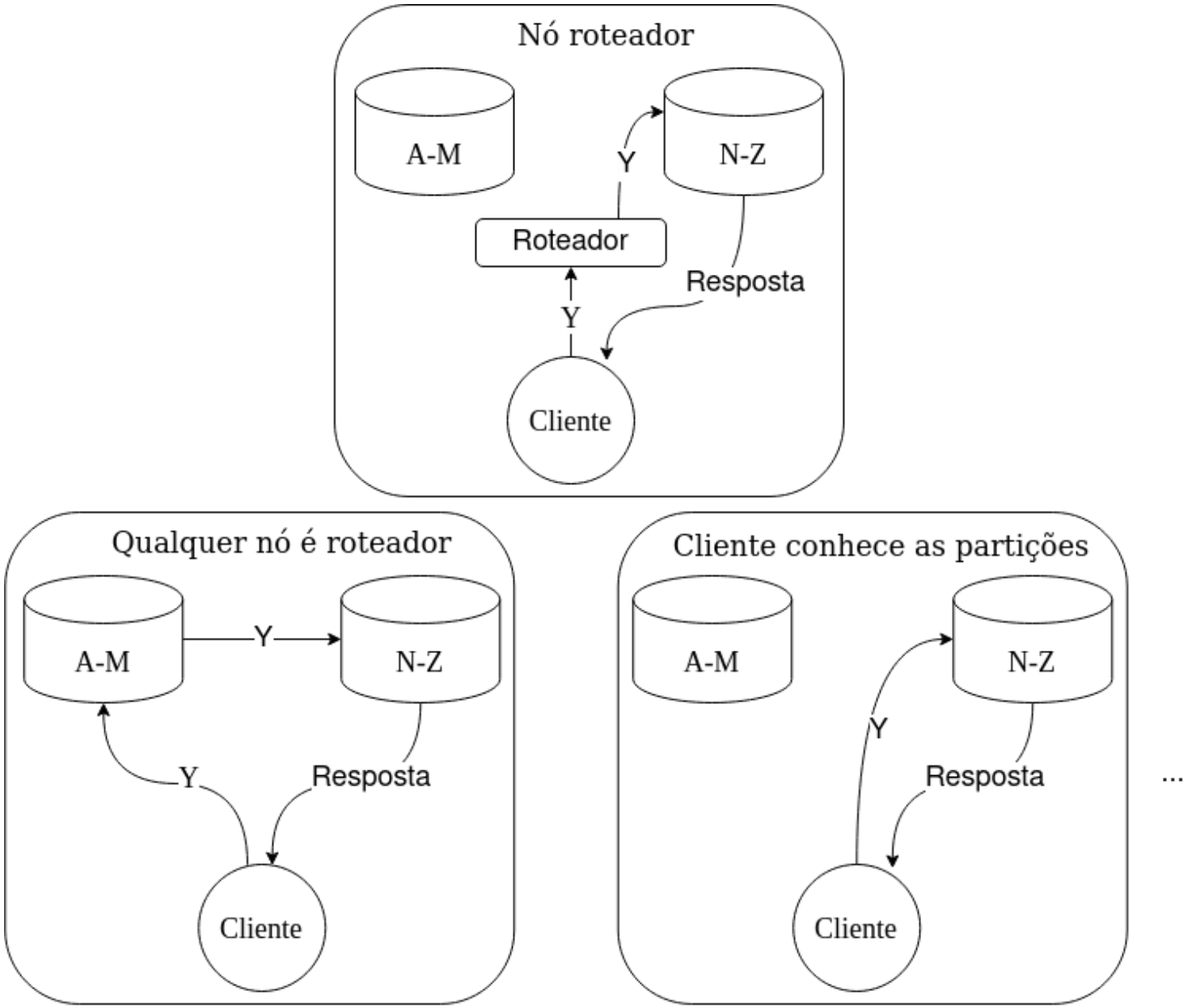
- O custo de armazenamento de todos os dados do sistema em uma máquina é muito caro
- A quantidade de requisições sendo atendidas pelo único servidor é muito grande
- Os dados estão muito distantes de alguns clientes

Sharding

Podemos particionar e distribuir os dados do sistema

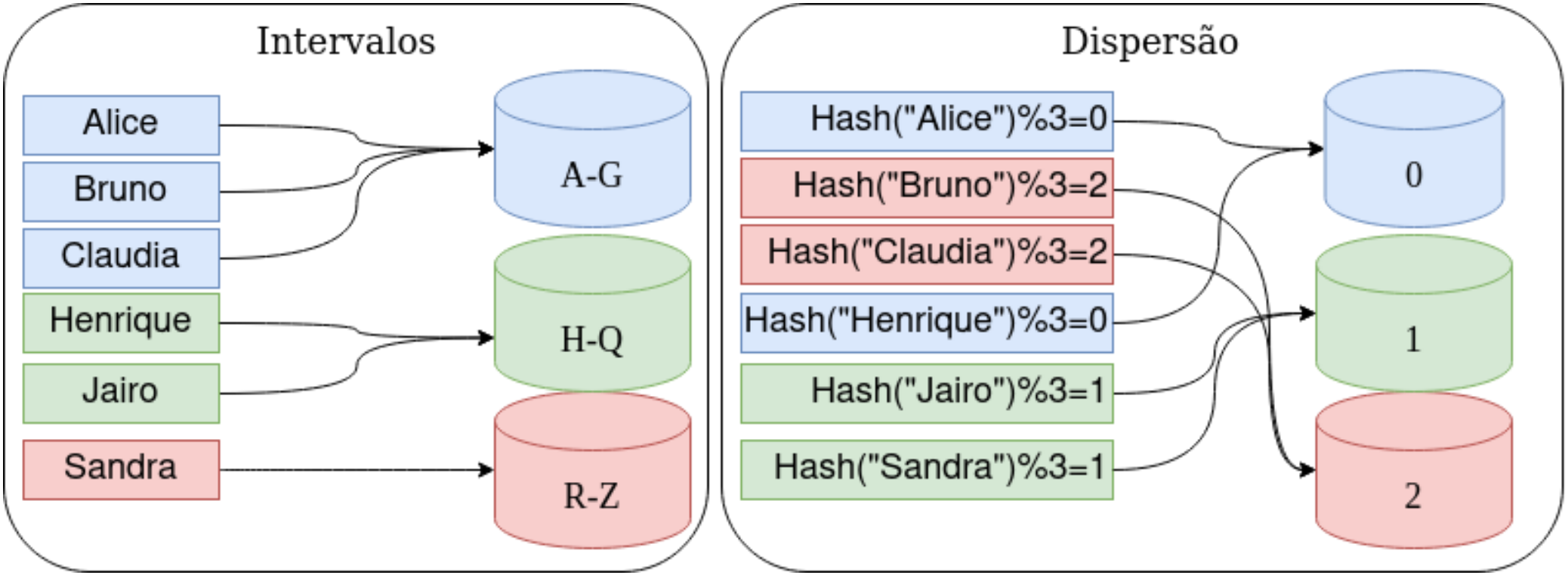


Encontrando o servidor certo



Distribuição de dados

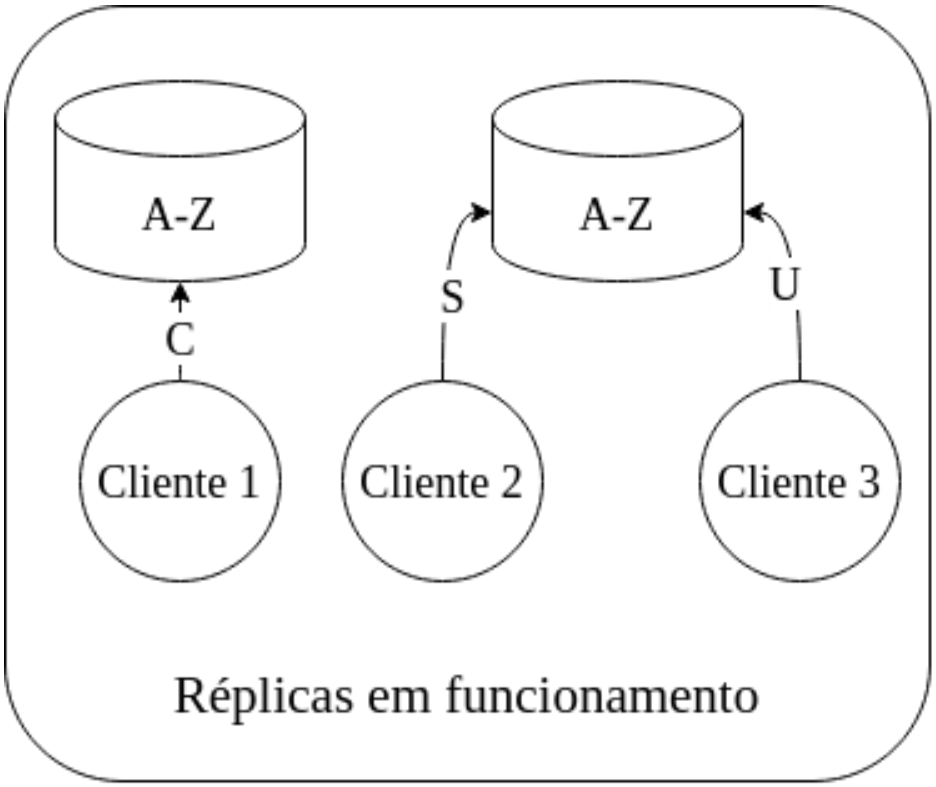
Definimos regras para a distribuição

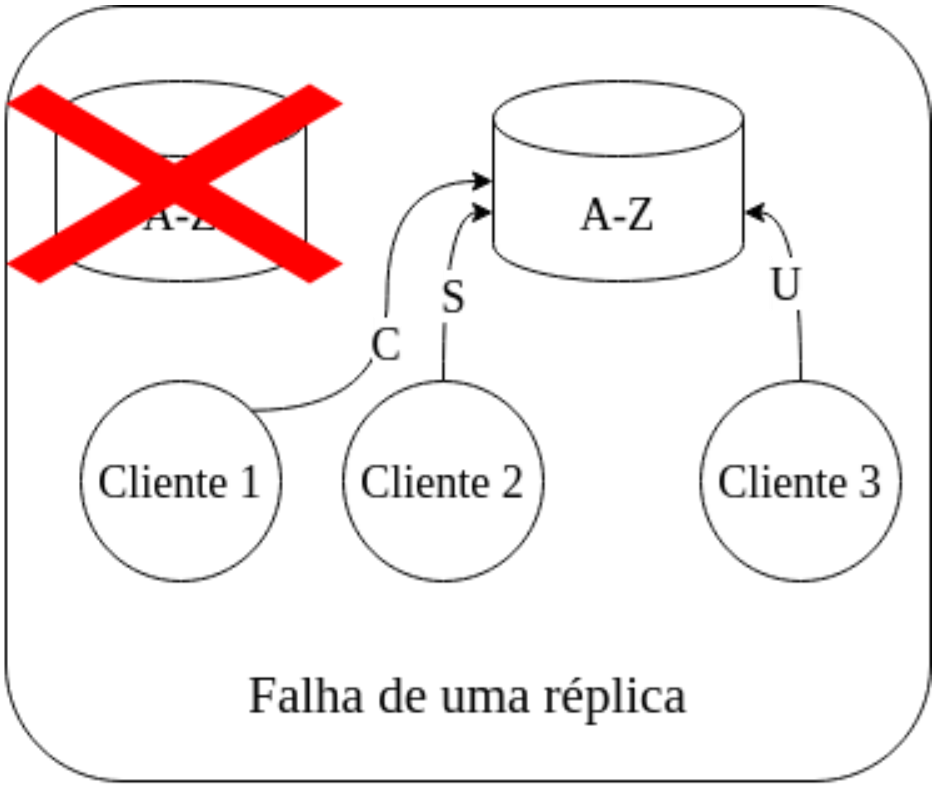


Distribuição de Carga entre Réplicas

Replicação

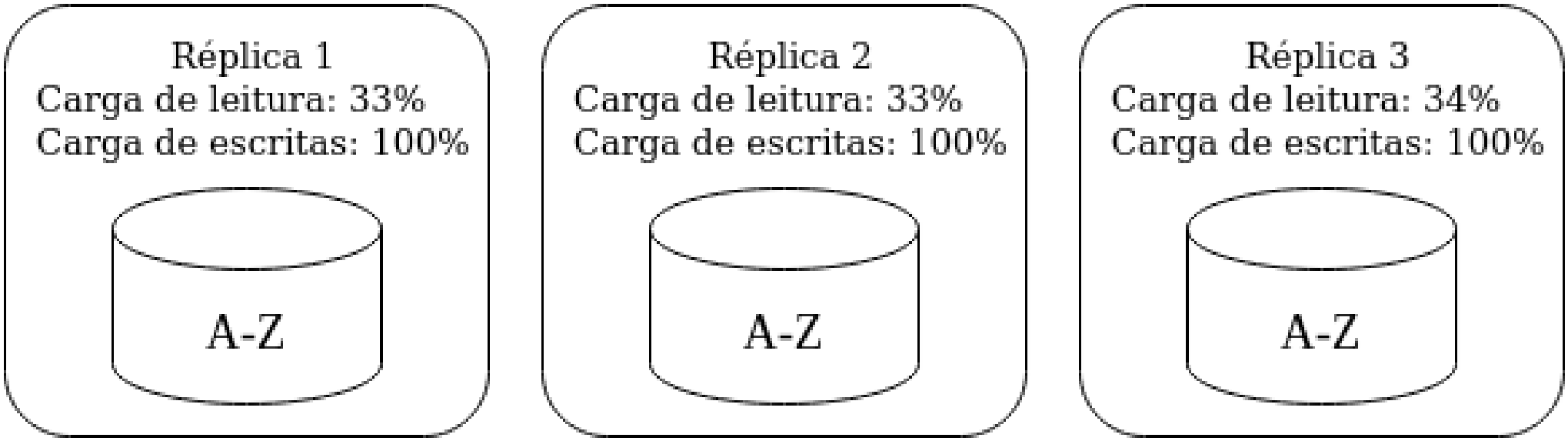
- Diferentes servidores contém o estado do sistema
- Todos os servidores podem atender requisições
- Oferece tolerância a falhas





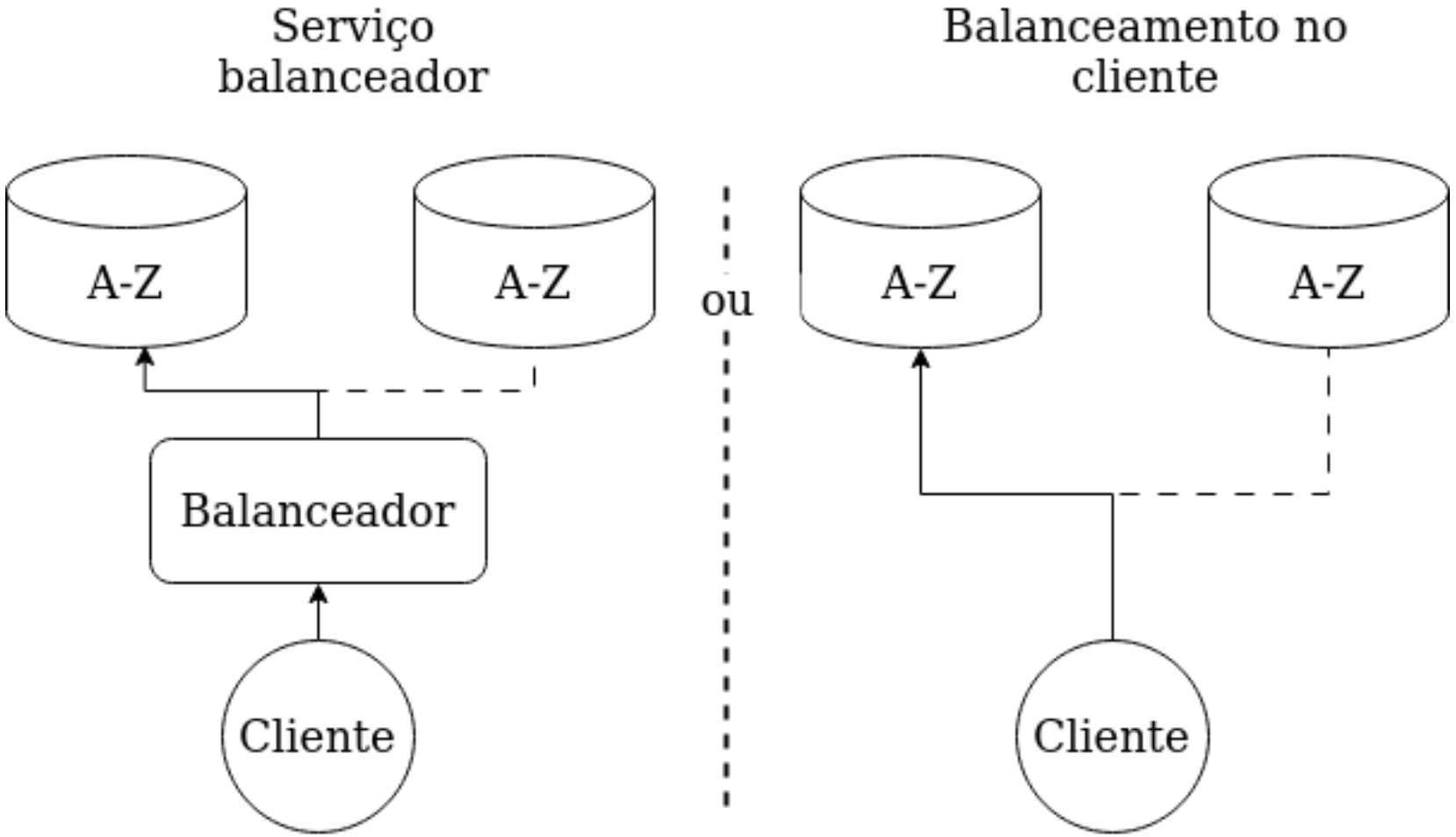
Balanceamento de carga

- Distribuir a carga entre os diferentes servidores
- Evitar excesso trabalho em alguma réplica
- Pode usar diferentes estratégias de balanceamento



Balanceador

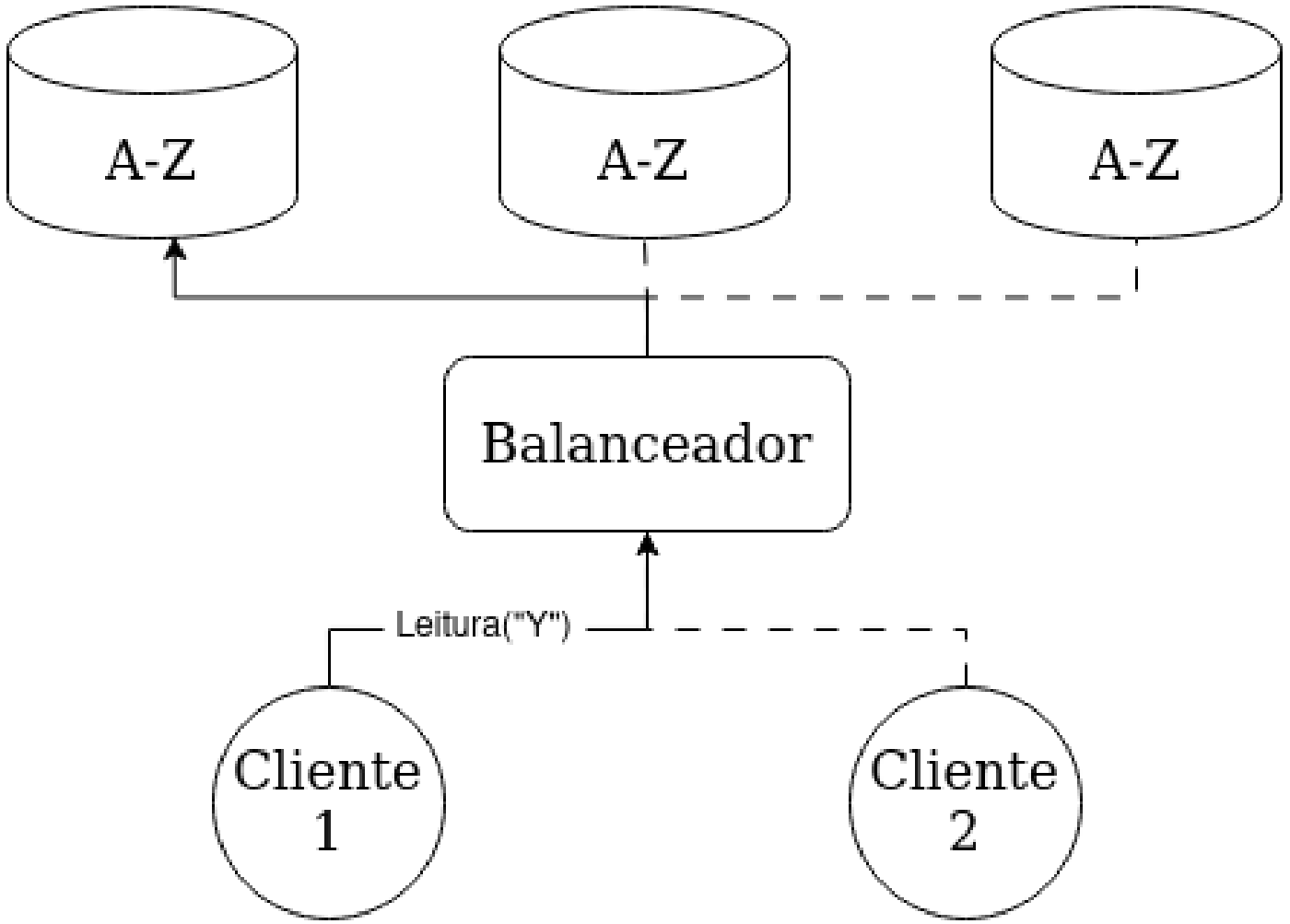
- Pode ser um componente do sistema
- Ou o balanceador pode fazer parte do cliente



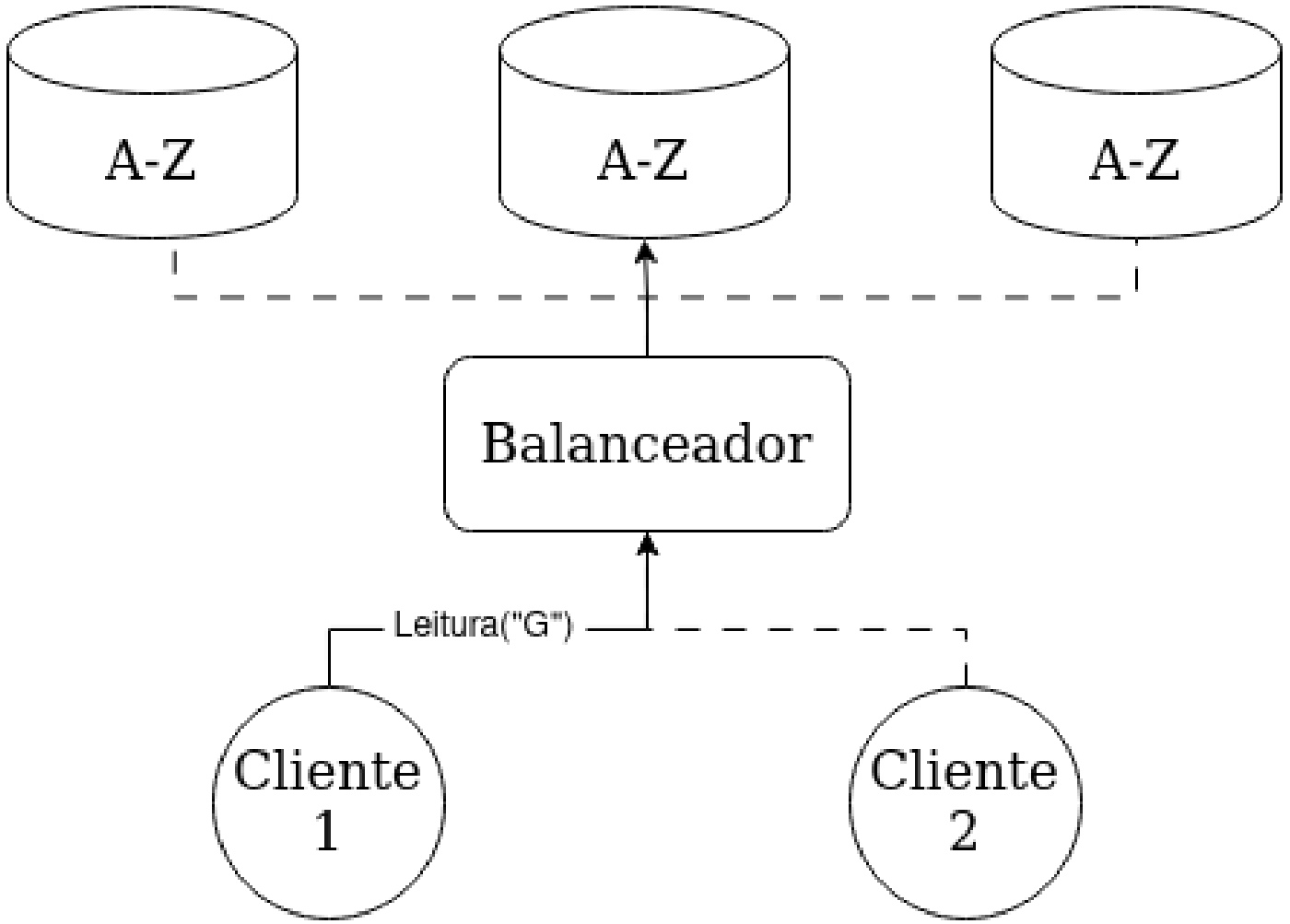
Estratégias de balanceamento

- Distribuição circular (*Round-Robin)*
- Dispersão (*Hash)*
- Intervalos
- Lista
- Temporizado
- …

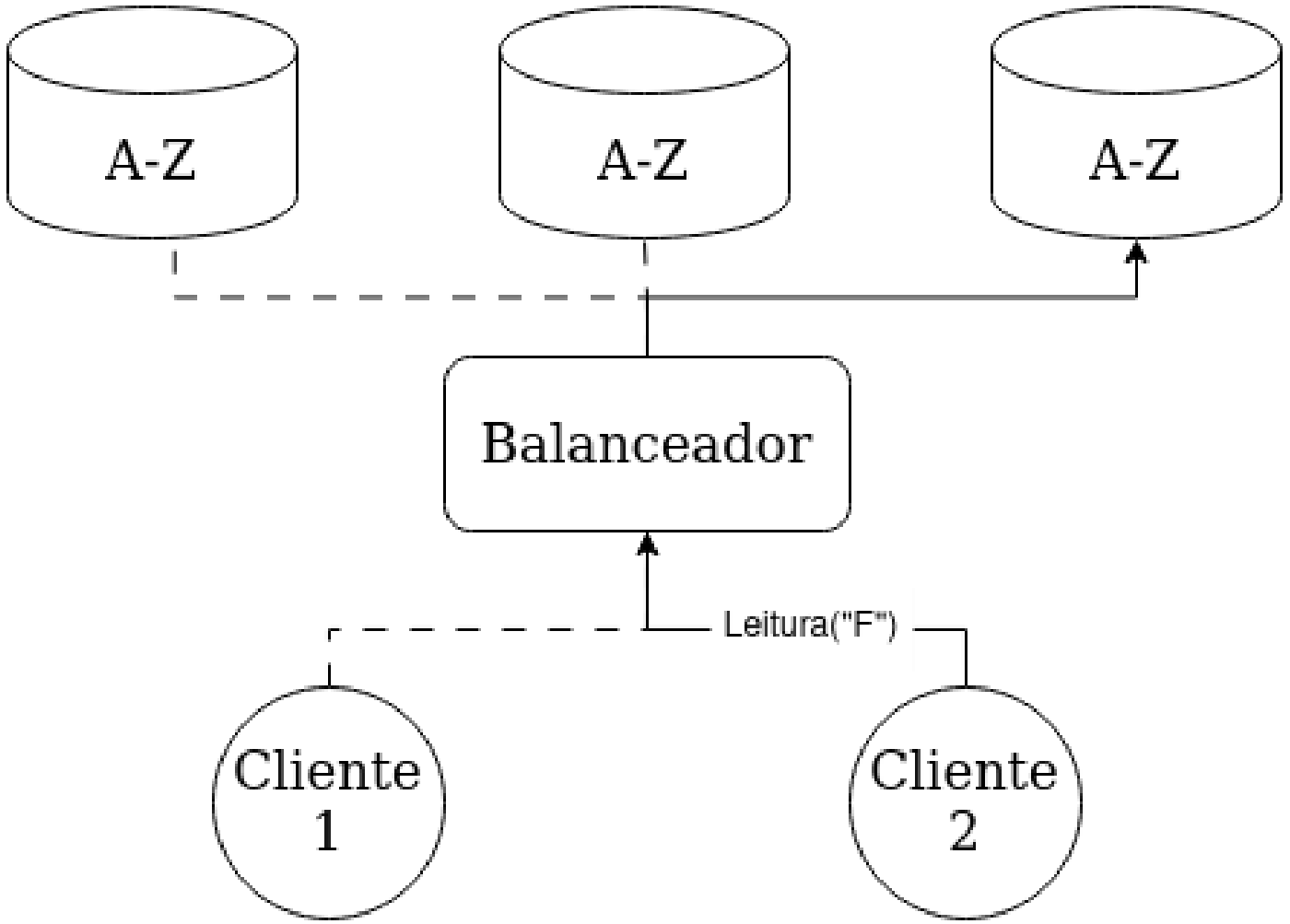
Distribuição circular (Round-robin)



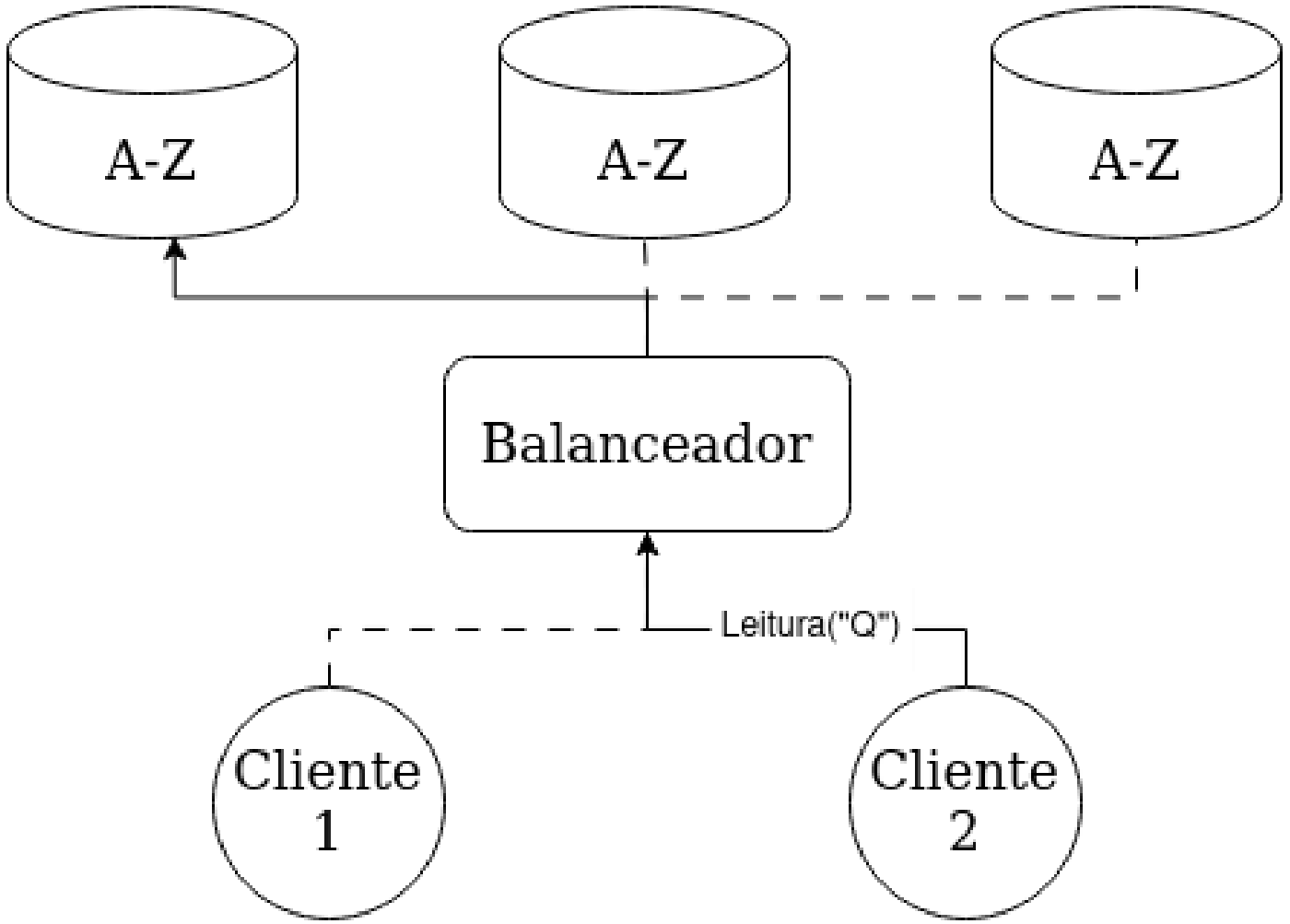
Distribuição circular (Round-robin)



Distribuição circular (Round-robin)



Distribuição circular (Round-robin)



Desafios

- Replicação
- Estratégia de balanceamento adequada
- Escalabilidade e latência com o balanceador

Scatter / Gather

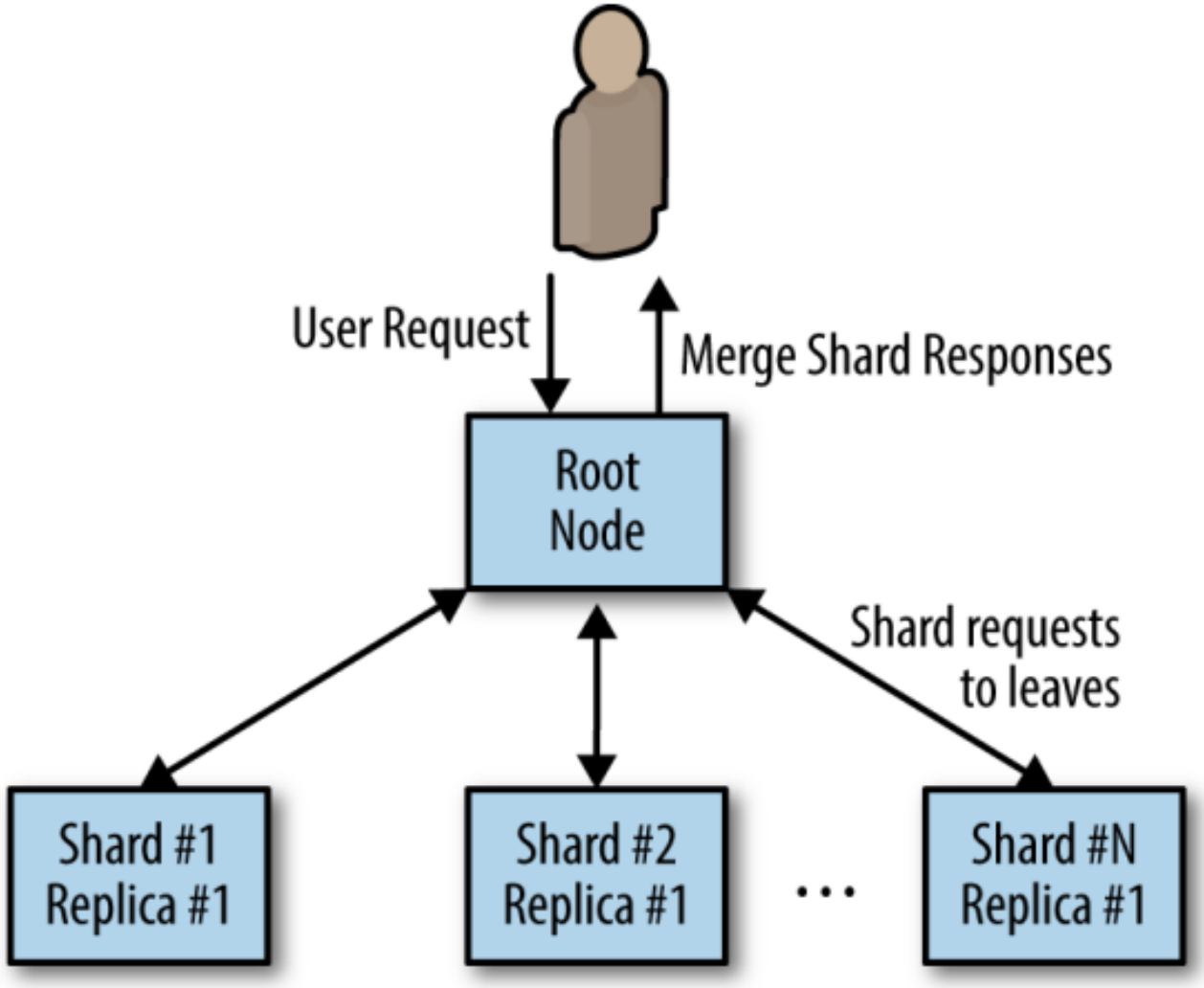
Situações recorrentes

- Sistemas é replicado ou particionado (*sharded)*
- Partes do trabalho podem ser feitas paralelamente
- É necessário compor uma resposta com os trabalhos parciais

- Requisições podem ser divididas
- Os dados necessários para atender as requisições estão em servidores diferentes
- É necessário compor uma resposta com as requisições parciais

Scatter / Gather

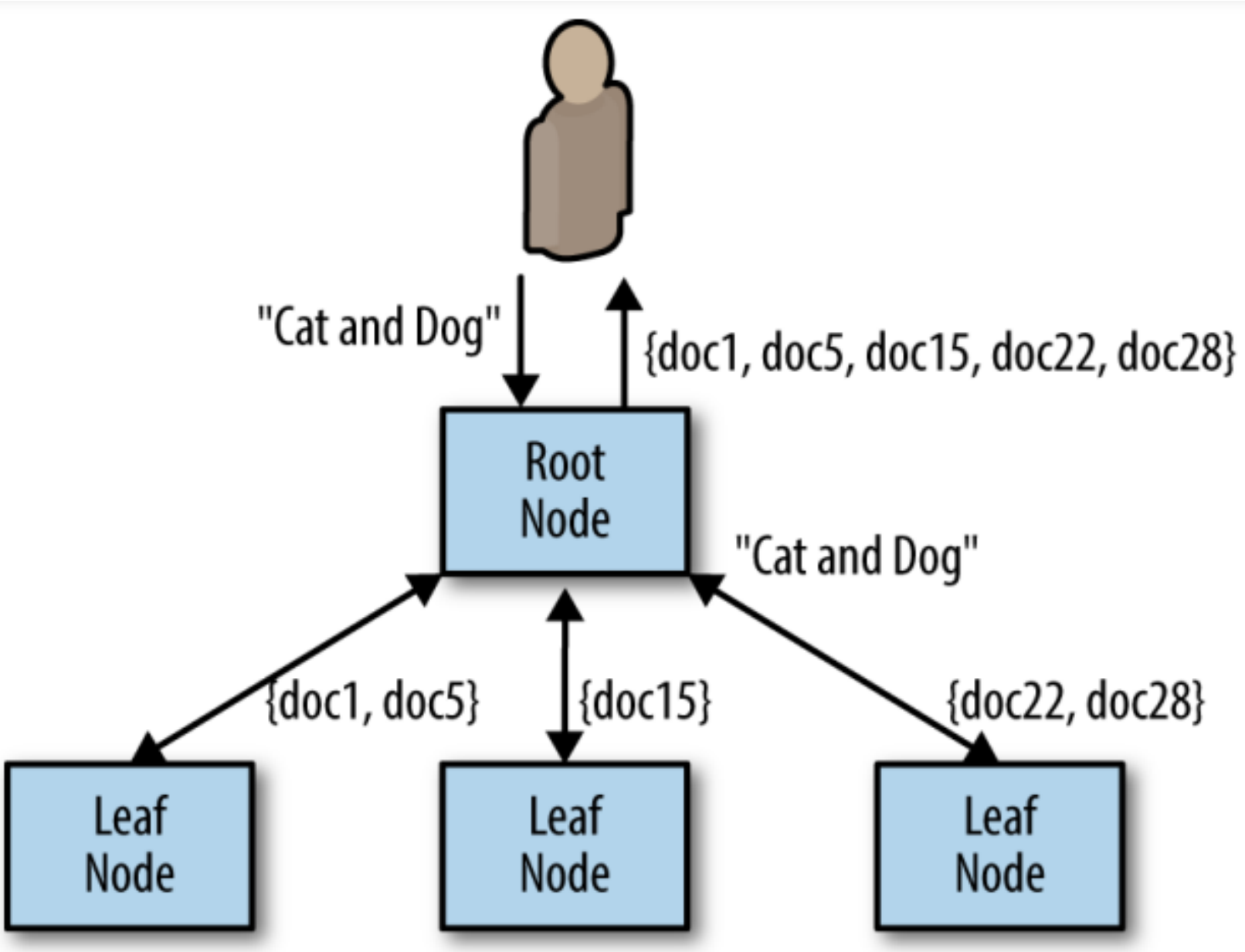
Divisão do atendimento em diferentes servidores



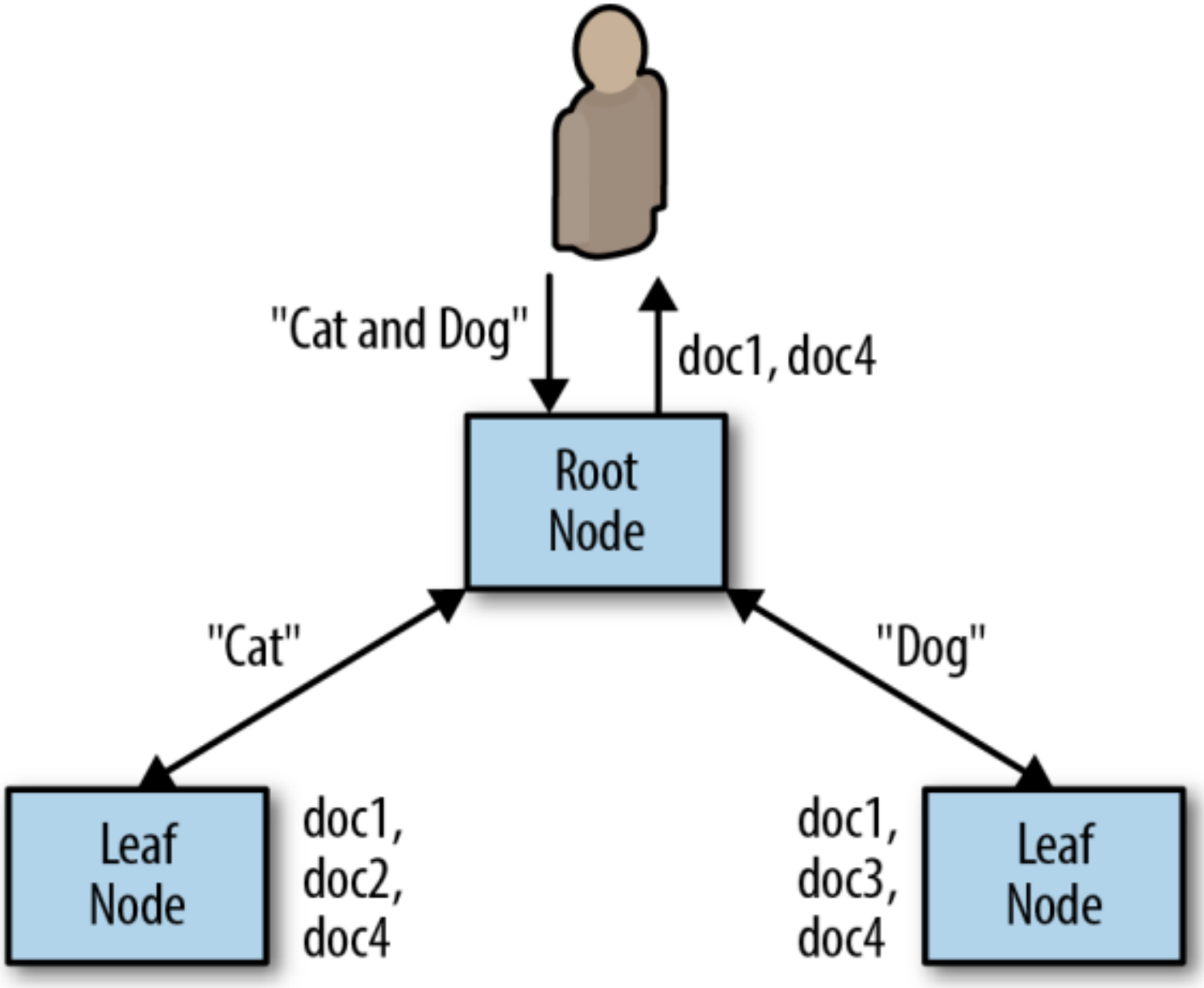
Scatter / Gather

- Pode ser um componente do sistema
  - ou fazer parte do cliente
- Recebe as requisições
- Distribui as requisições para as réplicas ou *shards*
- Combina as respostas e responde o cliente

Scatter / Gather com *sharding*



Scatter / Gather com replicação



Desafios

- Como dividir as requisições
- Escalabilidade e latência da raíz
- Falhas da raíz
- Recomposição das requisições

Referências

- https://martinfowler.com/bliki/CircuitBreaker.html
- Brendan Burns. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. O'Reilly Media, 2018.
- Andrew S Tanenbaum and Maarten Van Steen. Distributed Systems. Maarten Van Steen, 4th edition, January 2023.
- https://learn.microsoft.com/en-us/azure/architecture/patterns/