Replicação

FELIPE CUNHA

Melhorando serviços com dados replicados

Balanceamento de carga

- carga de trabalho é compartilhada entre servidores amarrando-se vários IPs a um único nome de DNS. Endereços IP são retornados utilizando uma política tipo round-robin.

Tolerância à falha

- no modelo fail-stop, se f de f + 1 servidores cai, pelo menos um mantém-se ativo e provendo o serviço

Aumento da disponibilidade

- serviço pode não estar disponível quando servidores falham ou quando a rede é particionada

Melhorando serviços com dados replicados

Aumento da disponibilidade

P: probabilidade de falha de um servidor

1 – P: disponibilidade do serviço

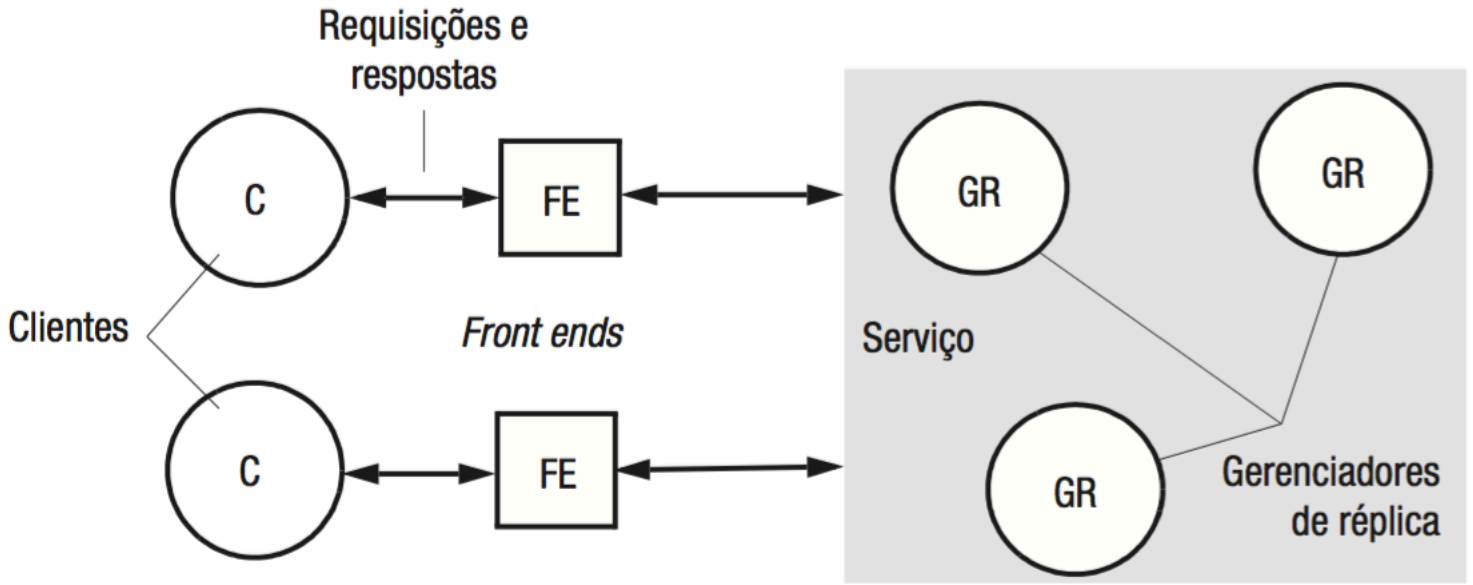
Por exemplo, se P = 5% => serviço está disponível 95% do tempo.

P^n: probabilidade de *n* servidores falharem

$1 - P^n$: disponibilidade do serviço

por exemplo, se P = 5%, n = 3 => serviço está disponível 99.875% do tempo

Modelo arquitetural básico para o gerenciamento de dados replicados



Transparência de replicação: usuário/cliente não precisa saber que existem diversas cópias do recurso

Consistência de replicação: os dados são consistentes entre todas as réplicas, ou estão no processo de se tornarem consistentes

Gerenciamento de replicação

Comunicação das requisições

- requisições podem ser feitas a um RM ou a múltiplos RM

Coordenação: os RM devem decidir:

- se a requisição será aplicada

- a ordem em que as requisições serão aplicadas
  - **ordem FIFO:** se um FE envia uma requisição r e então uma requisição r', então toda RM deve tratar r e depois r'
  - **ordem causal:** se a requisição r "*happens-before*" r', então toda RM deve tratar r e depois r'
  - **ordem total:** se uma RM trata r e depois r', então todas as RM devem tratar r e depois r'

Execução: se os RM tentam executar a requisição

Gerenciamento de replicação

Acordo/Consenso: as RMs tentam alcançar o consenso sobre o efeito da requisição

por exemplo: Two-phase commit através de um coordenador

se bem sucedido, a requisição se torna permanente

Resposta: uma ou mais RM respondem ao FE

no modelo fail-stop, o FE retorna a primeira resposta obtida

Comunicação de grupos

Grupos estáticos: membros são pré-definidos

Grupos dinâmicos: membros podem entrar (*join*) e sair (*leave*) do grupo

- Um serviço de gerenciamento de membros em um grupo mantém visões de grupo (*views*):
  - listas dos membros atuais do grupo
  - não é uma lista mantida por um membro, mas cada membro possui sua *view* local

Views

- Uma *view* $V_p(g)$ é o entendimento do processo p de seu grupo (lista de membros)
  - Exemplo: $V_{p.0}(g) = \{p\}$, $V_{p.1}(g) = \{p, q\}$, $V_{p.2}(g) = \{p, q, r\}$, $V_{p.3}(g) = \{p,r\}$
- Uma nova view é disseminada através do grupo sempre que um membro entra ou sai do grupo
  - membros que detectarem uma falha de um outro membro, envia um multicast confiável notificando a mudança da *view* (requer ordem causal ou total para multicasts)
- Mensagens enviadas em uma *view* devem ser enviada a todos os membros do grupo

Views

Requisitos para entrega de uma *view*

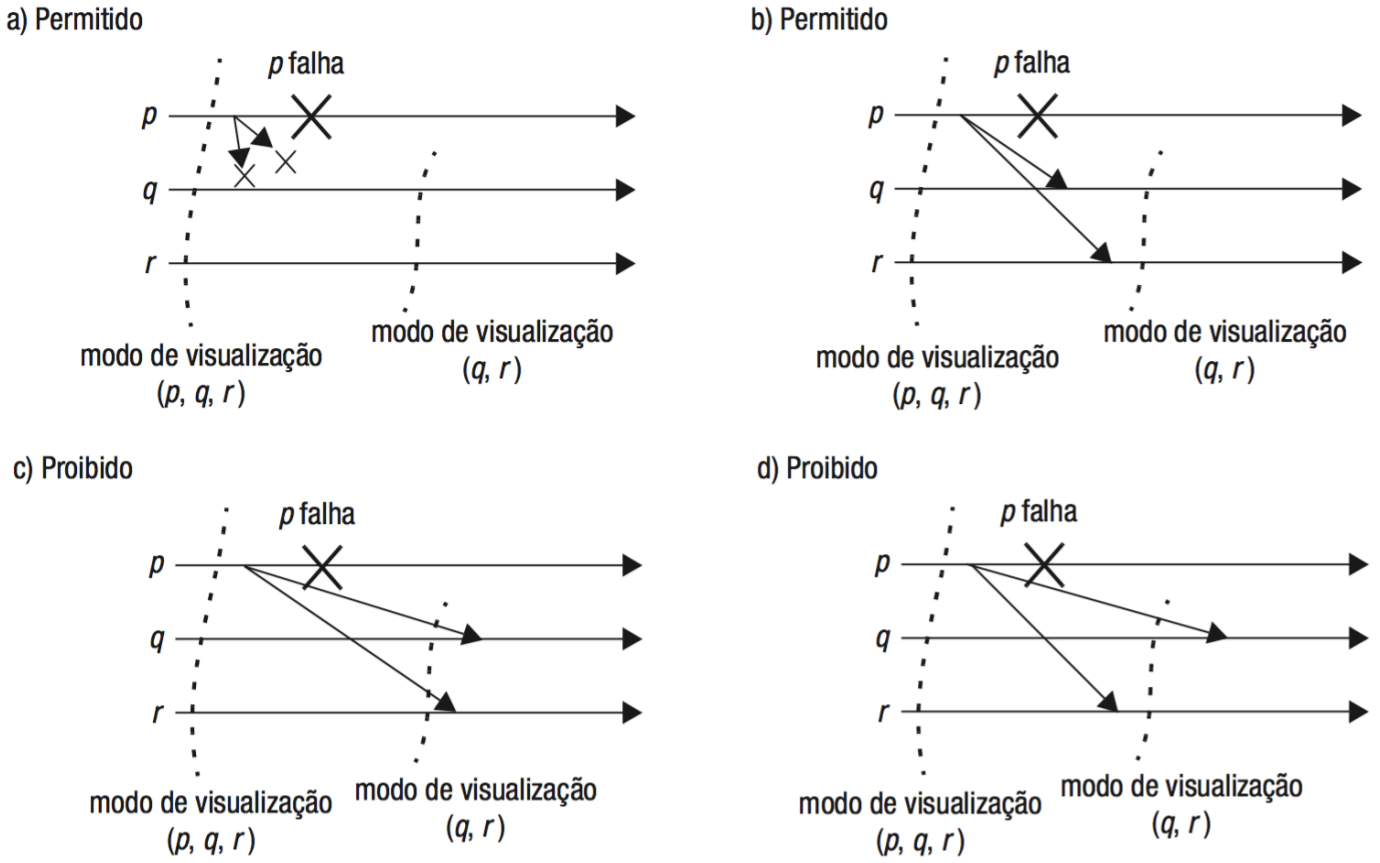
- **Ordem**: se *p* entrega uma $v_i(g)$, e então uma $v_{i+1}(g)$, então nenhum outro processo *q* envia $v_{i+1}(g)$ antes de $v_i(g)$
- **Integridade**: se *p* envia $v_i(g)$, então *p* está na *view* $v_i(g)$
- **Não trivialidade**: se o processo *q* entra em uma *view* e se torna alcançável pelo processo *p*, então, eventualmente, *q* estará sempre presente nas *views* entregues a *p*

Comunicação síncrona em uma view

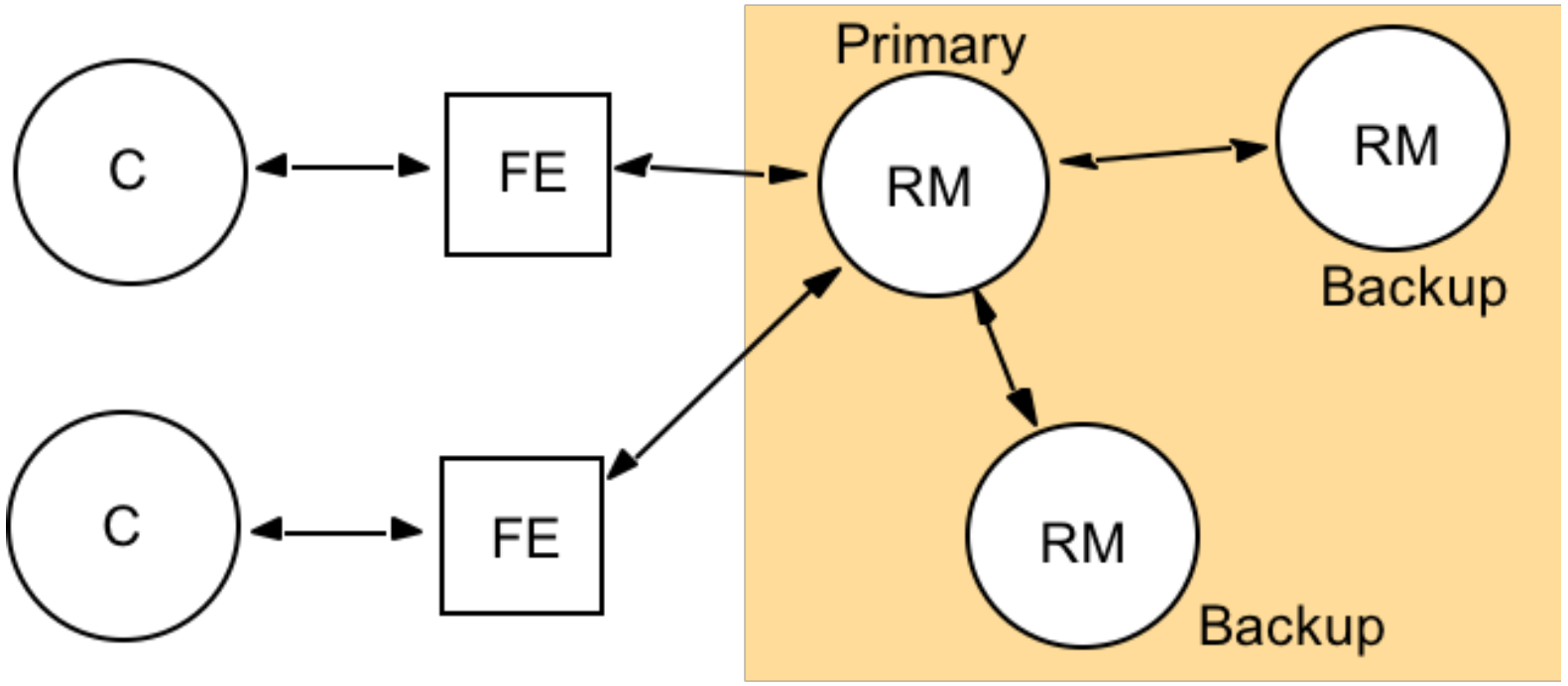
- Usa serviço de comunicação de grupo + multicast confiável

- Garantias providas pelo protocolo multicast confiável:
  - **Integridade**: se *p* enviou a mensagem *m*, *p* não irá enviar *m* novamente, e $p \in grupo(m)$.
  - **Validade**: processos corretos sempre entregam todas as suas mensagens: Se *p* entrega a mensagem m na *view* $v(g)$, e algum processo $q \in v(g)$ não entrega a mensagem m na *view* $v(g)$, então a próxima view $v'(g)$ entregue a *p* não irá incluir *q*.
  - **Acordo/Consenso**: processos corretos entregam o mesmo conjunto de mensagens em uma *view*: se *p* entrega *m* em *V*, e então entrega *V'*, então todos os processos em $V \cap V'$ entregam *m* na *view V*

View-synchronous group communication



Modelo de tolerância à falhas baseado em replicação passiva (backup primário)



Replicação passiva (backup primário)

Comunicação: a requisição é encaminhada para o RM primário e possui um único identificador de requisição

Coordenação: o RM primário recebe todas as requisições atomicamente, em ordem, checa o id (reenvia resposta se não é novo identificador)

Execução: RM primário executa e armazena as requisições

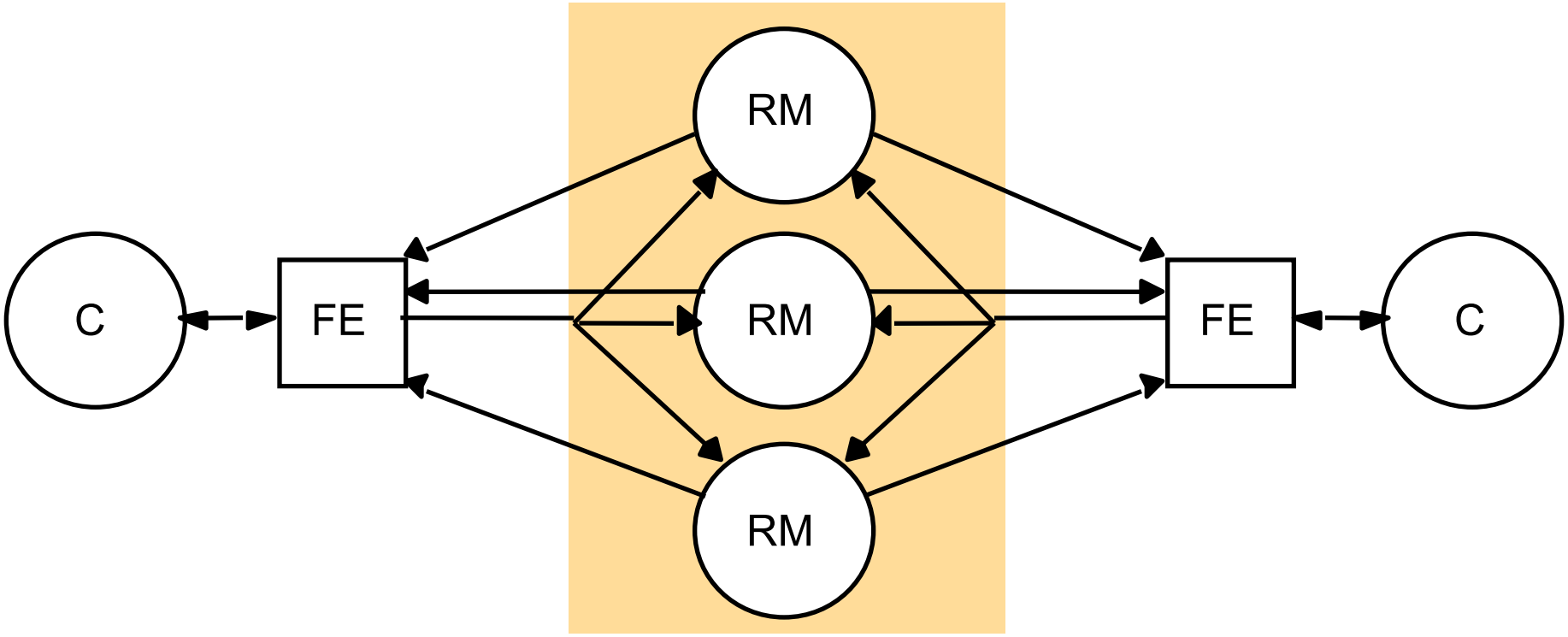
Acordo/Consenso: se é atualização, RM primário envia estado atualizado/resultado do objeto, identificador de requisição e resposta para todos os RM de backups (1-phase commit).

Resposta: RM primário envia resposta ao front end

Tolerância à falhas na replicação passiva

- Se RM primário falha, um backup se torna o líder por eleição e as RM que sobreviveram concordam sobre o conjunto de operações que foram realizados até o ponto em que o novo líder assume
  - requisito é alcançado se as RM estão organizadas como um grupo e a réplica primária utiliza visão síncrona de grupo para comunicar updates

Replicação ativa



Replicação ativa

Comunicação: a requisição contém um identificador único e é enviada por multicast confiável ordenado a todos os RM

Coordenação: comunicação de grupo garante que as requisições são entregues a cada RM na mesma ordem (pode ser em instantes físicos diferentes)

Execução: cada réplica executa a requisição (réplicas corretas retornam a mesma resposta)

Acordo/consenso: não é necessário acordo, devido às primitivas semânticas do multicast

Resposta: cada réplica envia a resposta diretamente para o front end

Tolerância à falhas na replicação ativa

RM exercem papeis equivalente -> respondem a uma sequencia de requisições da mesma forma

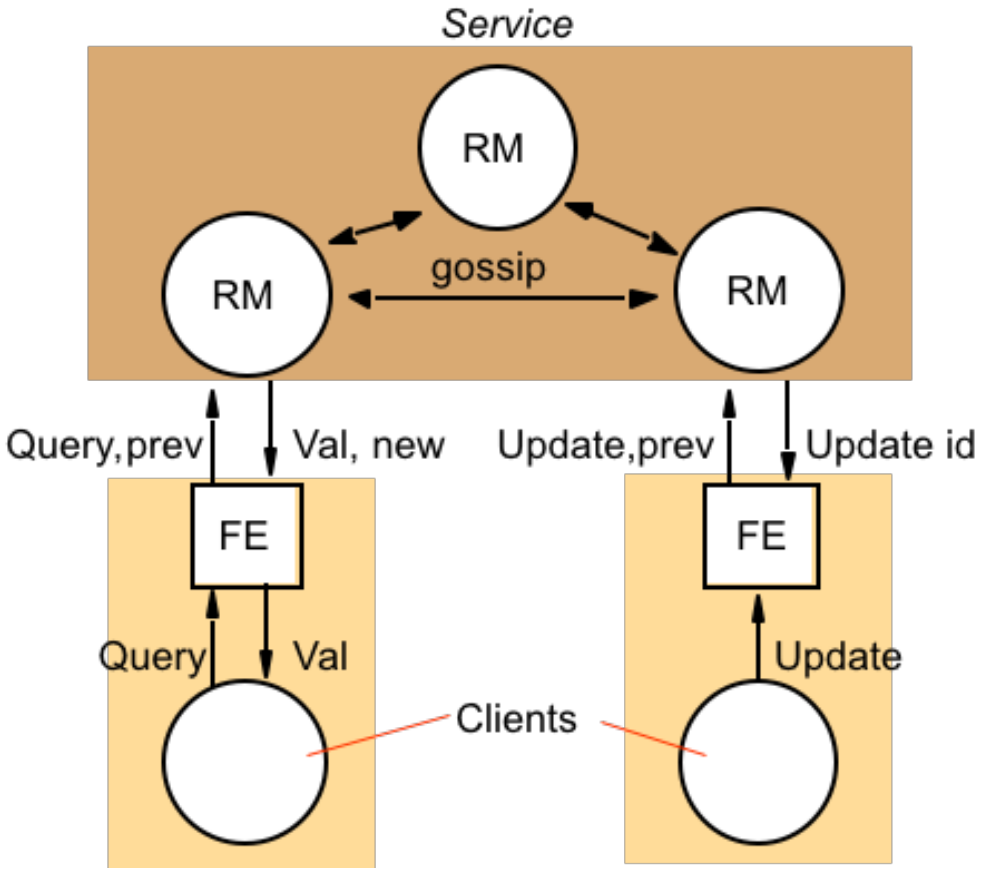
Se alguma RM cai, o estado é mantido pelas demais RMs corretas

Implementa consistência sequencial

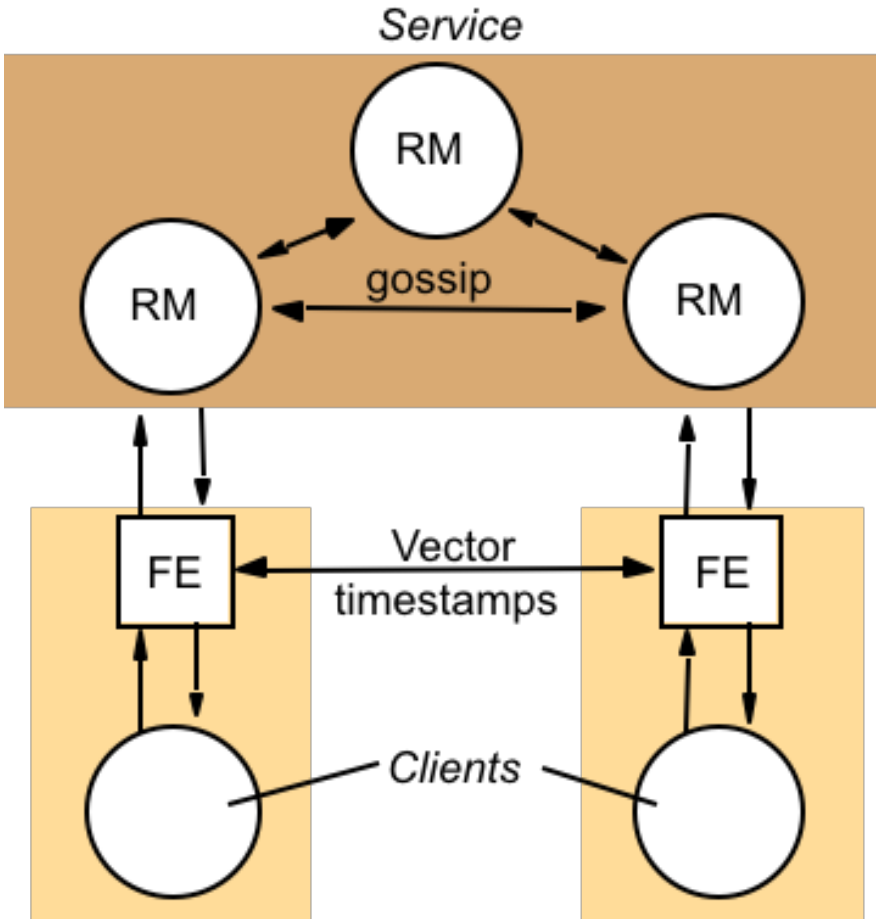
Ordenação total de requisições garante que todas as réplicas executem a mesma sequencia de requisições

Cada requisição do front end é executada na ordem FIFO, porque o FE espera pela resposta para fazer nova requisição

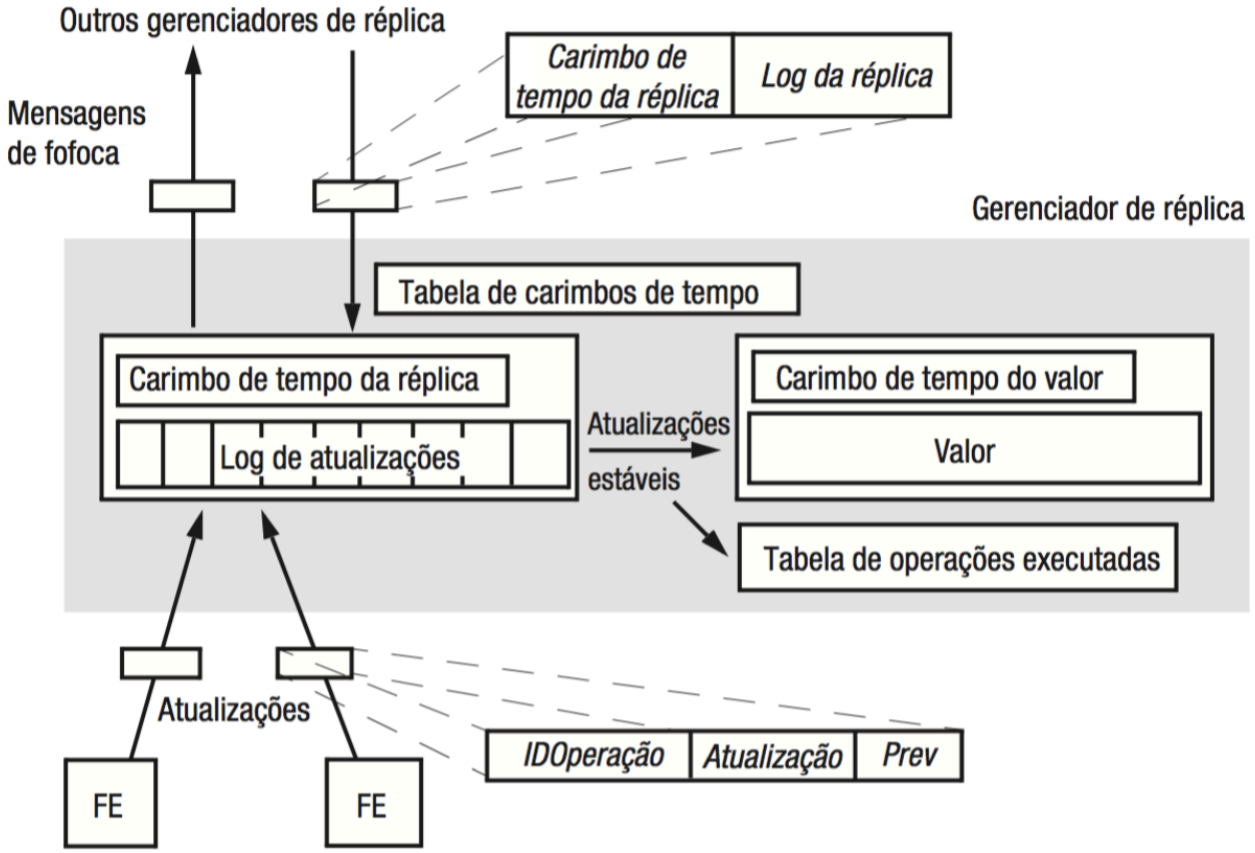
Operações de consulta (query) e updates no serviço gossip



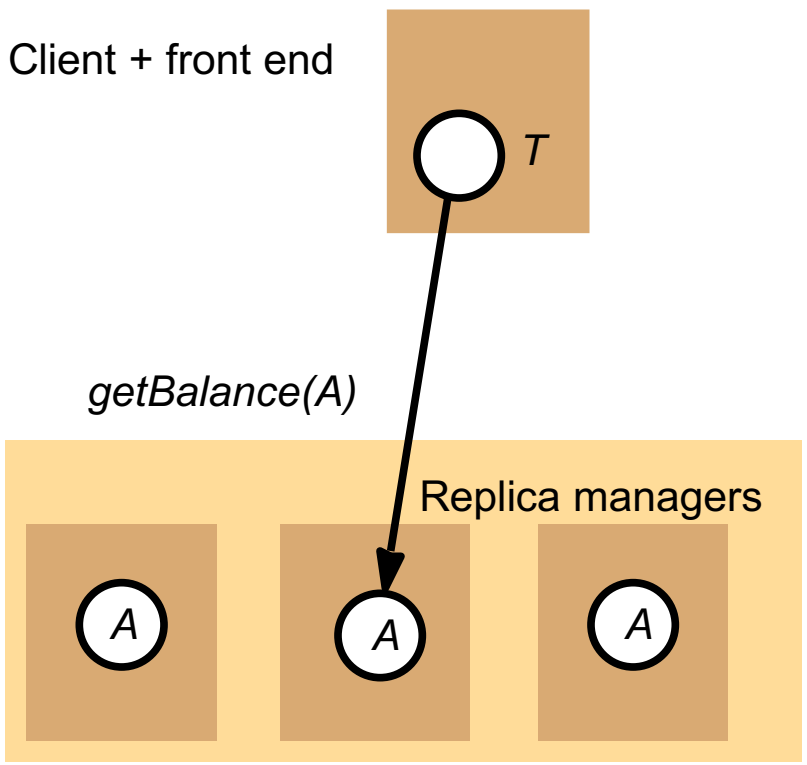
Front ends propagam seus timestamps sempre que um cliente se comunica diretamente

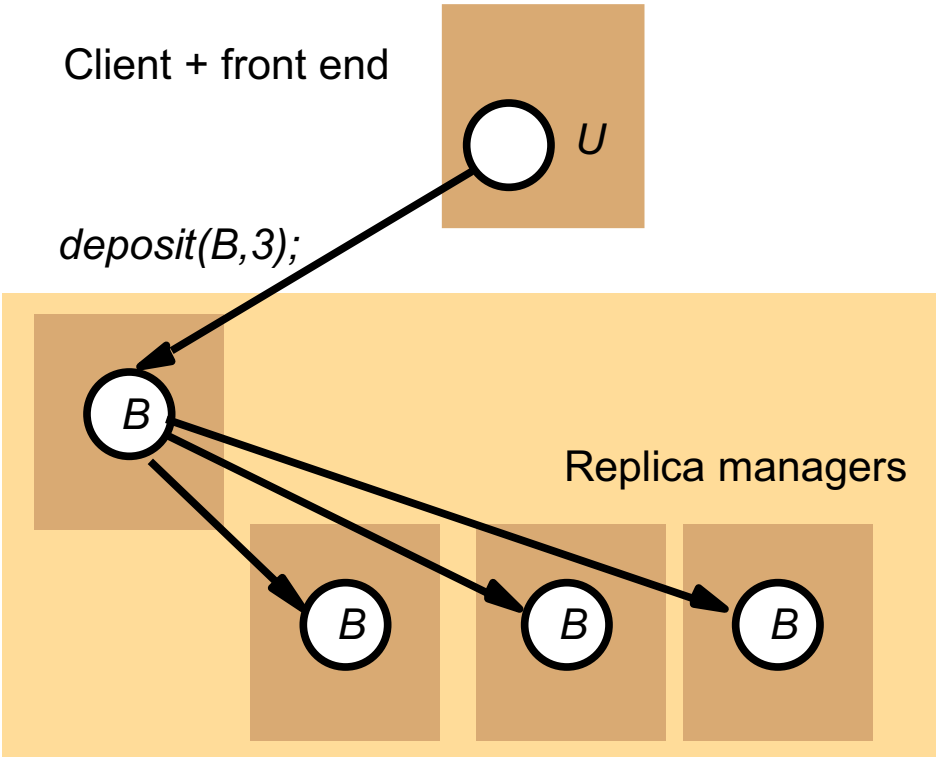


Os componentes principais do estado de uma RM gossip

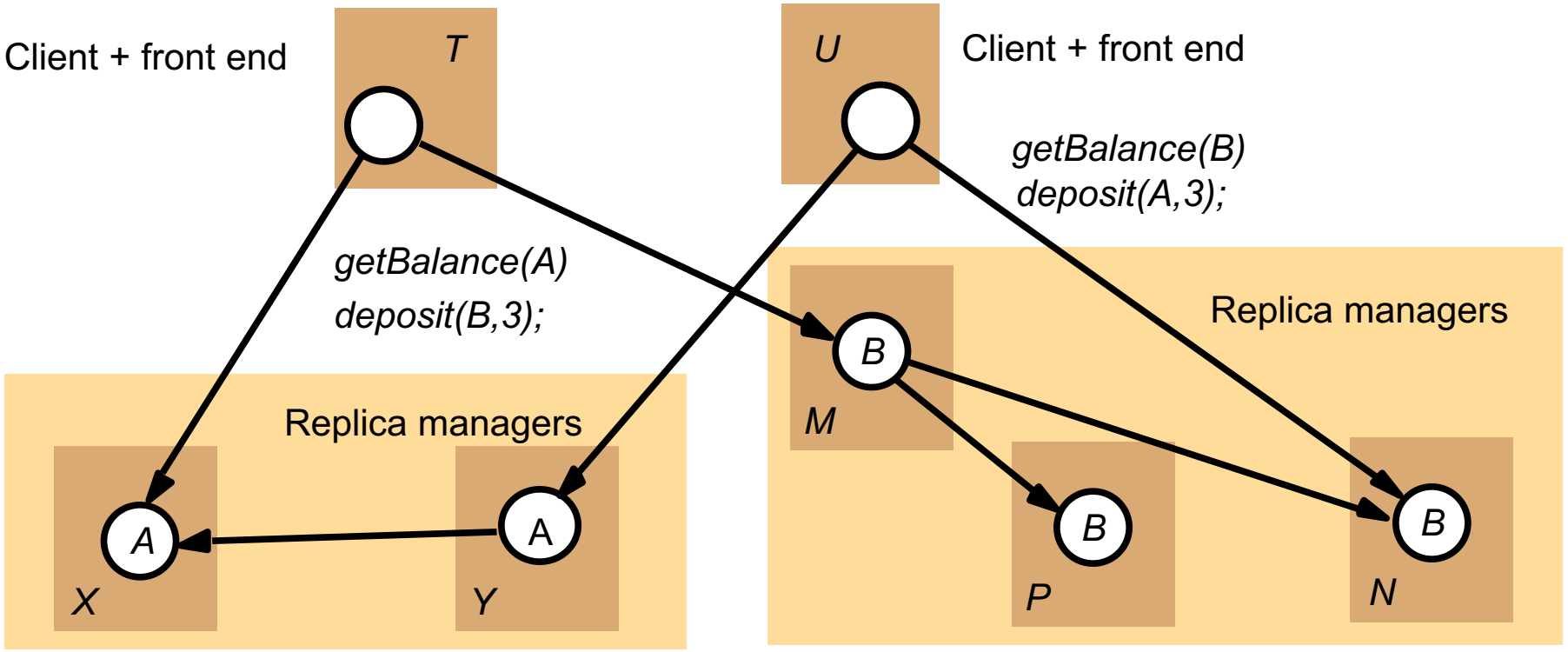


Transações em dados replicados

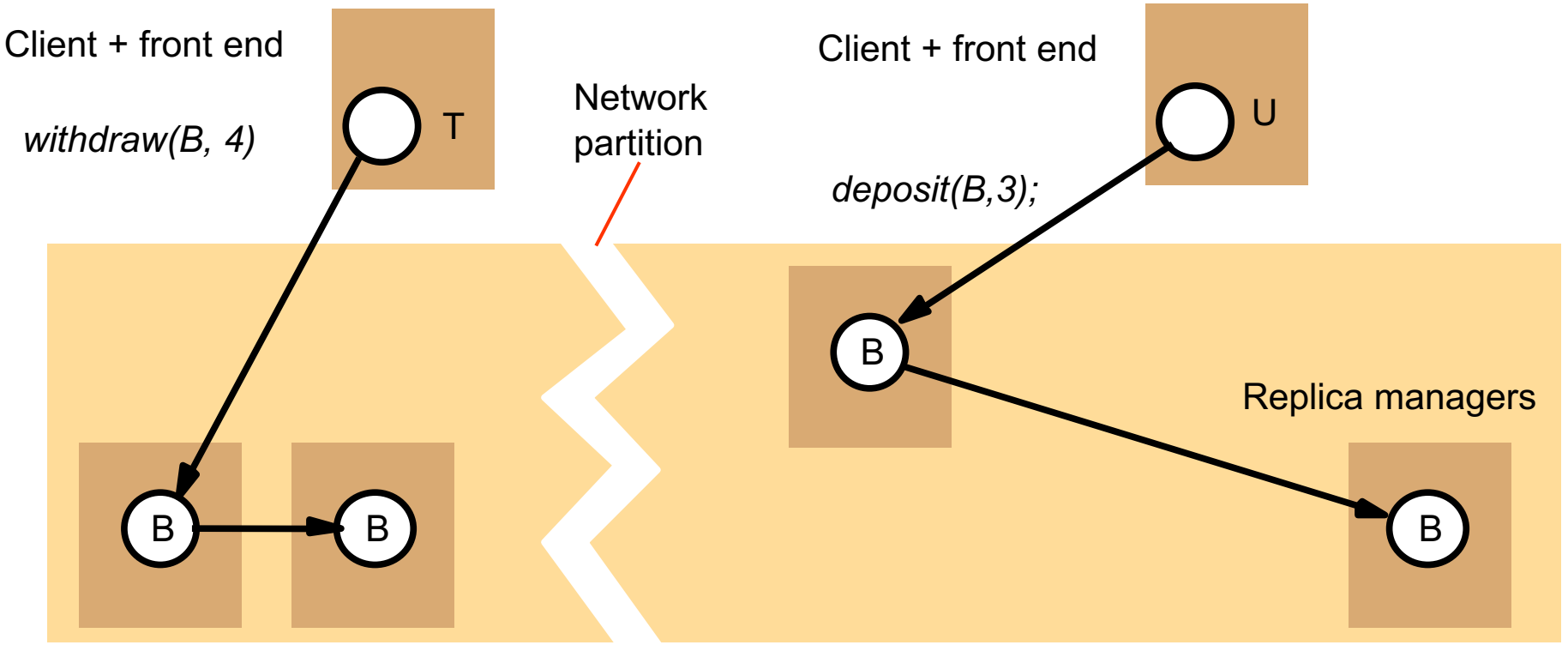




Cópias disponíveis



Partição de rede



Protocolo Quorum Consensus

Sistema de Quóruns:

conjunto de subconjuntos das réplicas, tal que quaisquer dois subconjuntos se intersectam.

Por exemplo: N réplicas -> Quórum: qualquer maioria: $|Q|>N/2$

Cada réplica guarda:

- valor do objeto (registo)
- respectivo timestamp

Protocolo Quorum Consensus

Operação de Leitura

Envia pedido de leitura para todas as réplicas (retransmitindo-o até concluir a operação, para superar falhas temporárias na rede)

Ao receber pedido, réplica responde ao cliente com valor atual de <val,ts>

Cliente aguarda resposta de um quórum

Escolhe valor associado ao maior timestamp

Protocolo Quorum Consensus de Gifford

Vantagens:

- Primeiro protocolo que tolera falhas silenciosas em sistemas assíncronos

- Réplica que falhe temporariamente e recupera está imediatamente pronta para participar

- Ficará naturalmente atualizada quando receber próximo pedido de escrita

Desvantagens:

- À medida que os nós falham, há uma degradação da disponibilidade.

- Quoruns são grandes

Votação Dinâmica

Melhorar Quorum Consensus em redes com particionamento

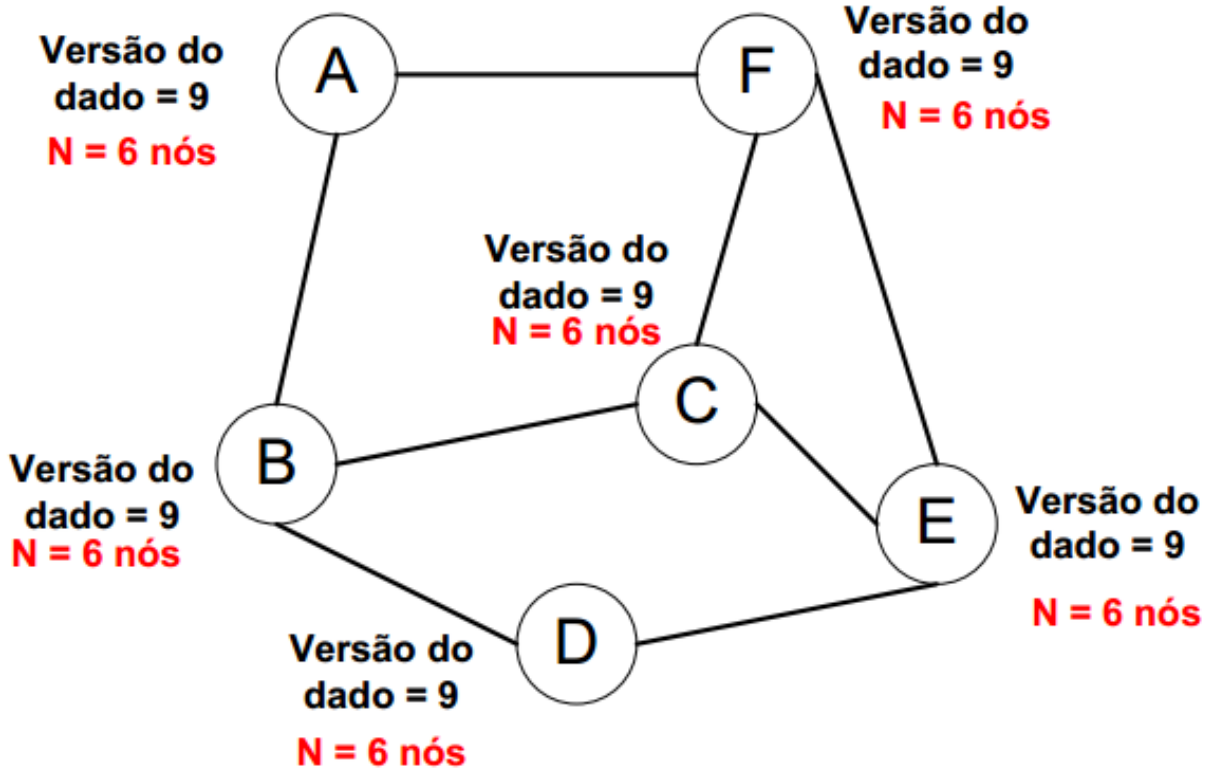
A ideia da votação dinâmica é realizar votações com apenas os nós ativos e concorrentes

Cada nó armazena as seguintes informações:

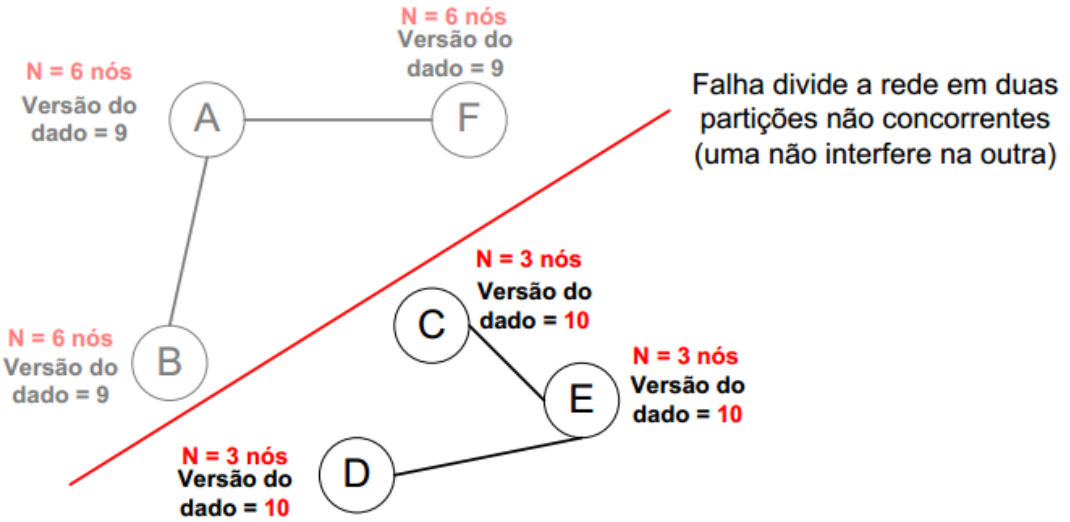
- Versão do dado armazenado

- Número de nós ativos que participaram da última operação

Consenso com votação dinâmica



Consenso com votação dinâmica



Obs.: Quando ocorre o particionamento de uma rede, é necessário garantir que somente uma partição vai sobreviver, pois se uma partição "morre" e depois "retorna", então haverá problemas de inconsistência na versão dos dados.