SISTEMAS DISTRIBUÍDOS

Capítulo 8

Introdução à replicação e consistência

NOTA PRÉVIA

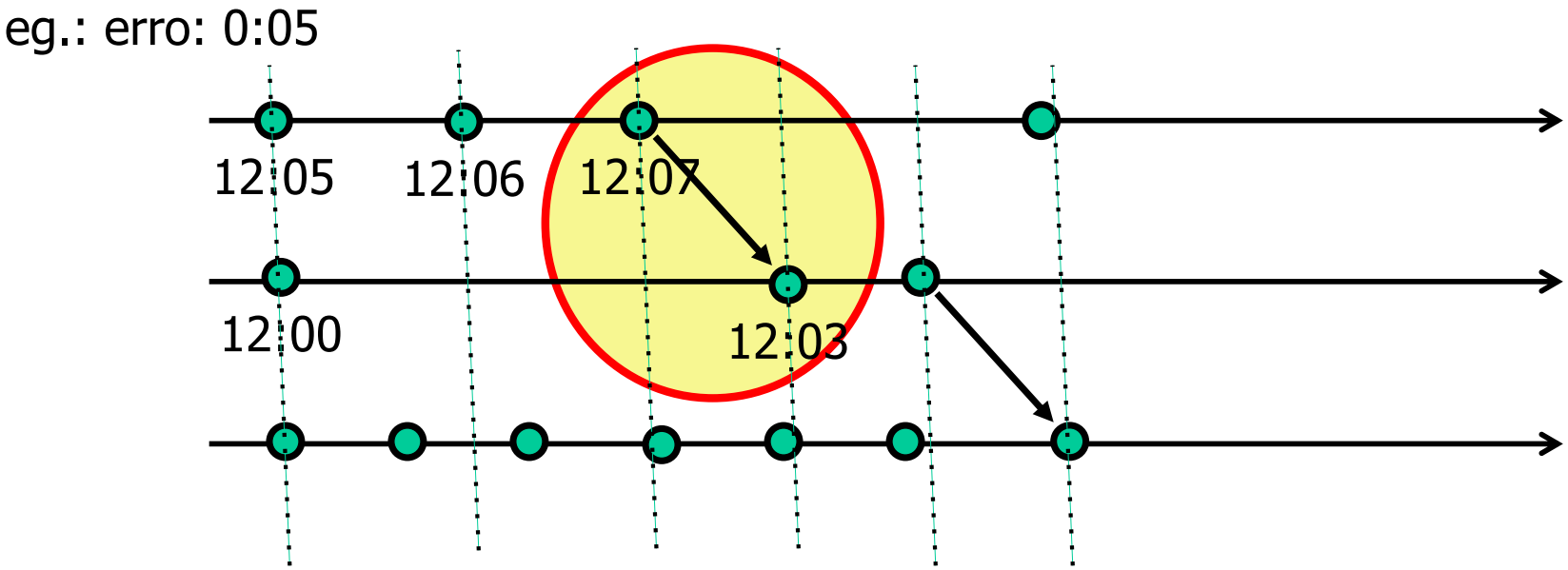
A apresentação utiliza algumas das figuras livro de base do curso

G. Coulouris, J. Dollimore and T. Kindberg,
Distributed Systems - Concepts and Design,
Addison-Wesley, 5th Edition

NA ÚLTIMA AULA....

Num sistema distribuído é impossível sincronizar os relógios de vários computadores a menos dum dado valor.

Assim, é impossível usar o valor do relógio em diferentes computadores para saber a ordem dos eventos.

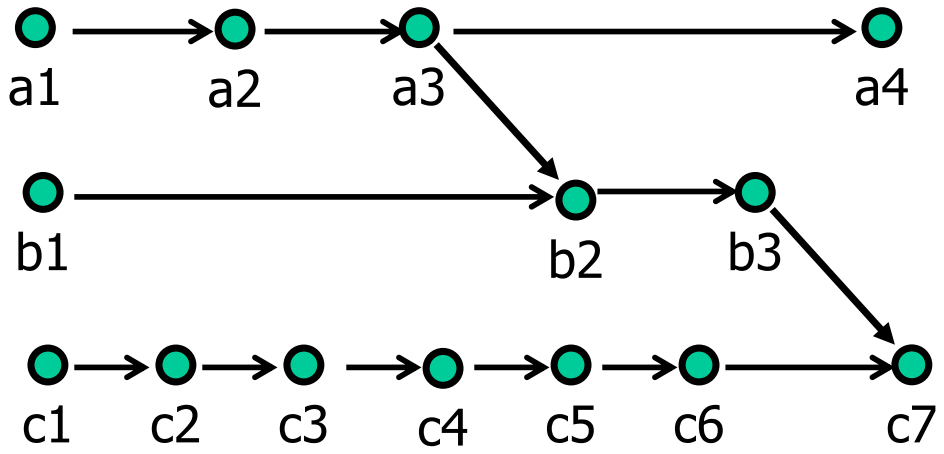


RELAÇÃO "ACONTECEU ANTES" (LAMPORT 1978)

e1 aconteceu antes de e2, sse:

- $\mathbf{e_1}$ e $\mathbf{e_2}$ ocorreram no mesmo processo e $\mathbf{e_1}$ ocorreu antes de $\mathbf{e_2}$
- $\mathbf{e_1}$ e $\mathbf{e_2}$ são, respetivamente, os eventos de enviar e receber a mensagem **m**
- $\exists \mathbf{e_x}$: $\mathbf{e_1} \rightarrow \mathbf{e_x}$ e $\mathbf{e_x} \rightarrow \mathbf{e_2}$ (relação transitiva)

Dois eventos e1, e2 dizem-se concorrentes **(e1||e2)** se
¬ e1 → e2 e ¬ e2 → e1



"ACONTECEU ANTES": MECANISMOS

Dados dois eventos, $\mathbf{e_1}$ e $\mathbf{e_2}$, $\mathbf{e_1 \rightarrow e_2 \Rightarrow C_{lock}(e_1) < C_{lock}(e_2)}$

- ***Relógios lógicos*** (também chamado ***relógio de Lamport***): inteiro

Dados dois eventos, e_1 e e_2, $C_{lock}(e_1) < C_{lock}(e_2) \Rightarrow e_1 \rightarrow e_2$

- ***História causal***: conjunto de (identificadores de) eventos
- ***Relógio vetorial***: array (ou mapa) de inteiros, que são sumário da história causal
- ***Vetor versão***: relógio vetorial, mas registando apenas eventos importantes

NESTE CAPÍTULO DA MATÉRIA....

Introdução à replicação

- Consistência forte. E.g.: primário-secundário
- Consistência fraca.

Caching

- Sistemas de ficheiros distribuídos
- Caching NFS
- Caching CIFS
- Caching Callback Promise

NESTE CAPÍTULO DA MATÉRIA....

Introdução à replicação

- **Consistência forte. E.g.: primário-secundário**
- Consistência fraca.

Caching

- Sistemas de ficheiros distribuídos
- Caching NFS
- Caching CIFS
- Caching Callback Promise

PROBLEMA

As falhas dos componentes são inevitáveis...

Como melhorar a disponibilidade de um serviço na presença de falhas dos componentes?

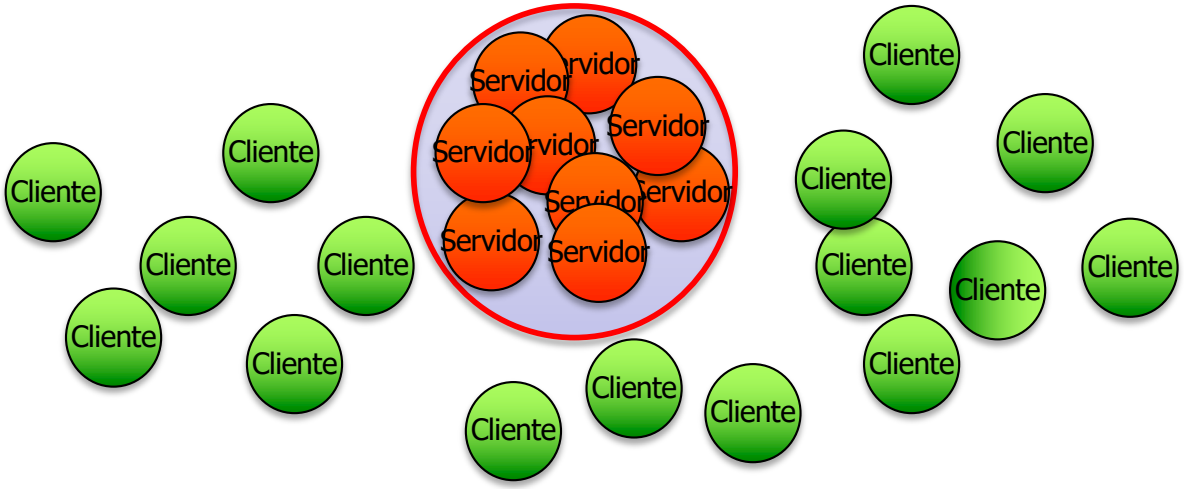
Componentes Replicados

Havendo componentes redundantes para o mesmo serviço, é possível continuar a fornecer o serviço enquanto existirem suficientes réplicas a operar.

VARIANTES DO MODELO CLIENTE/SERVIDOR: SERVIDOR

Cliente/servidor replicado

- Existem vários servidores **idênticos** (i.e. capazes de responder aos mesmos pedidos, de igual forma)



REPLICAÇÃO

Servidores/componentes idênticos…

- Capazes de responder aos mesmos pedidos, **de igual forma**

Igual código mas, sobretudo, **estado equivalente…**

OBJETIVO

Garantir que o estado de vários servidores é mantido equivalente…

... na presença de vários clientes a fazer modificações (e leituras) dos seus estados….

Como fazer?

PRIMEIRA ABORDAGEM: CONSISTÊNCIA FORTE

Objetivo: ter várias réplicas, mas que se comportem como se existisse apenas uma réplica que tolerasse falhas.

Modelo de falhas:

- as mensagens podem-se perder e trocar de ordem;
- um servidor pode falhar por crash, i.e., deixar de responder (e eventualmente recuperar).

Desafios:

1. Garantir que todas as réplicas convergem para o mesmo estado
2. Garantir que os clientes observam sempre a última escrita

PRIMEIRA ABORDAGEM: CONSISTÊNCIA FORTE

Objetivo: ter várias réplicas, mas que se comportem como se existisse apenas uma réplica que tolerasse falhas.

Modelo de falhas:

- as mensagens podem-se perder e trocar de ordem;
- um servidor pode falhar por crash, i.e., deixar de responder (e eventualmente recuperar).

Desafios:

1. **Garantir que todas as réplicas convergem para o mesmo estado**
2. Garantir que os clientes observam sempre a última escrita

REPLICAÇÃO DE MÁQUINA DE ESTADOS

Como *garantir que todas as réplicas convergem para o mesmo estado* ?

1. Ter operações deterministas
   - Em todas as execuções, executar *f* no estado S_a origina o estado S_d , i.e., $f(S_a) = S_d$
2. Executar a mesma sequência de operações em todas as réplicas
   - A partir do mesmo estado inicial S_0, todas as réplicas chegam ao mesmo estado $\mathbf{f_n(...f_2(f_1(S_0)))}$ ao executarem as operações deterministas, f_1, f_2, ..., f_n

REPLICAÇÃO DE MÁQUINA DE ESTADOS

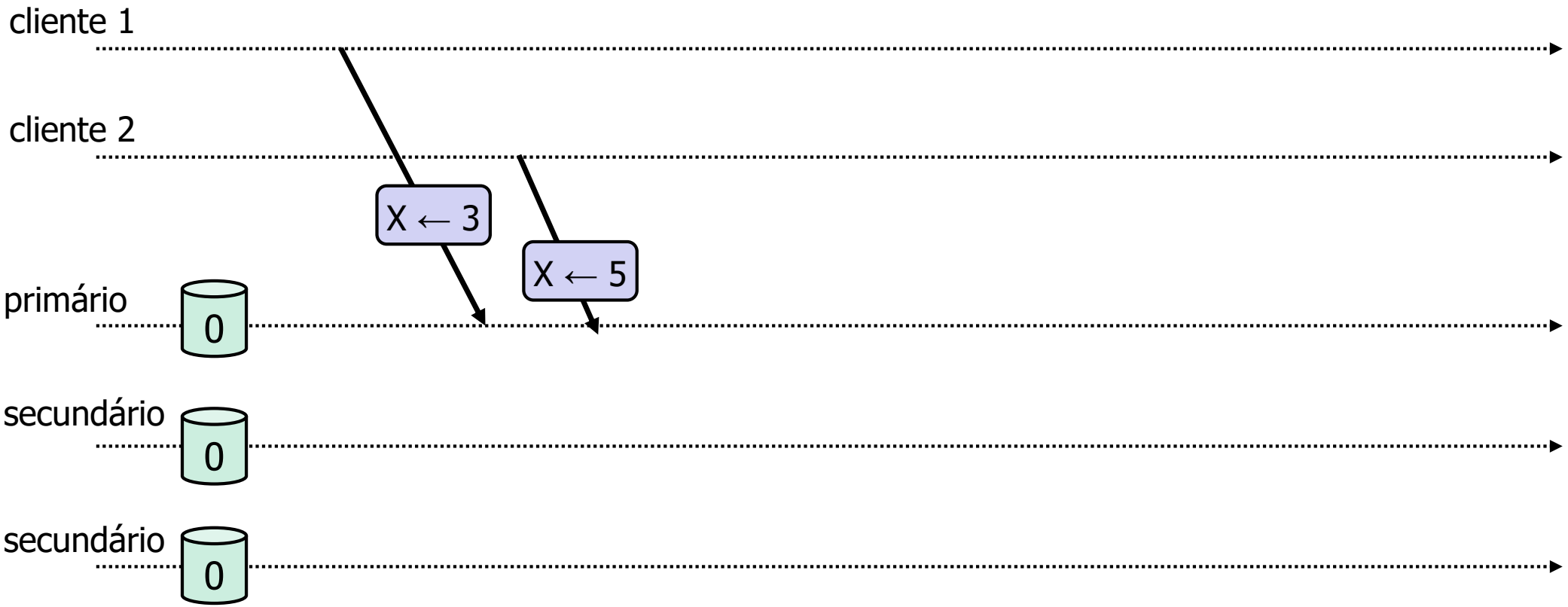
Protocolo **primário/secundário** (passivo)

Primário: mantém a versão oficial dos dados

Secundário: mantém uma cópia da versão do primário

PRIMÁRIO / SECUNDÁRIO : ESCRITA

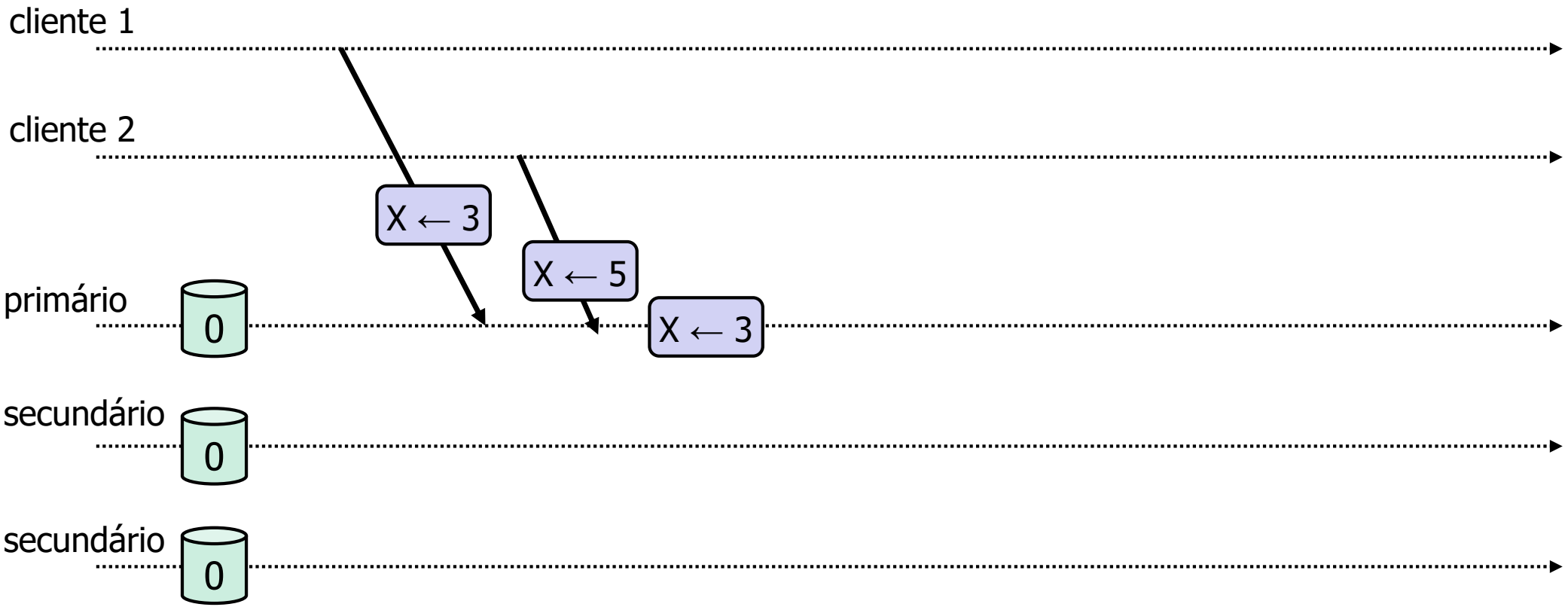
Cliente envia escrita para o primário



PRIMÁRIO / SECUNDÁRIO : ESCRITA

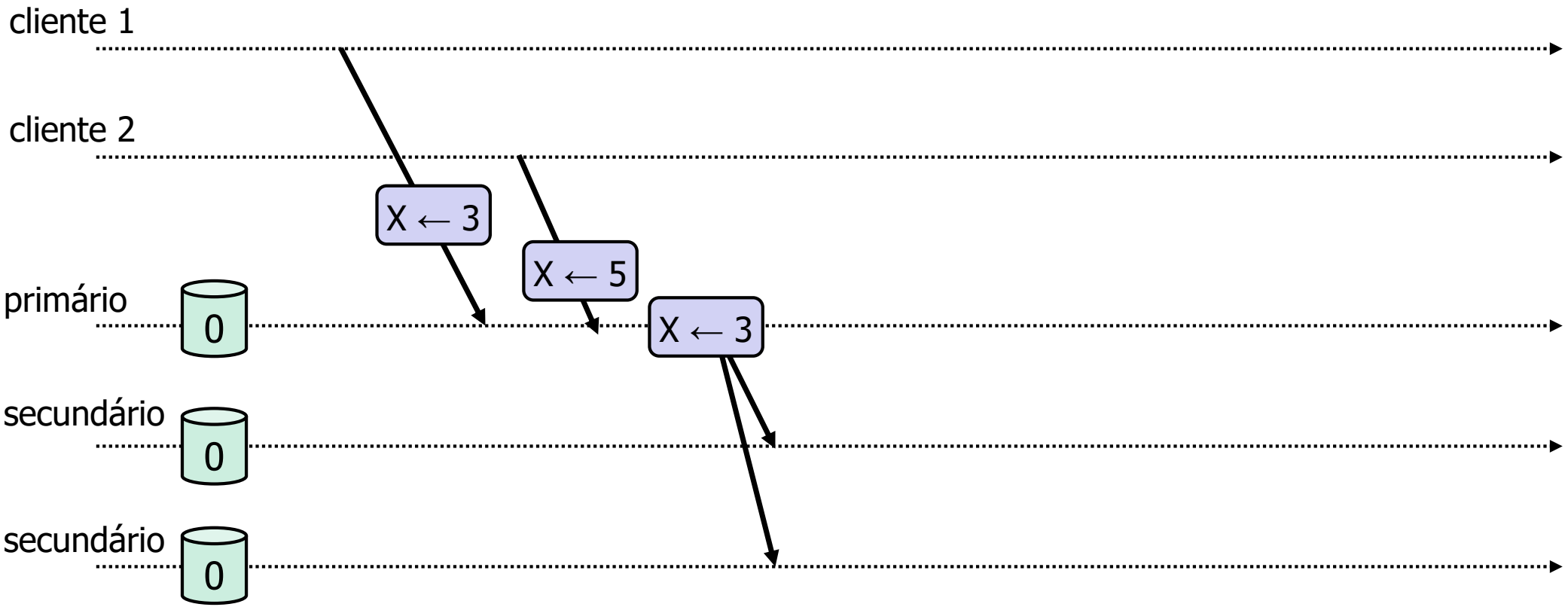
Cliente envia escrita para o primário

Primário **serializa as escritas** e executa-as ordenadamente



PRIMÁRIO / SECUNDÁRIO : ESCRITA : EXECUÇÃO

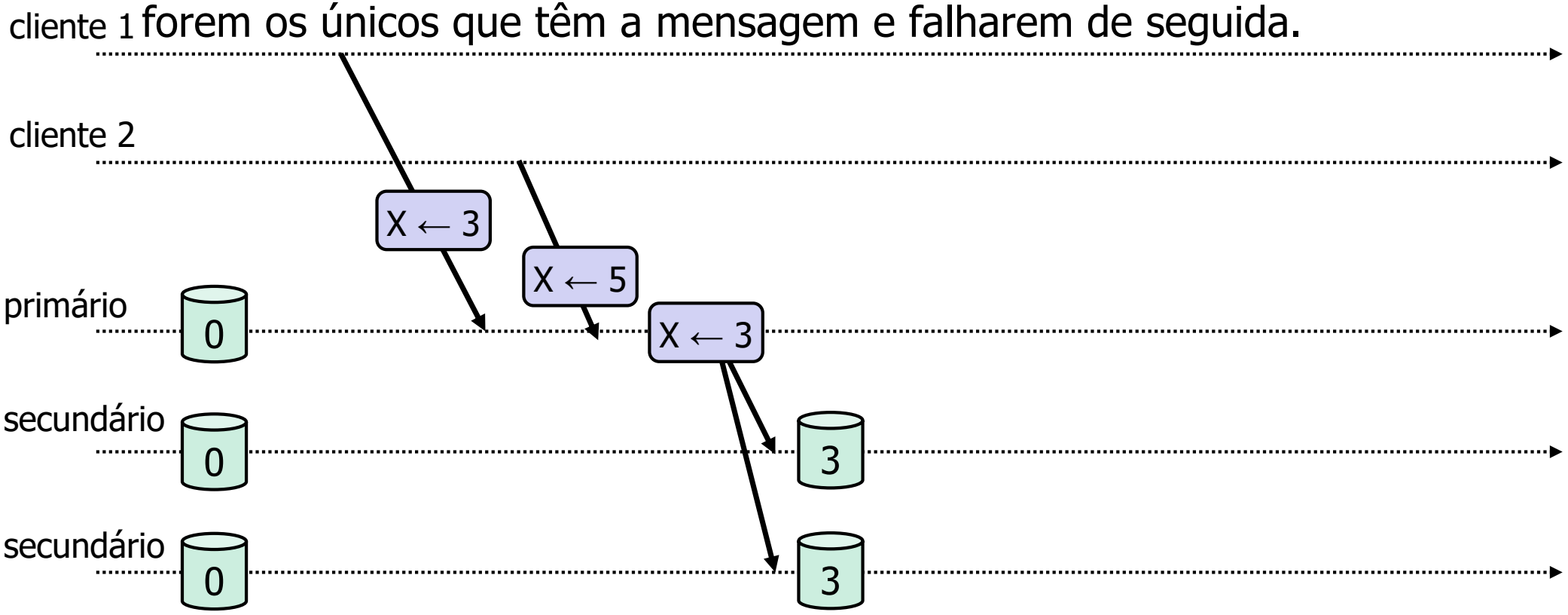
1. **Envio da operação para os secundários**



PRIMÁRIO / SECUNDÁRIO : ESCRITA : EXECUÇÃO

1. **Envio da operação para os secundários**
2. **Secundário executa operação se f=1**

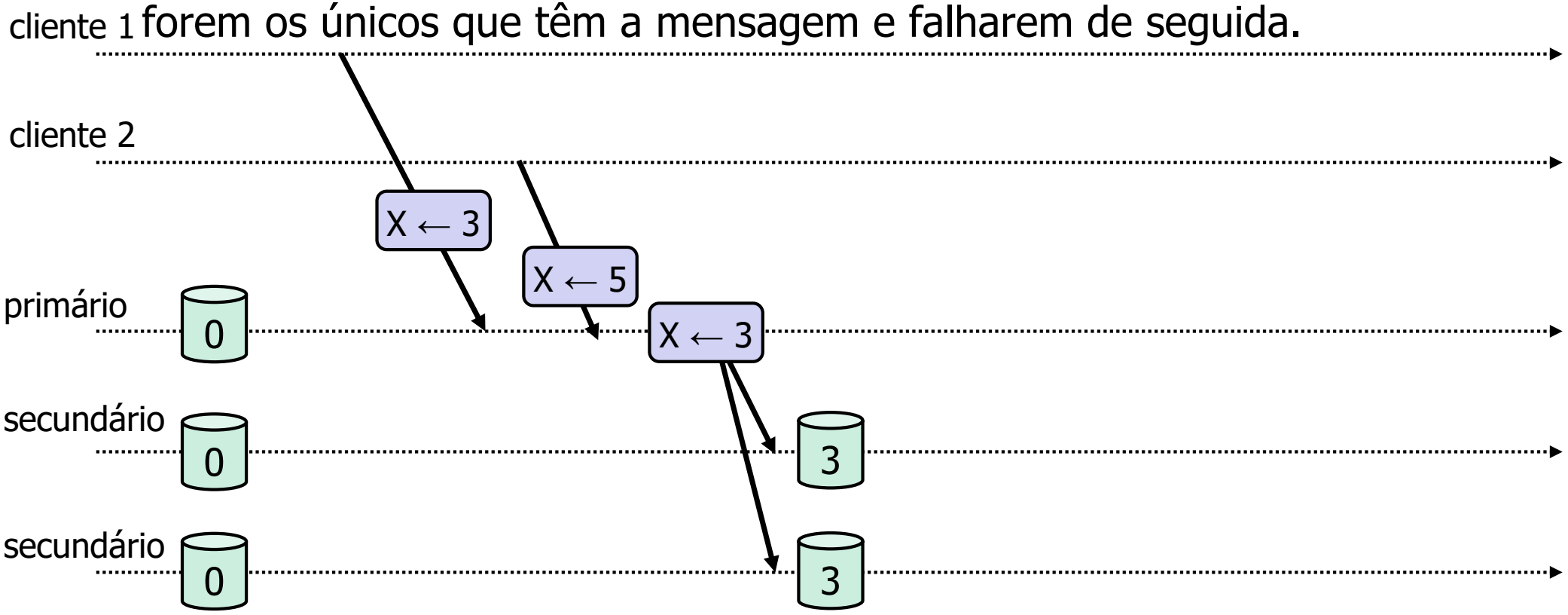
NOTA: Com f > 1, um secundário não pode excutar imediatamente a operação, porque esta pode-se perder se o primário e esse secundário forem os únicos que têm a mensagem e falharem de seguida.



PRIMÁRIO / SECUNDÁRIO : ESCRITA : EXECUÇÃO

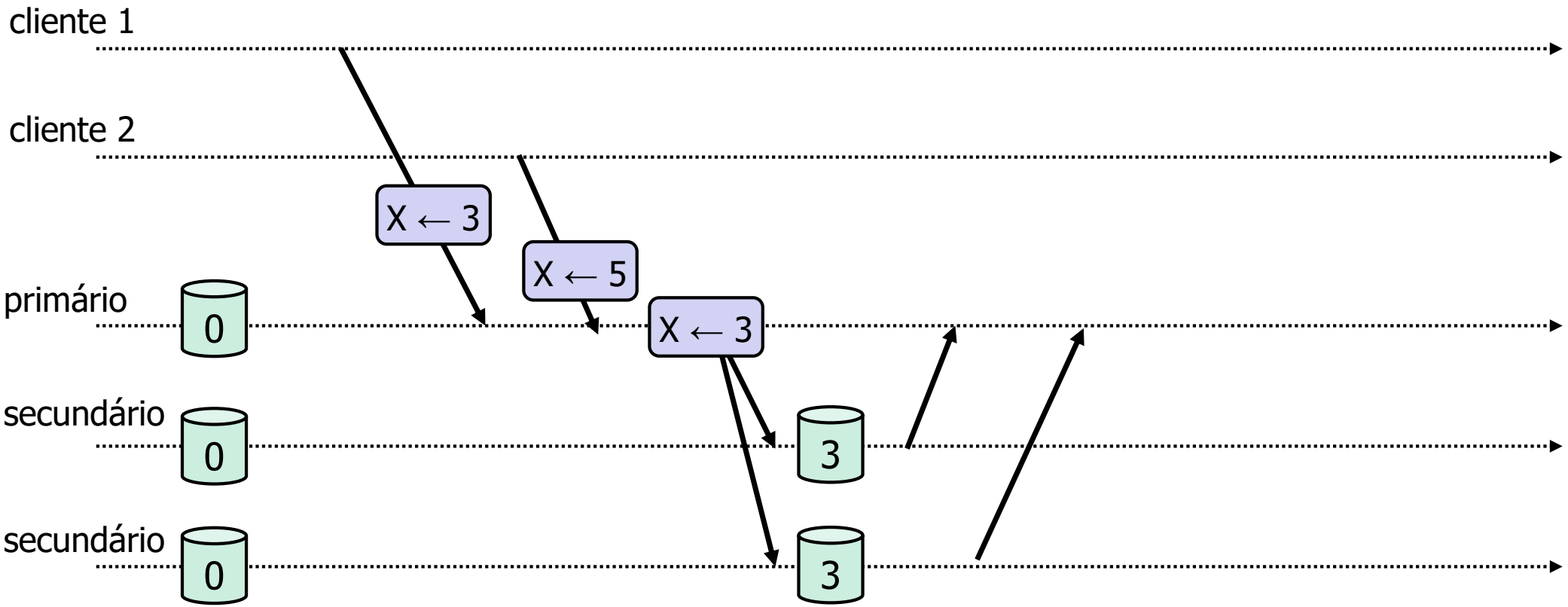
1. **Envio da operação para os secundários**
2. **Secundário executa operação se f=1**

NOTA: Com f > 1, um secundário não pode excutar imediatamente a operação, porque esta pode-se perder se o primário e esse secundário forem os únicos que têm a mensagem e falharem de seguida.



PRIMÁRIO / SECUNDÁRIO : ESCRITA : EXECUÇÃO

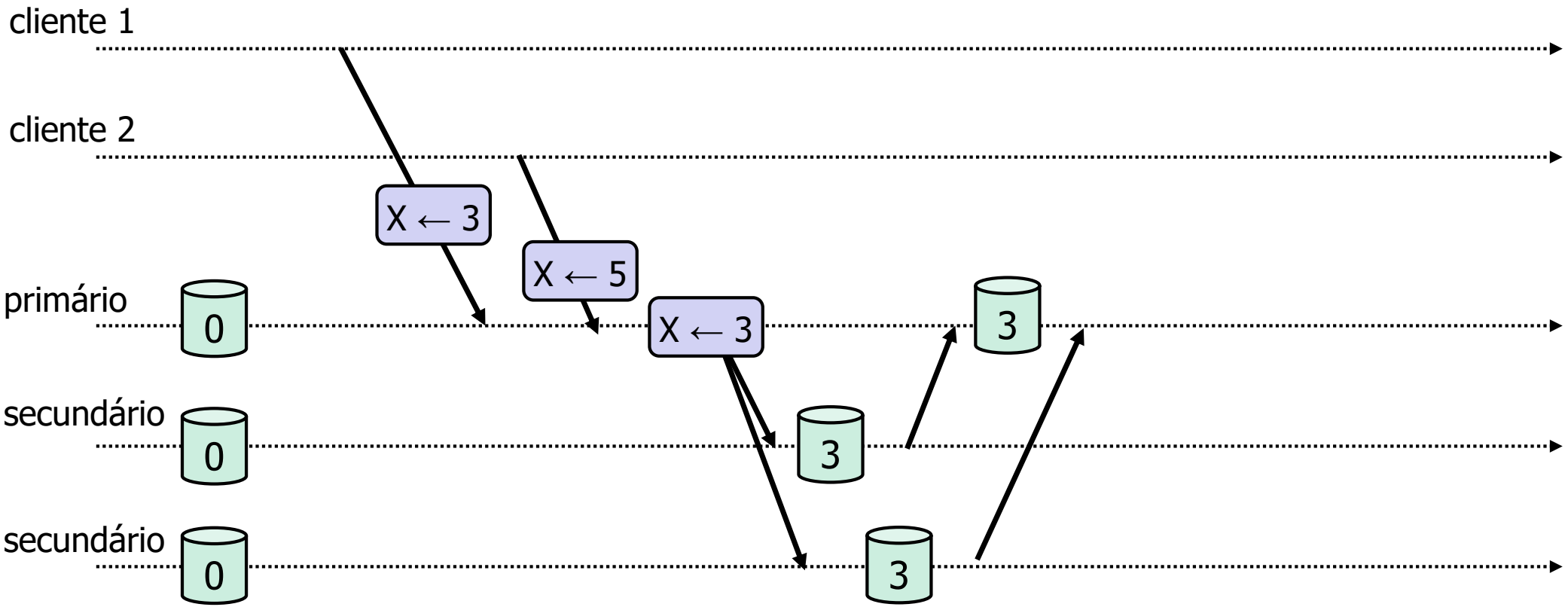
1. **Envio da operação para os secundários**
2. **Secundário executa operação se f=1**
3. **Secundário envia ACK para primário**



PRIMÁRIO / SECUNDÁRIO : ESCRITA : EXECUÇÃO

Quando recebe f acks (com f o número de falhas a tolerar)

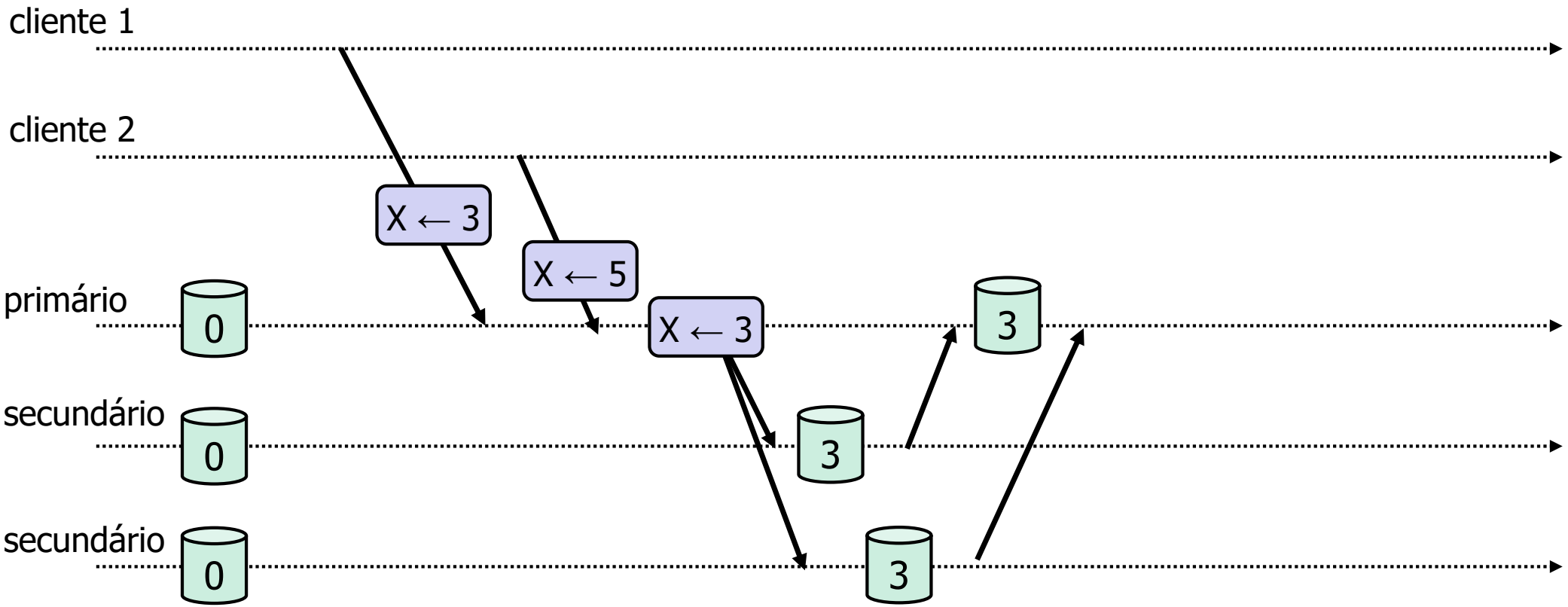
4. **Execução no primário**



PRIMÁRIO / SECUNDÁRIO : ESCRITA : EXECUÇÃO

Quando recebe f acks (com f o número de falhas a tolerar)

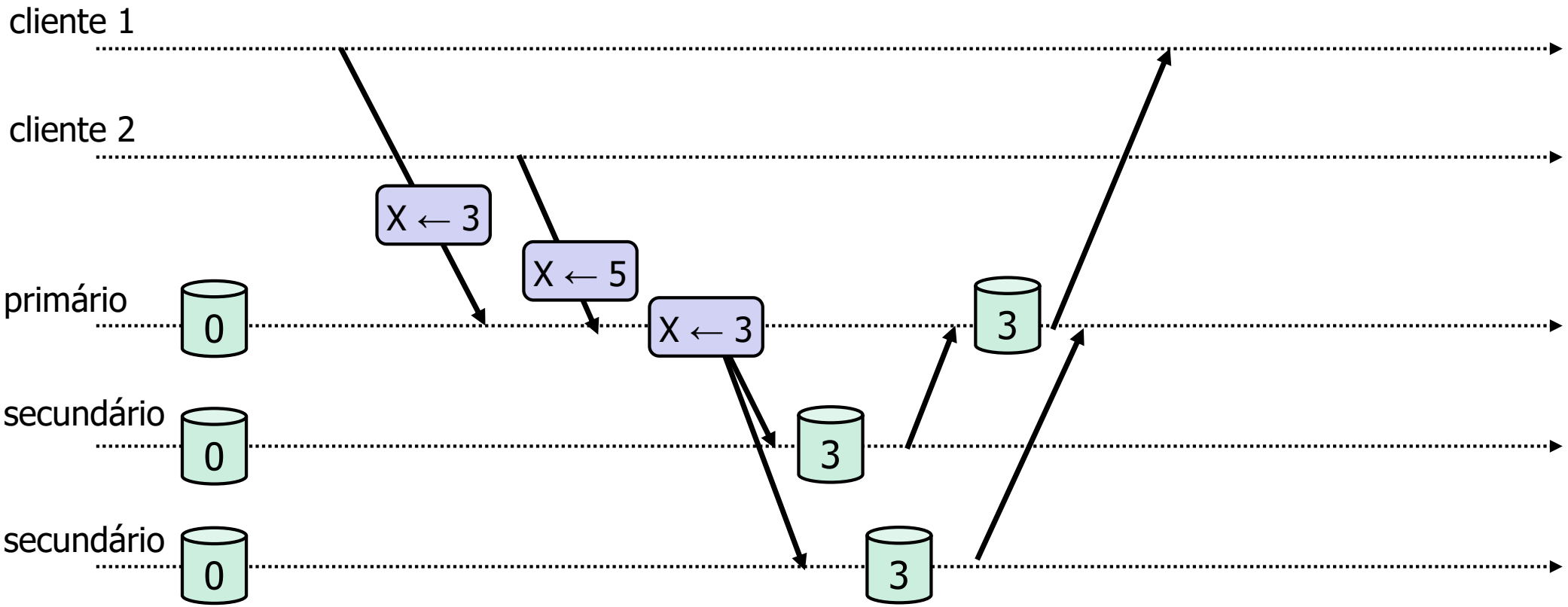
4. **Execução no primário**



PRIMÁRIO / SECUNDÁRIO : ESCRITA : EXECUÇÃO

Quando recebe f acks (com f o número de falhas a tolerar)

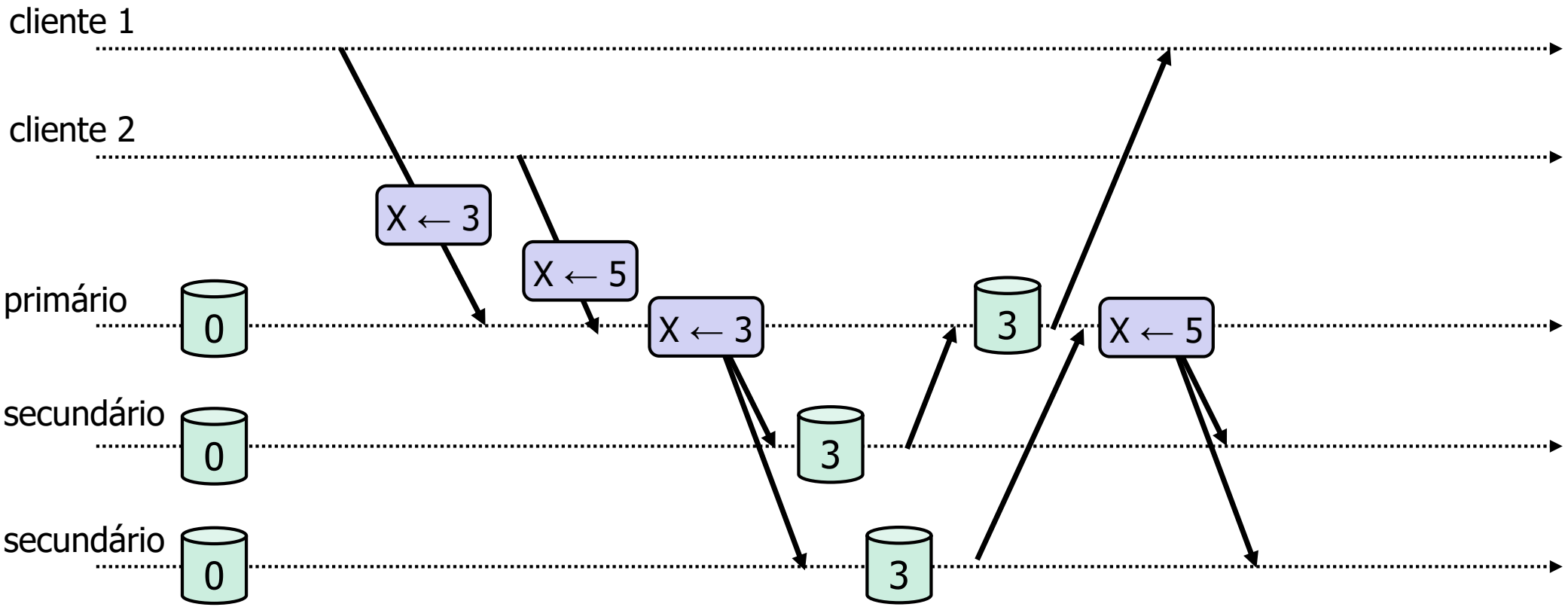
4. **Execução no primário**
5. **Primário envia ACK para cliente**



PRIMÁRIO / SECUNDÁRIO : ESCRITA : EXECUÇÃO

Quando recebe f acks (com f o número de falhas a tolerar)

4. **Execução no primário**
5. **Primário envia ACK para cliente**



PRIMEIRA ABORDAGEM: CONSISTÊNCIA FORTE

Objetivo: ter várias réplicas, mas que se comportem como se existisse apenas uma réplica que tolerasse falhas

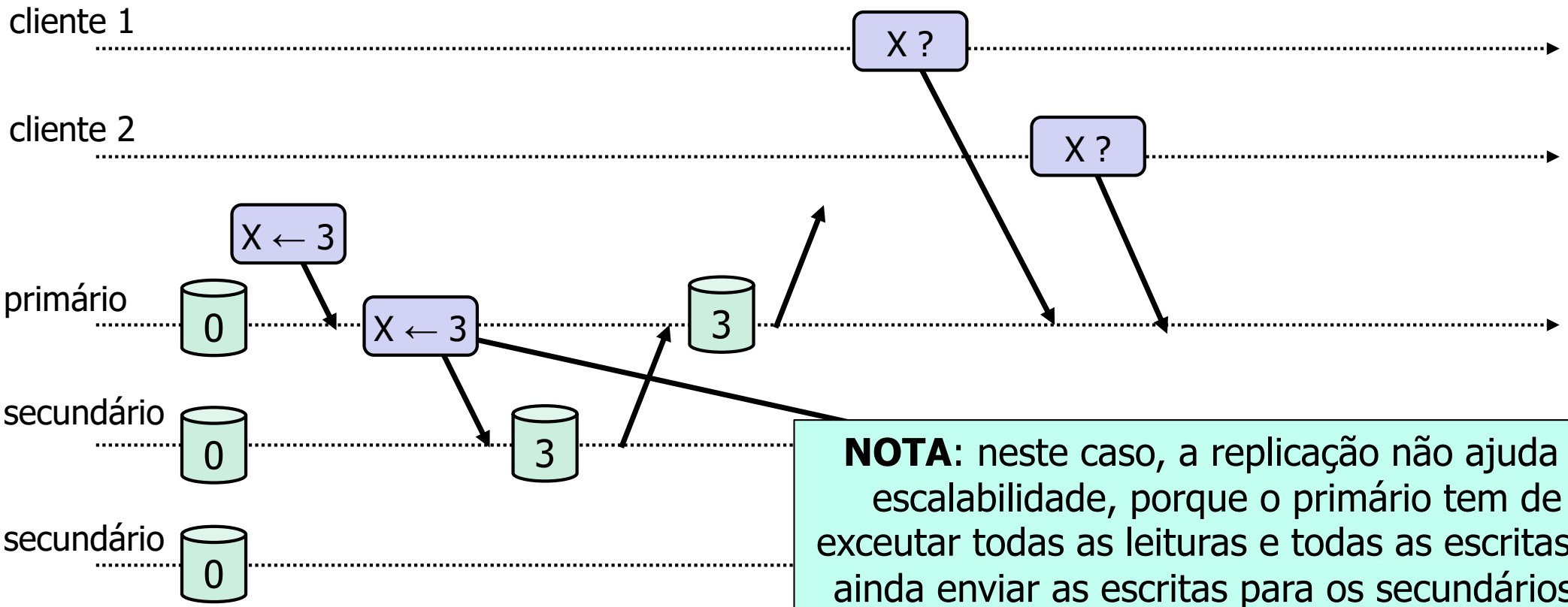
Desafios:

1. Garantir que todas as réplicas convergem para o mesmo estado
2. **Garantir que os clientes observam sempre a última escrita**

PRIMÁRIO / SECUNDÁRIO : LEITURA

Para se garantir que se observa a última escrita, os clientes têm de fazer as leituras no primário. **Porquê?**

Porque um secondário pode não ter recebido a última escrita, se a propagação do primário para o secundário falhou.



NOTA: neste caso, a replicação não ajuda a escalabilidade, porque o primário tem de exceutar todas as leituras e todas as escritas (e ainda enviar as escritas para os secundários).

PRIMEIRA ABORDAGEM: CONSISTÊNCIA FORTE

Objetivo: ter várias réplicas, mas que se comportem como se existisse apenas uma réplica que tolerasse falhas

Desafios:

1. Garantir que todas as réplicas convergem para o mesmo estado
2. **Garantir que os clientes observam sempre a última escrita**

PRIMEIRA ABORDAGEM: CONSISTÊNCIA FORTE

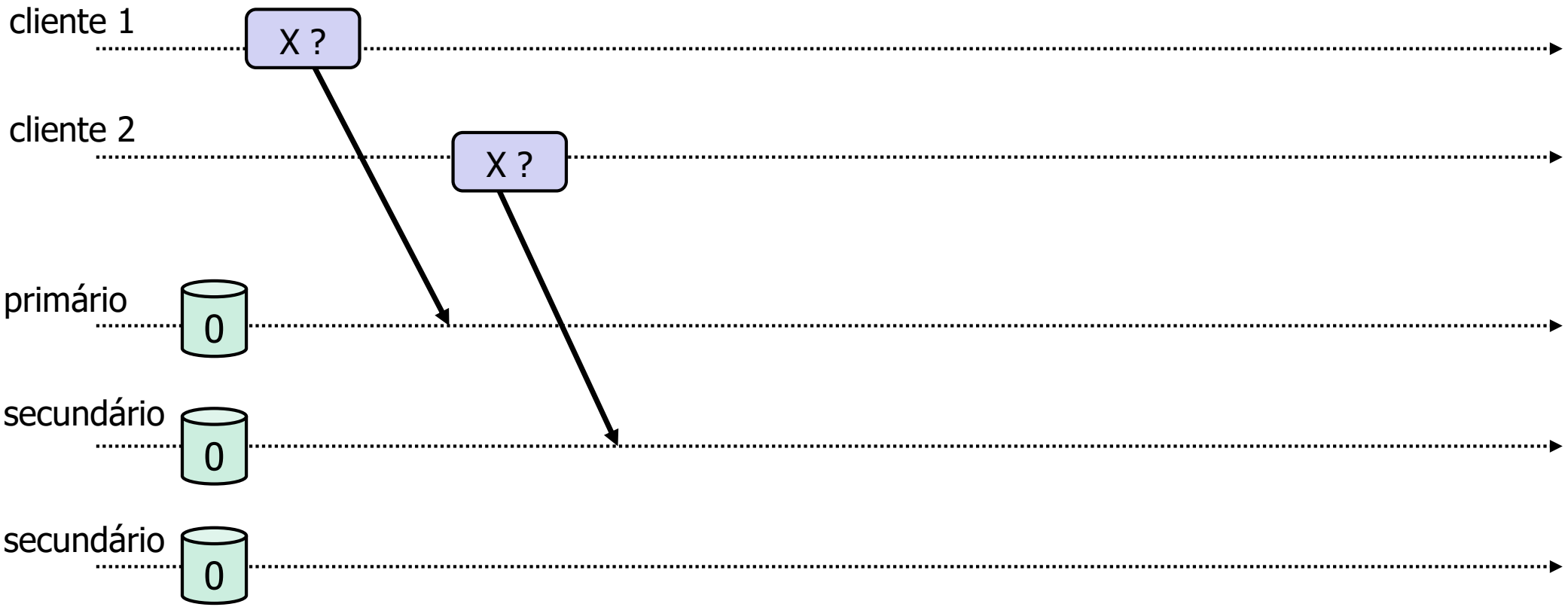
Objetivo: ter várias réplicas, mas que se comportem como se existisse apenas uma réplica que tolerasse falhas

Desafios:

1. Garantir que todas as réplicas convergem para o mesmo estado
2. ~~**Garantir que os clientes observam sempre a última escrita**~~
2. **Garantir que os clientes observam sempre o resultados das suas escritas e que nunca observam um valor mais antigo do que o que observaram anteriormente.**

PRIMÁRIO / SECUNDÁRIO : LEITURA

Leituras são enviadas para qualquer uma das réplicas

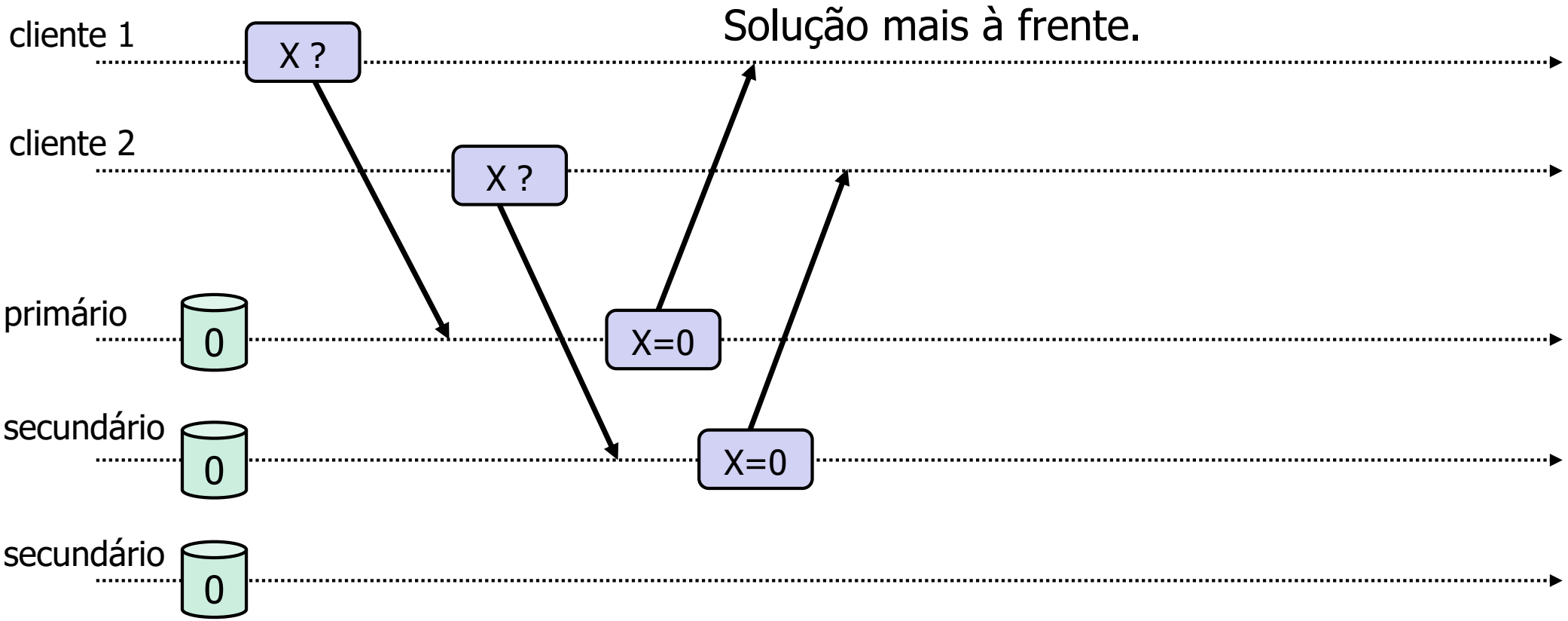


PRIMÁRIO / SECUNDÁRIO : LEITURA

Leituras são enviadas para qualquer uma das replicas

Secundário executa a operação de leitura e devolve o resultado

Com garantir que os clientes observam sempre o resultados das suas escritas e que nunca observam um valor mais antigo do que o que observaram anteriormente ?



PRIMÁRIO / SECUNDÁRIO : FALHAS

Modelo de falhas:

- as mensagens podem-se perder e trocar de ordem;
- um servidor pode falhar por crash, i.e., deixar de responder (e eventualmente recuperar).

PRIMÁRIO / SECUNDÁRIO : FALHAS

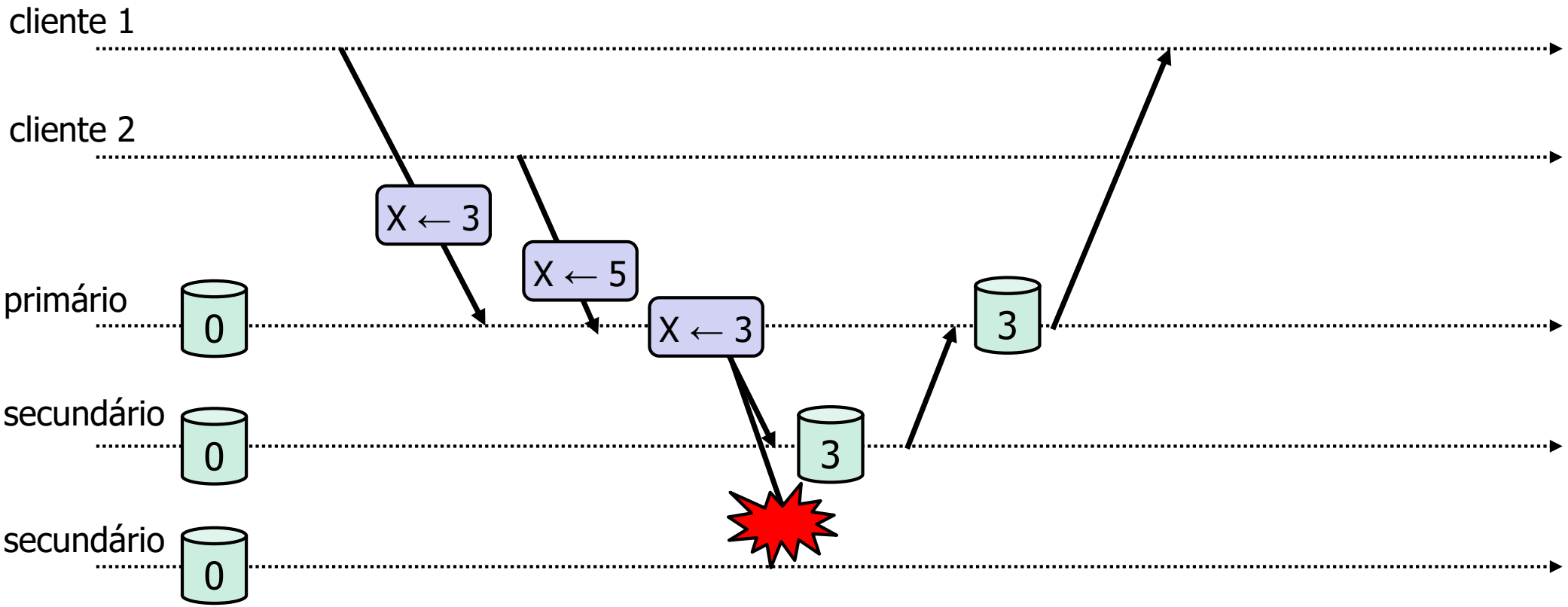
Modelo de falhas:

- **as mensagens podem-se perder e trocar de ordem;**
- um servidor pode falhar por crash, i.e., deixar de responder (e eventualmente recuperar).

PRIMÁRIO / SECUNDÁRIO : FALHA NAS MENSAGENS PARA UM SECUNDÁRIO

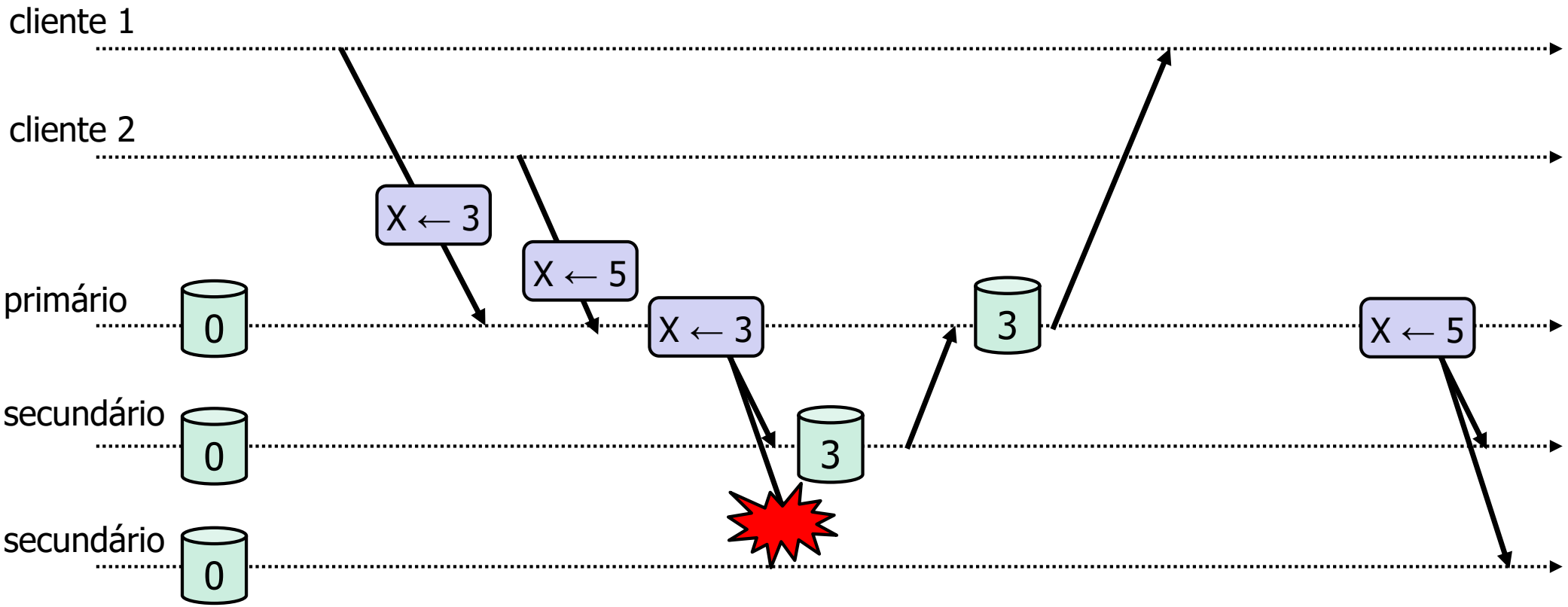
O que fazer quando uma mensagem para um secundário se perde?

O primário pode (e TEM de) devolver ACK ao cliente sem receber a resposta de todos os secundários. NOTA: isto é necessário porque o secundário pode ter falhado.



PRIMÁRIO / SECUNDÁRIO : FALHA NAS MENSAGENS PARA UM SECUNDÁRIO

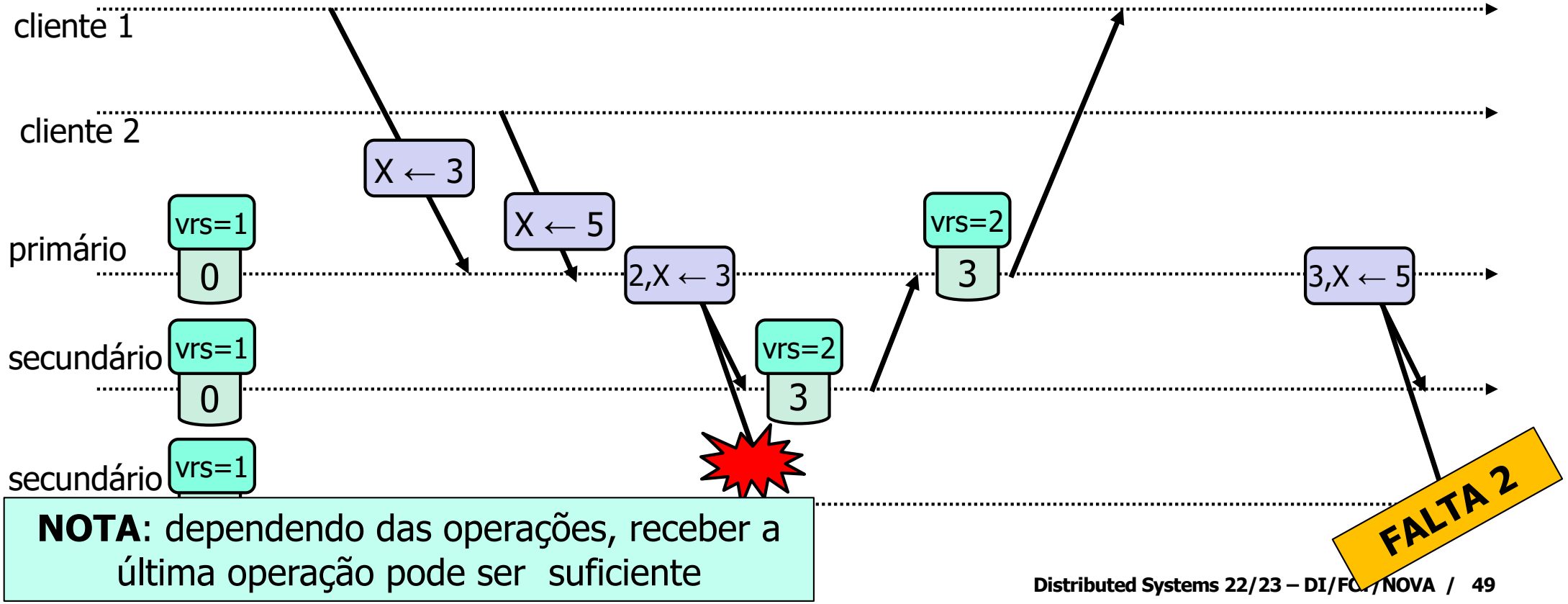
Como é que um secundário sabe que está a receber todas as operações?



PRIMÁRIO / SECUNDÁRIO : FALHA NAS MENSAGENS PARA UM SECUNDÁRIO

Como é que um secundário sabe que está a receber todas as operações?

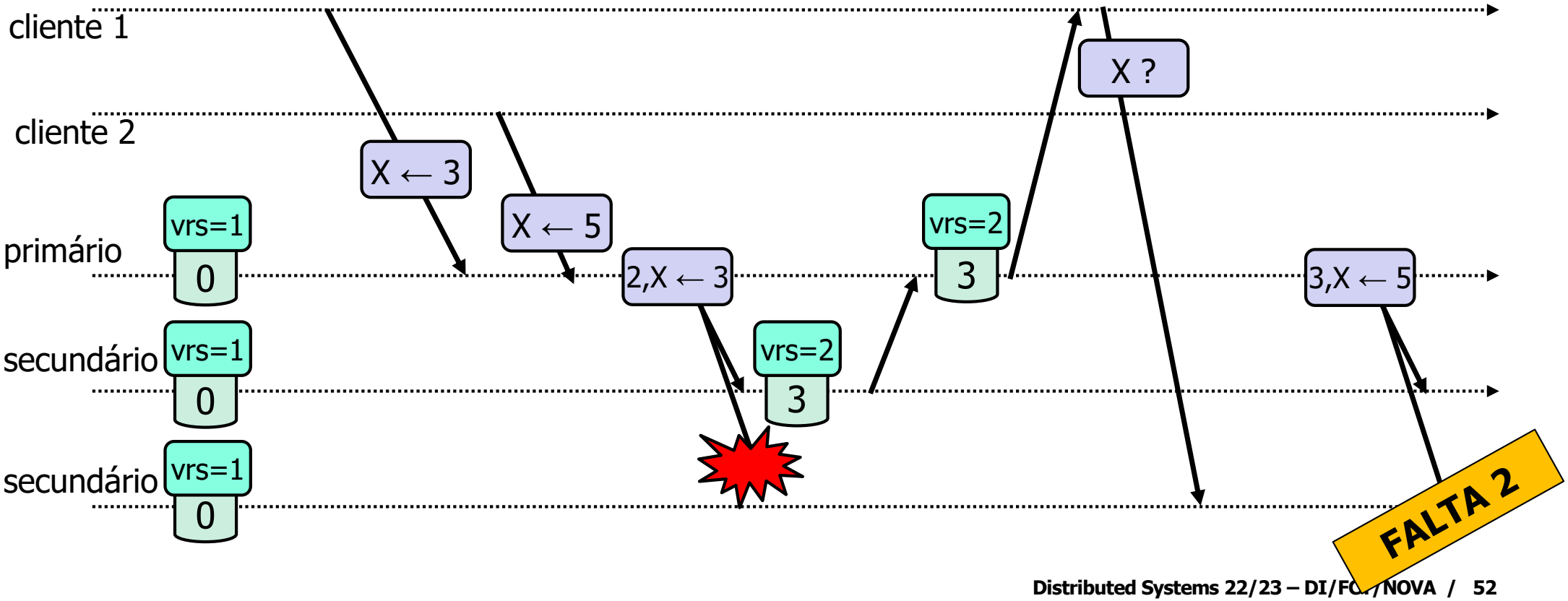
Operações devem ser numeradas pelo primário (canal FIFO) – este número pode ser usado para identificar a versão do estado.
Secundário deteta falta de operação na próxima operação (ou por contacto periódico com primário).



NOTA: dependendo das operações, receber a última operação pode ser suficiente

PRIMÁRIO / SECUNDÁRIO : FALHA NAS MENSAGENS PARA UM SECUNDÁRIO

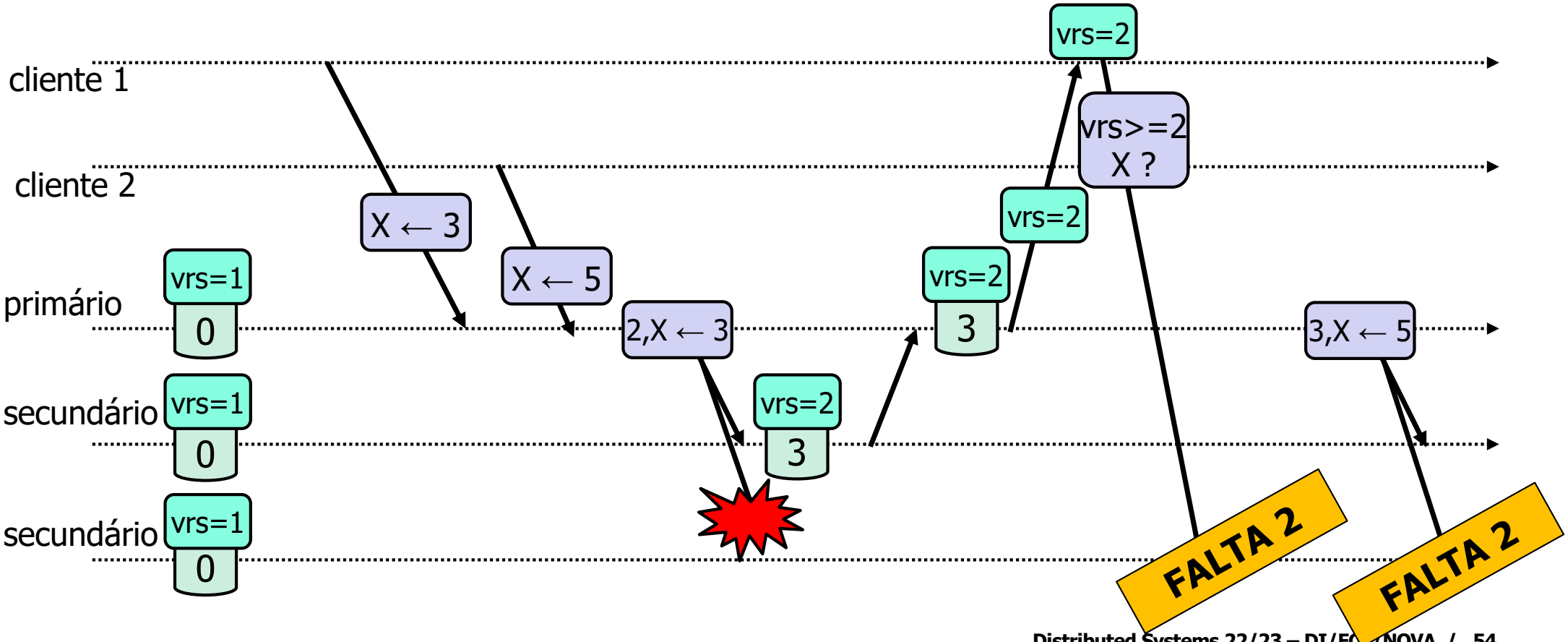
Se o clientes lerem do secundário, **como é que se garante que cliente não lê versão anterior à sua escrita?**



PRIMÁRIO / SECUNDÁRIO : FALHA NAS MENSAGENS PARA UM SECUNDÁRIO

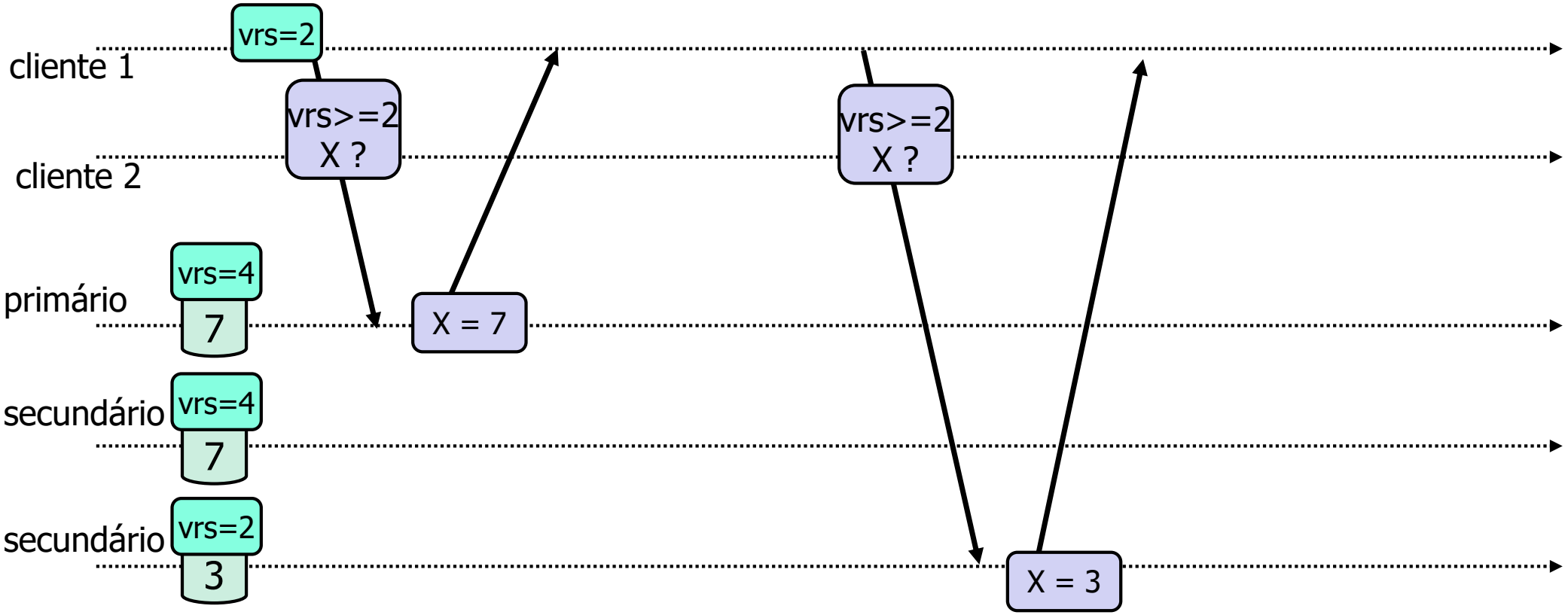
Se o clientes lerem do secundário, **como é que se garante que cliente não lê versão anterior à sua escrita?**

Cliente pode receber versão escrita e enviar essa versão no próximo pedido; apenas um servidor com uma versão igual ou superior responderá.



PRIMÁRIO / SECUNDÁRIO : FALHA NAS MENSAGENS PARA UM SECUNDÁRIO

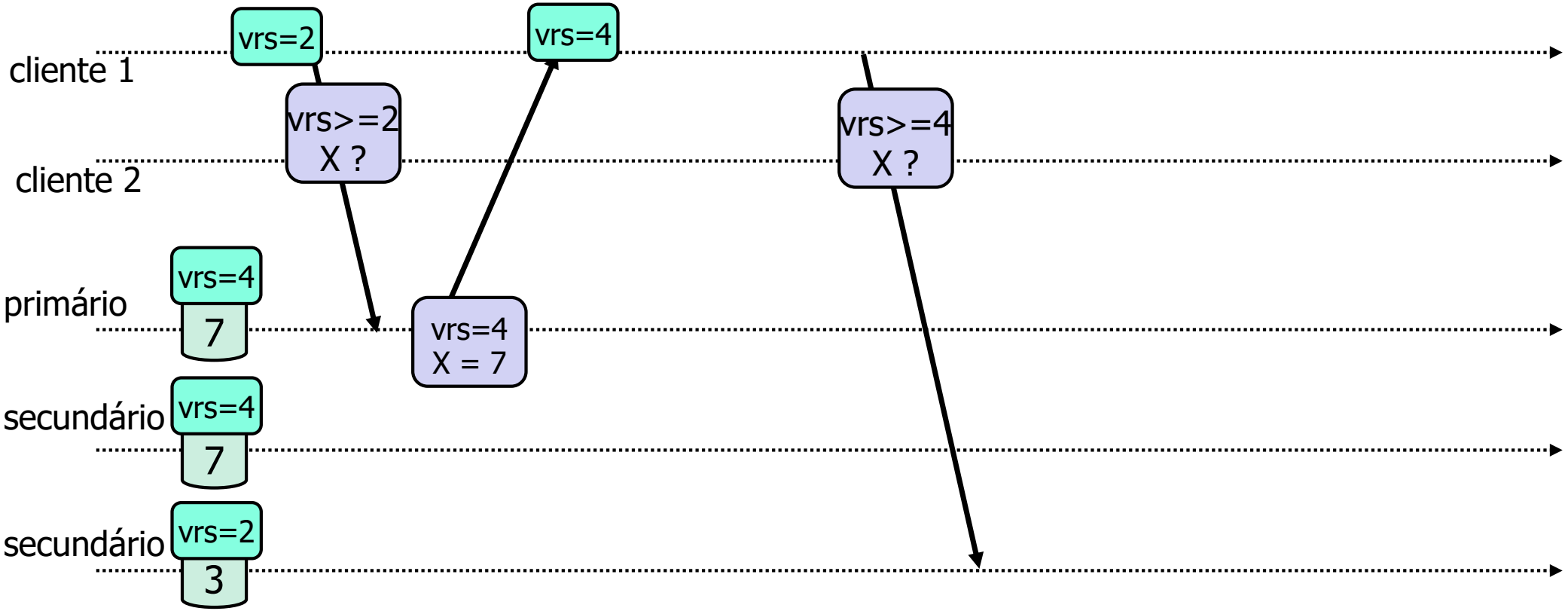
Se o clientes lerem do secundário, **como é que se garante que cliente não lê versão anterior à da última leitura?**



PRIMÁRIO / SECUNDÁRIO : FALHA NAS MENSAGENS PARA UM SECUNDÁRIO

Se o clientes lerem do secundário, **como é que se garante que cliente não lê versão anterior à da última leitura?**

Cliente pode receber versão lida e enviar essa versão no próximo pedido; apenas um servidor com uma versão igual ou superior responderá.



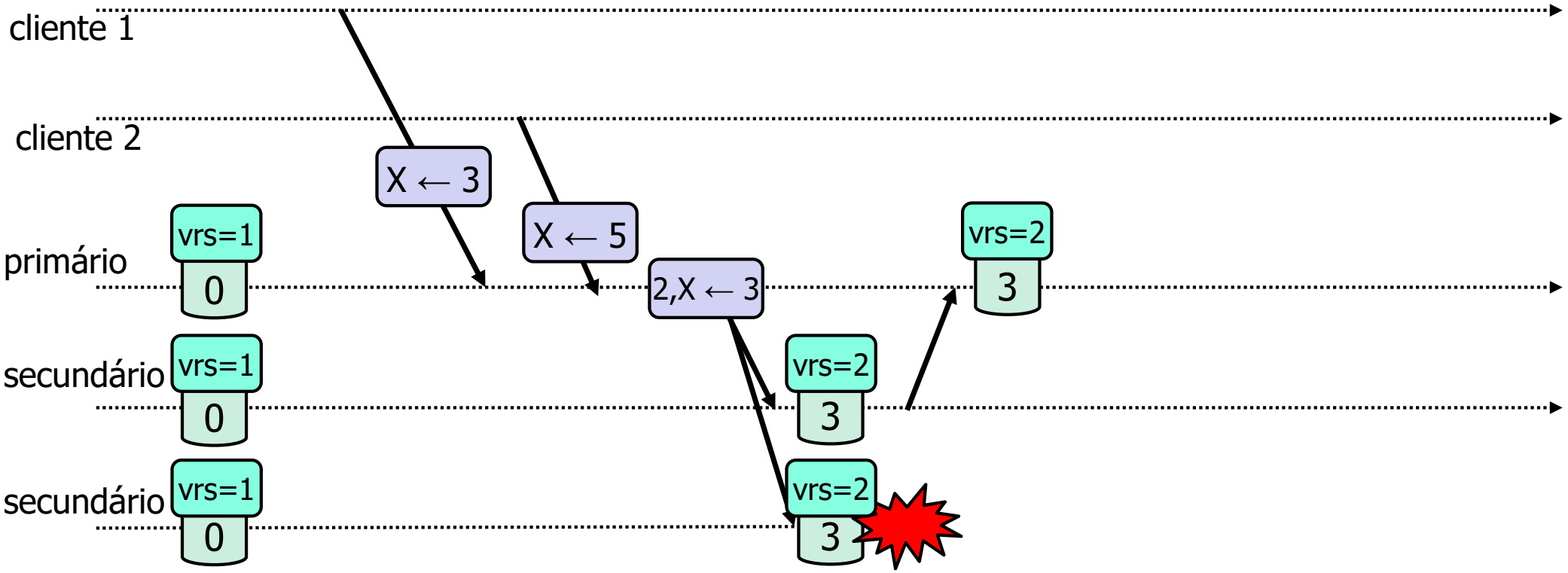
PRIMÁRIO / SECUNDÁRIO : FALHAS

Modelo de falhas:

- as mensagens podem-se perder e trocar de ordem;
- **um servidor pode falhar por crash, i.e., deixar de responder (e eventualmente recuperar).**

PRIMÁRIO / SECUNDÁRIO : FALHA NUM SECUNDÁRIO

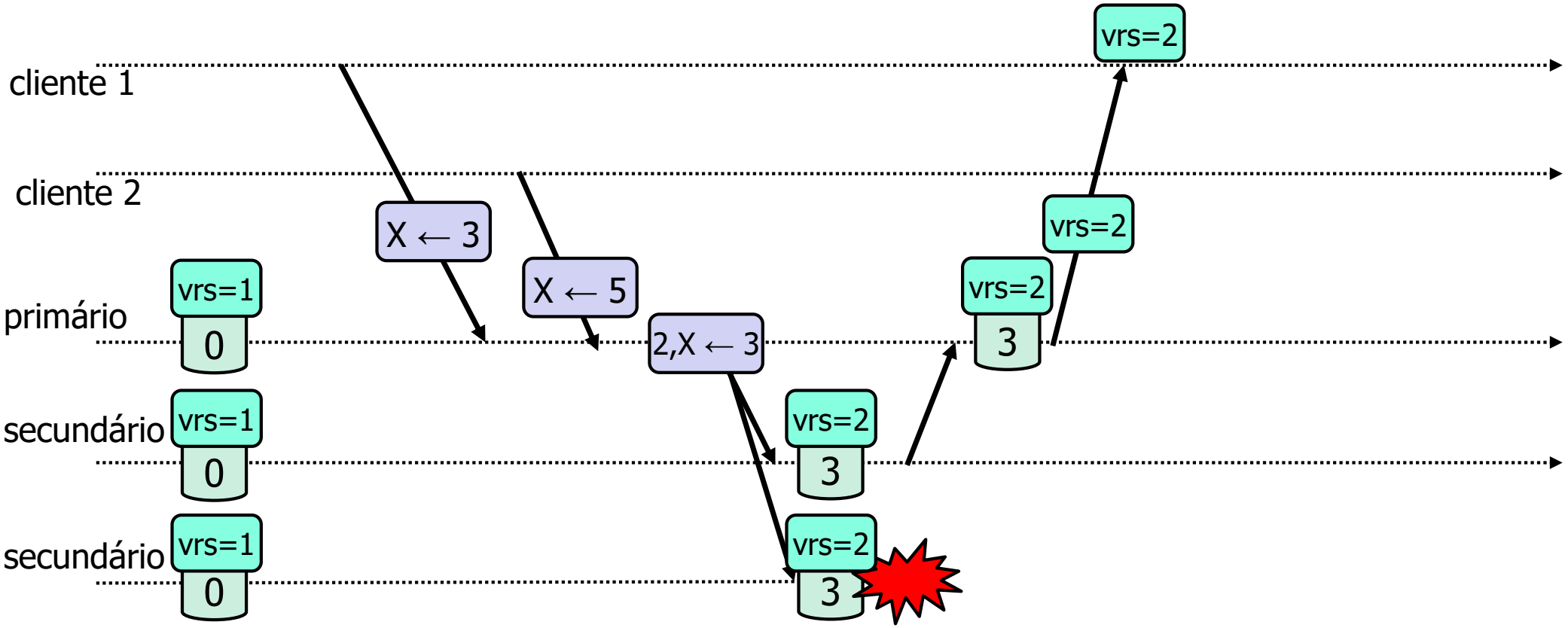
Como se lida com a falha num secundário?



PRIMÁRIO / SECUNDÁRIO : FALHA NUM SECUNDÁRIO

Como se lida com a falha num secundário?

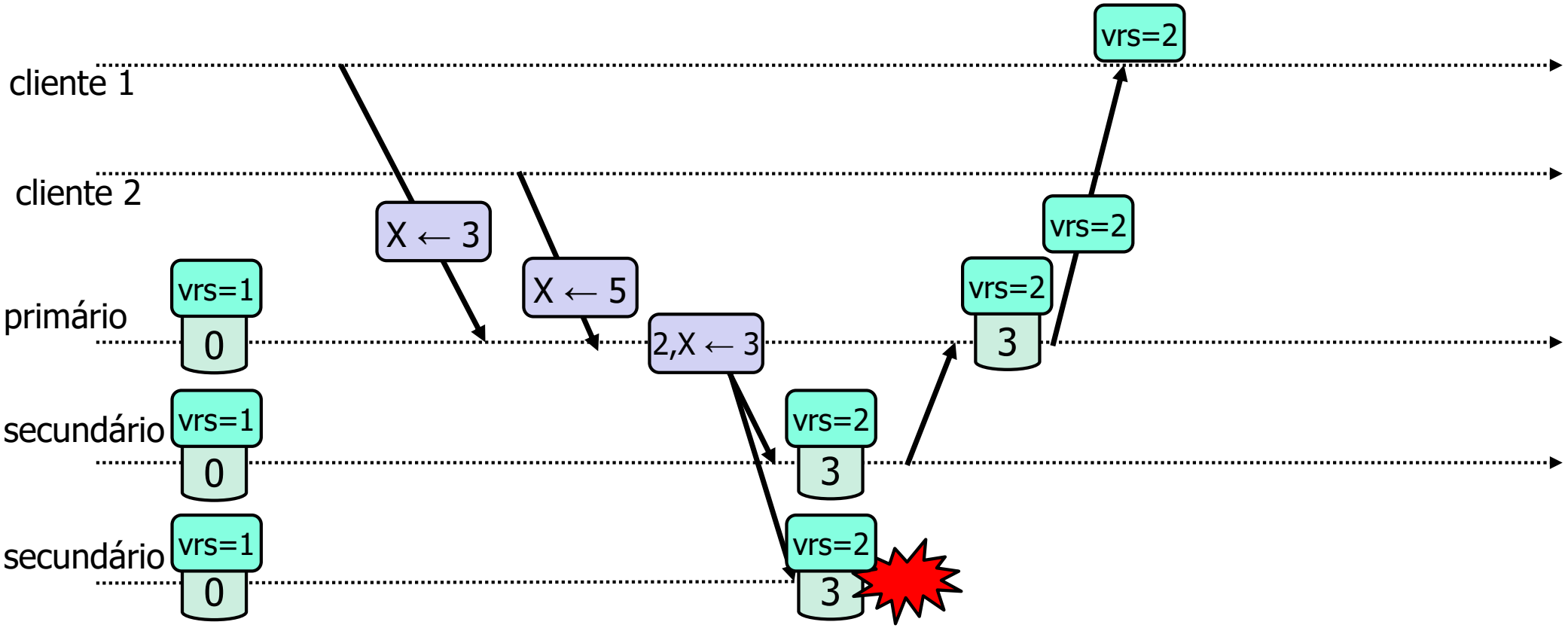
Da mesma forma que com as falhas de comunicação com o secundário. O primário e os outros secundários continuam a executar.



PRIMÁRIO / SECUNDÁRIO : FALHA NUM SECUNDÁRIO

Como se lida com a falha num secundário?

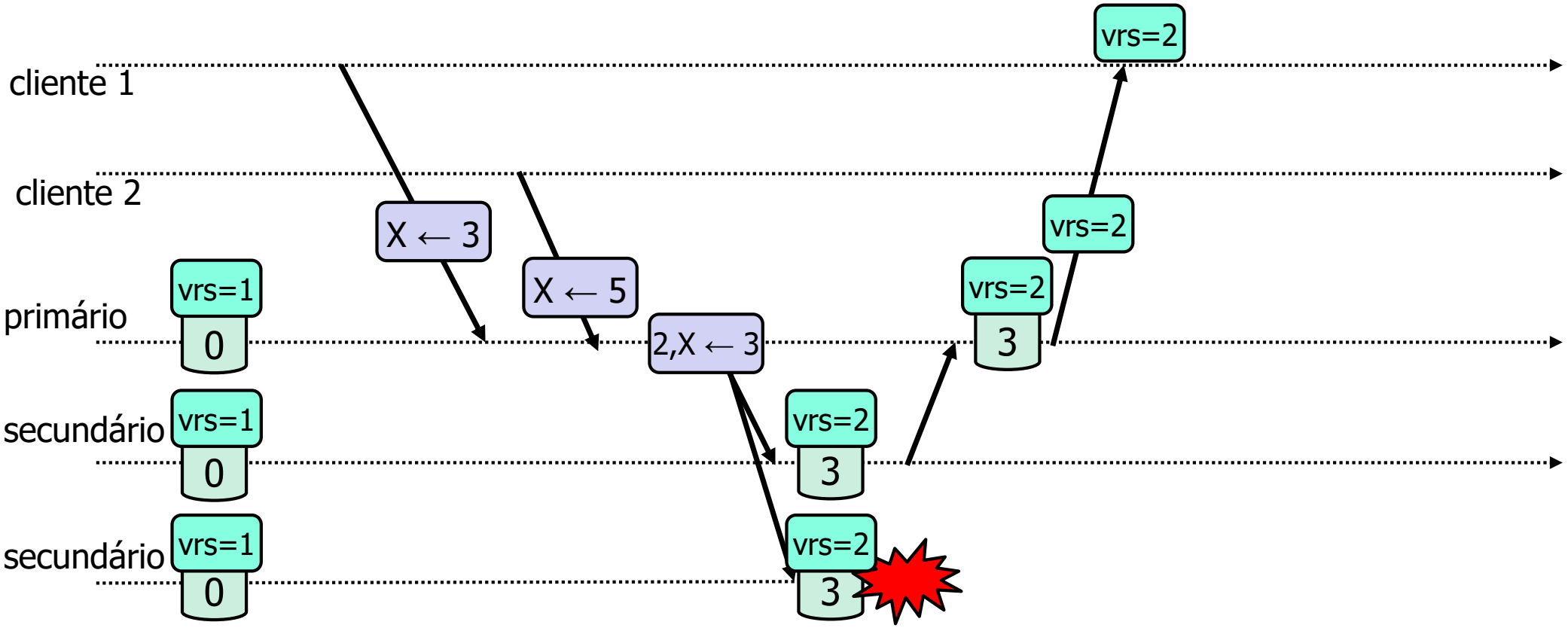
Da mesma forma que com as falhas de comunicação com o secundário. O primário e os outros secundários continuam a executar.



PRIMÁRIO / SECUNDÁRIO : FALHA NUM SECUNDÁRIO

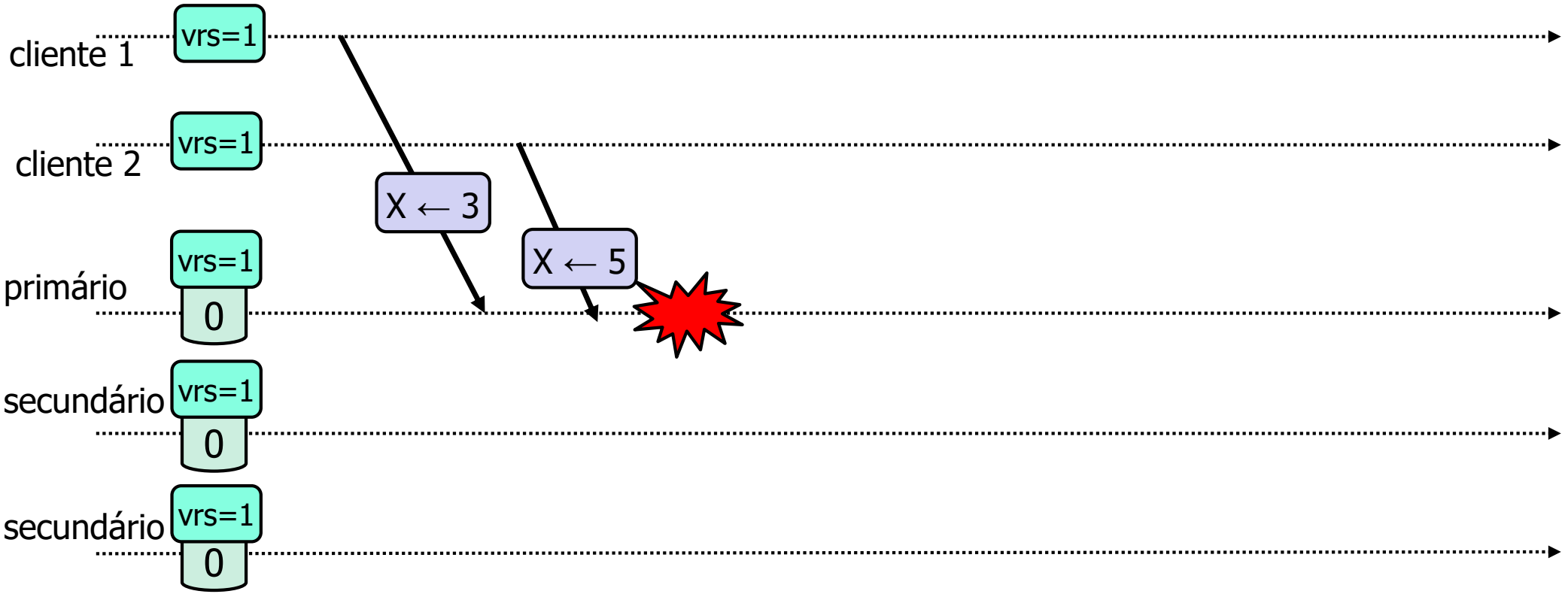
Quantos secundários precisam de responder de forma a que uma escrita não se perca?

f secundários para tolerar *f* falhas.



PRIMÁRIO / SECUNDÁRIO : FALHA NO PRIMÁRIO

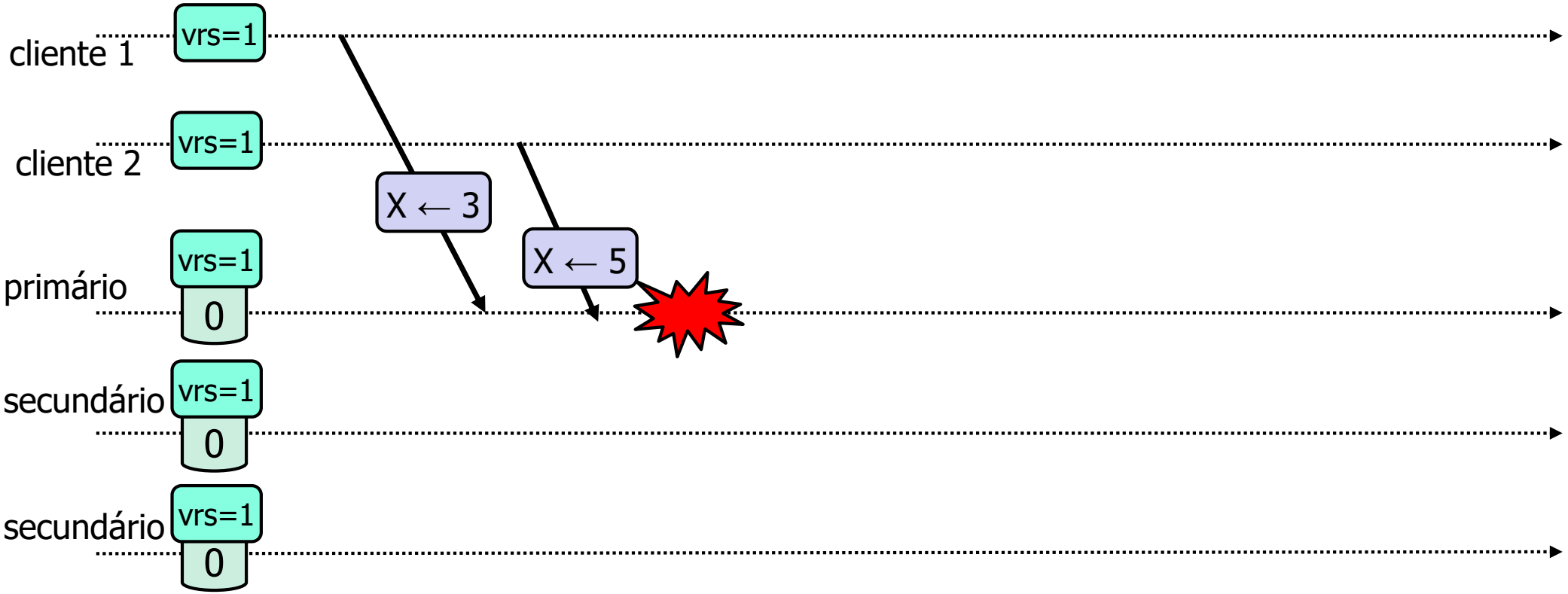
O que fazer quando o primário falha?



PRIMÁRIO / SECUNDÁRIO : FALHA NO PRIMÁRIO

O que fazer quando o primário falha?

HIP. 1: Parar de aceitar escritas. Neste caso não se está a tolerar 1 falha.

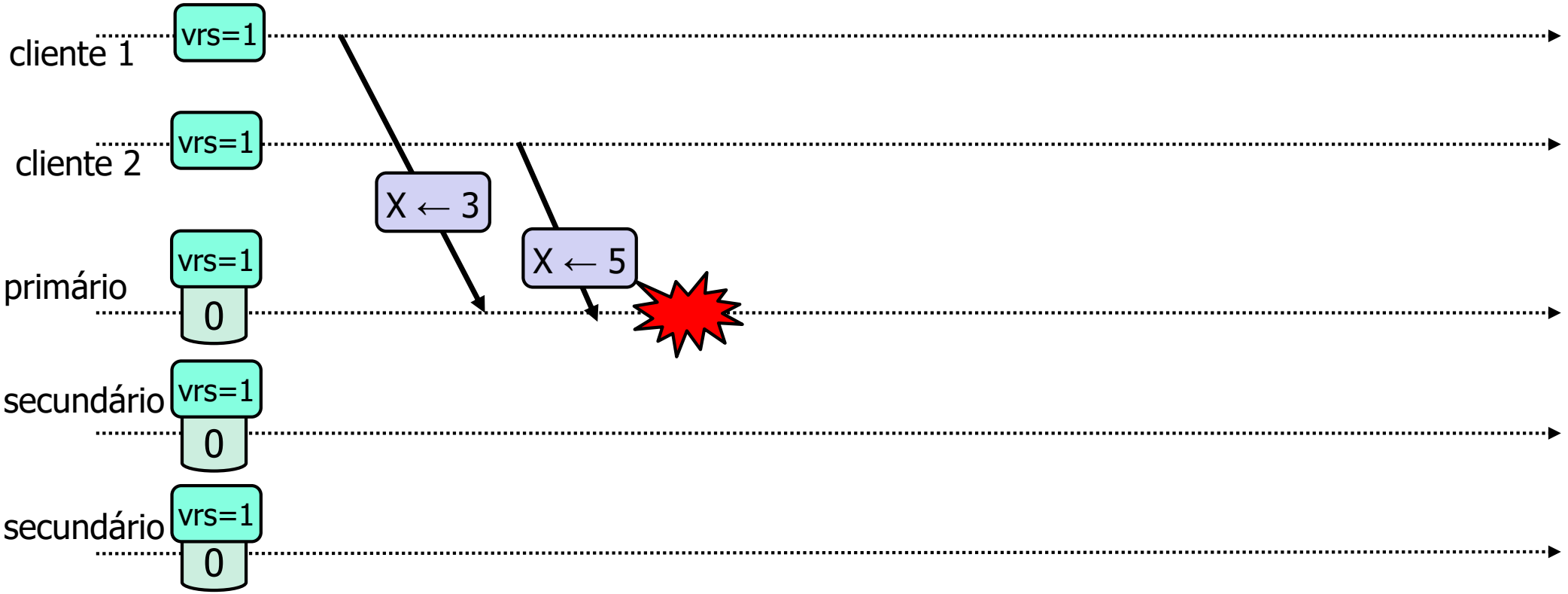


PRIMÁRIO / SECUNDÁRIO : FALHA NO PRIMÁRIO

O que fazer quando o primário falha?

Hip. 2: Continuar a executar.

1. Um secundário deve ser eleito o novo primário
2. Secundários devem concordar em qual a última operação executada (e possivelmente confirmada a um cliente)

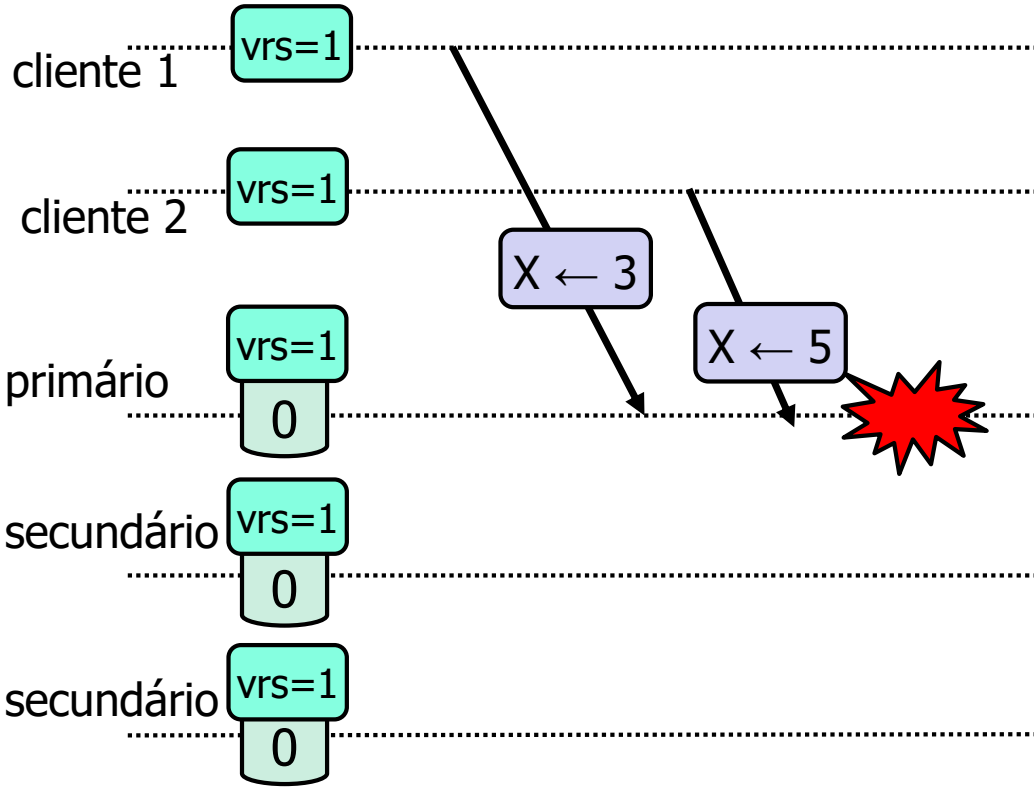


PRIMÁRIO / SECUNDÁRIO : FALHA NO PRIMÁRIO

O que fazer quando o primário falha?

Hip. 2: Continuar a executar.

1. Um secundário deve ser eleito o novo primário
2. **Secundários devem concordar em qual a última operação executada (e possivelmente confirmada a um cliente)**



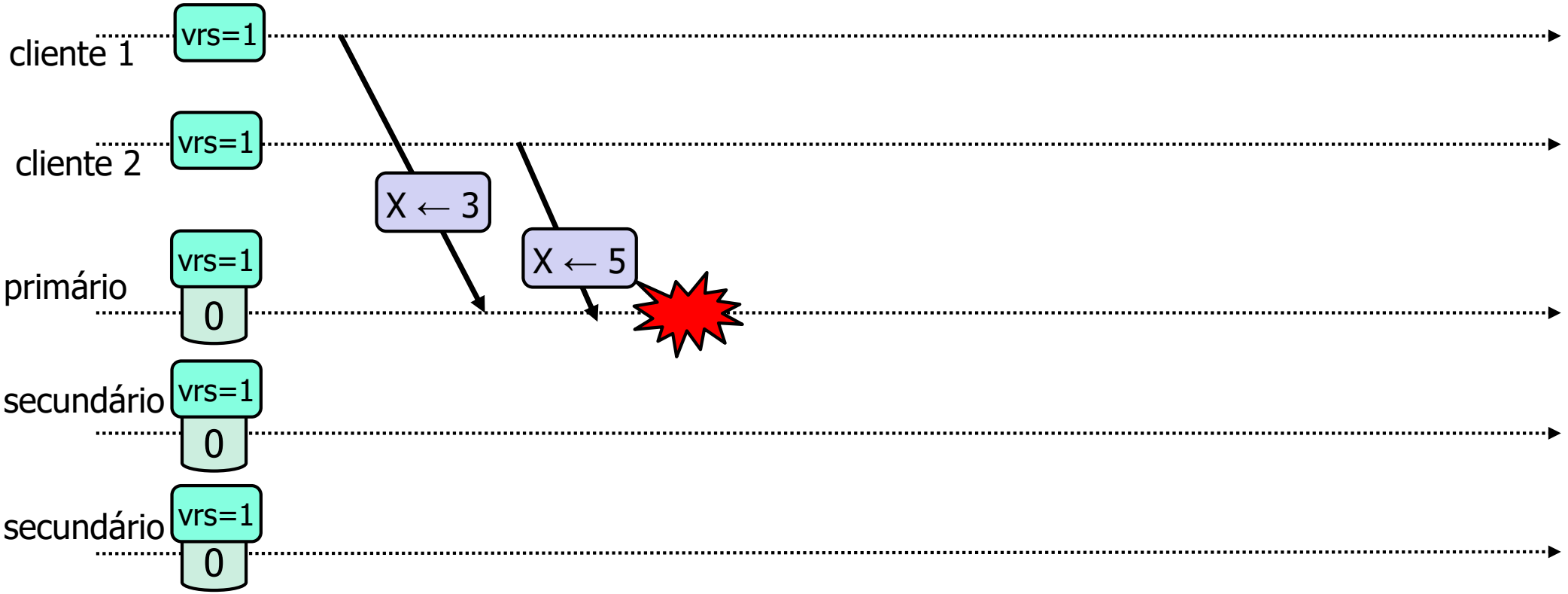
Como garantir que não se perdem operações?

Há pelo menos 1 secundário que tem a última operação confirmada a um cliente - porque só se responde quando f secundários fazem ack e só podemo ter falhado f-1 secundários.

PRIMÁRIO / SECUNDÁRIO : FALHA NO PRIMÁRIO

Como eleger um novo primário?

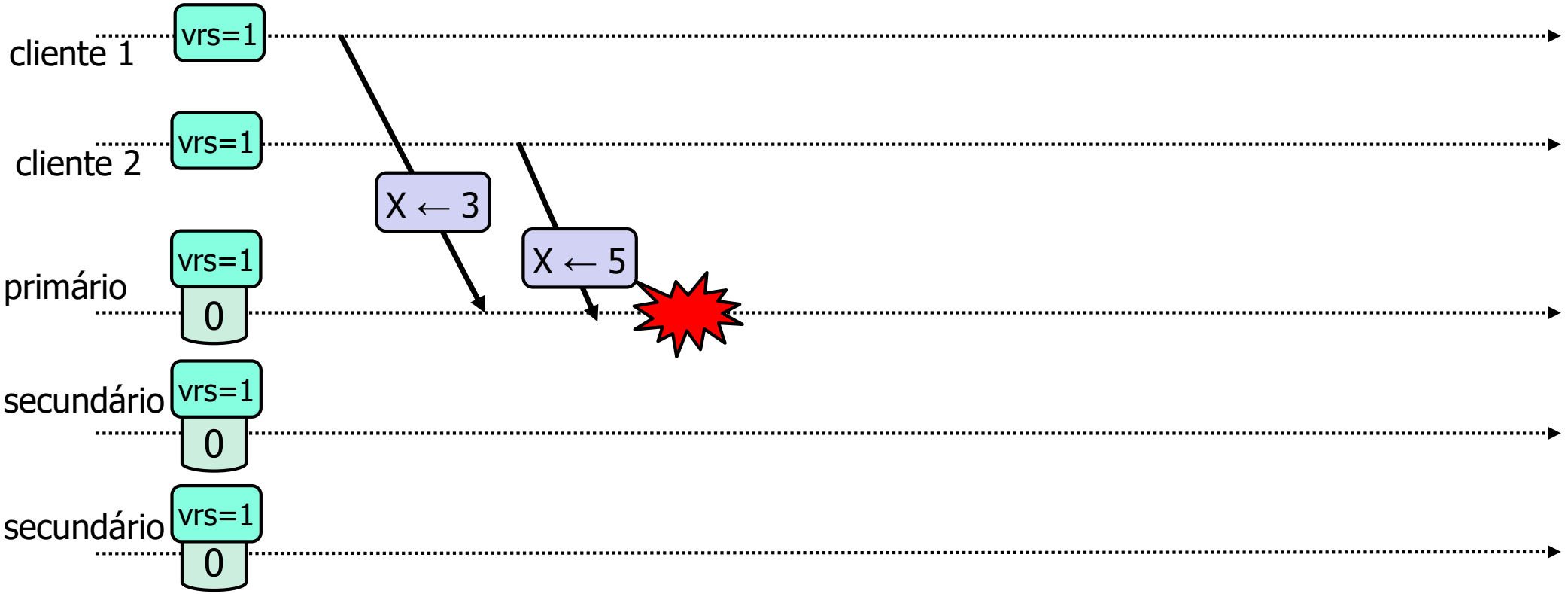
- Algoritmo de eleição, sistema de comunicação em grupo, etc.
- [algoritmos estudados em ASD]



PRIMÁRIO / SECUNDÁRIO : FALHA NO PRIMÁRIO

Como eleger um novo primário, na prática?

- Usar um sistema de coordenação (e.g. Zookeeper) para registar quem é o primário.



APACHE ZOOKEEPER

O Zookeeper é um sistema de coordenação centralizado e replicado com elevada disponibilidade.

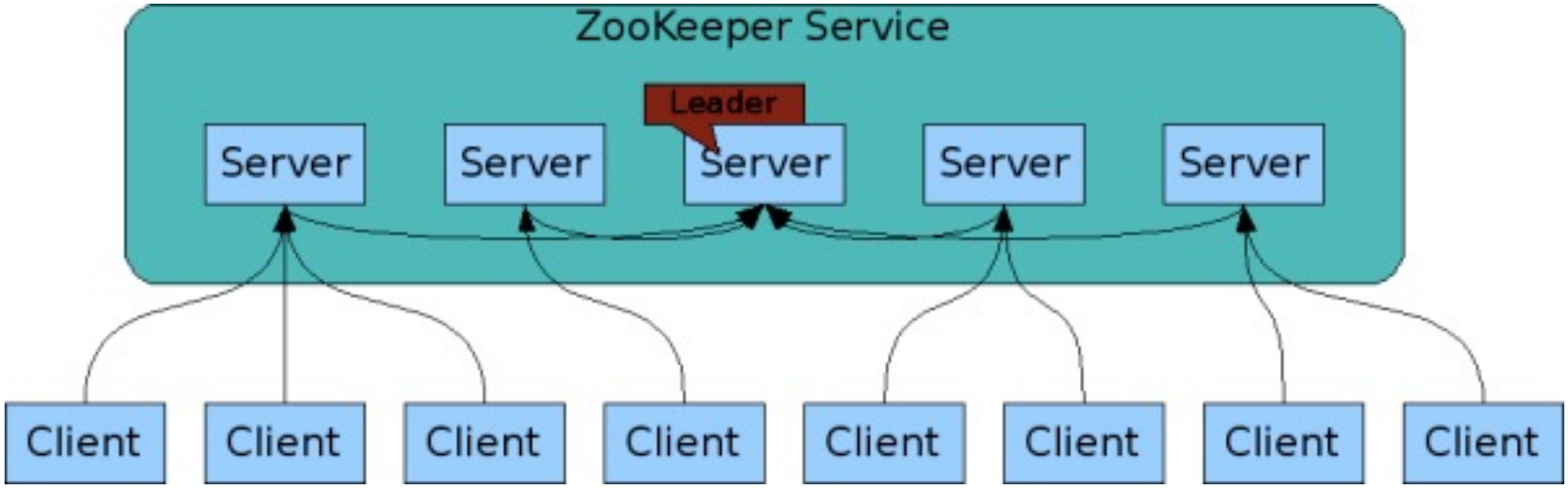
Usado para manter informação de configuração.

> *"ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services."*

ZOOKEEPER: ARQUITETURA

Conjunto de servidores mantém réplicas da "base de dados".

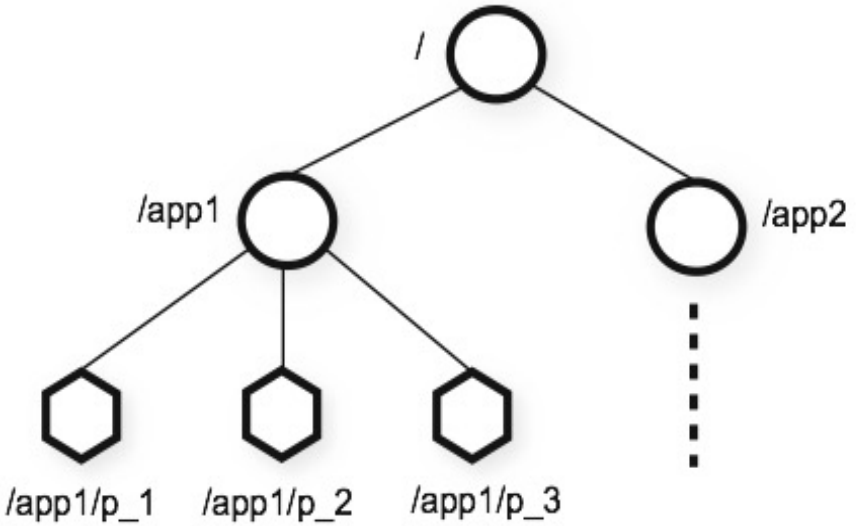
Operações totalmente ordenadas.



ZOOKEEPER: MODELO DE DADOS

Nós organizados numa estrutura hierárquica

> "ZooKeeper was designed to store coordination data: status information, configuration, location information, etc., so the data stored at each node is usually small, in the byte to kilobyte range."



Tipos de nós

- Permanentes
- Efémeros – que apenas existem enquanto o nó que os criou está a executar

ZOOKEEPER: OPERAÇÕES

Criar nó

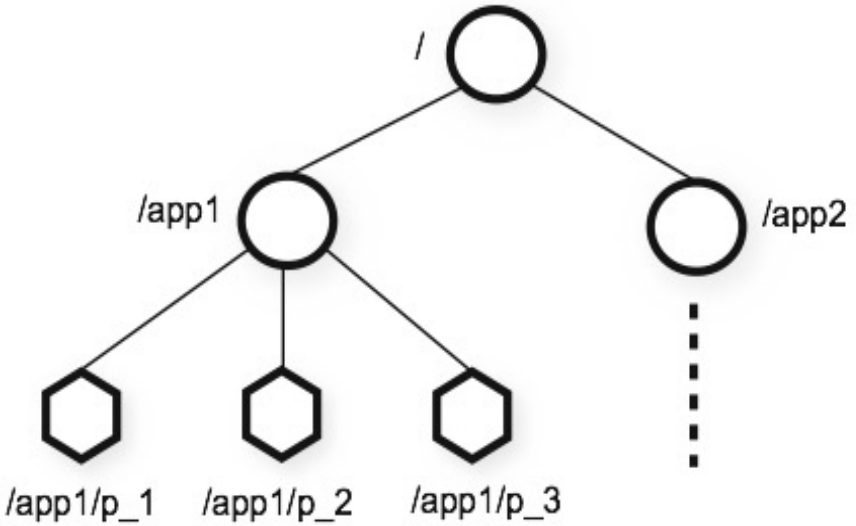
- Permanente ou efémero
- Normal ou sequencial (um nó sequencial é um nó ao qual é adicionado um número de sequência ao nome)

Ler / escrever valor do nó

Escrever condicionalmente

Remover nó

Notificação de alteração – pode-se pedir para ser notificado de alterações no sistema



ELEIÇÃO DO PRIMÁRIO: ALGORITMO

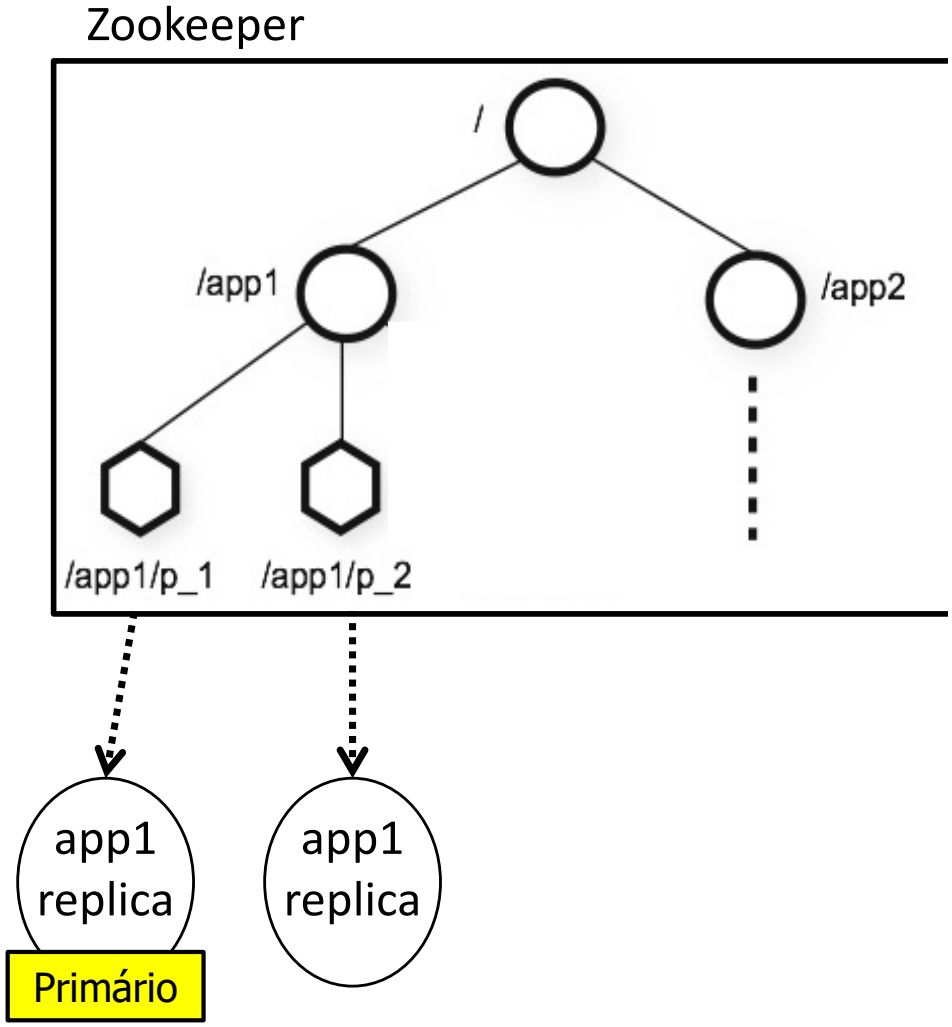
Cria-se diretoria para o serviço (e.g. "/app1")

Um servidor:

1. cria um nó na diretoria, com o tipo efémero e sequencial (e.g. "/app1/p_")
2. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário

Quando há uma alteração:

1. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário



ELEIÇÃO DO PRIMÁRIO: ALGORITMO

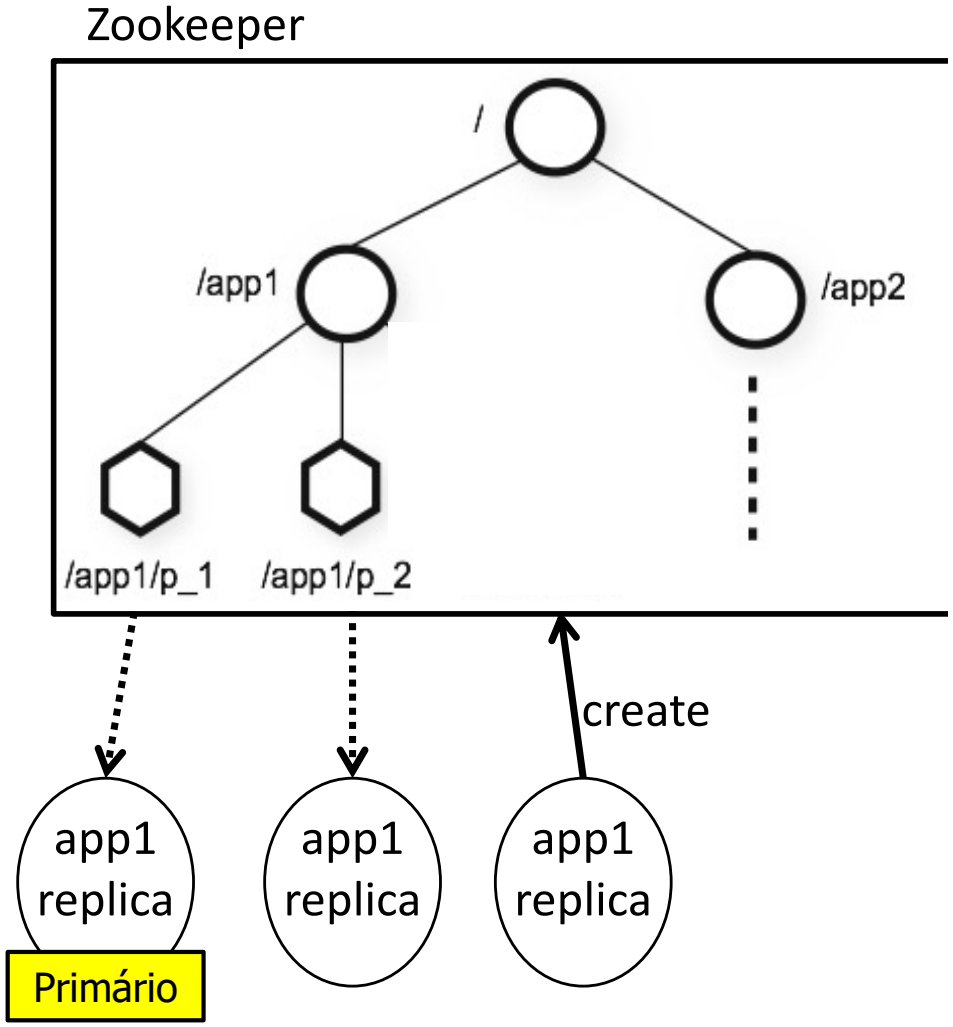
Cria-se diretoria para o serviço (e.g. "/app1")

Um servidor:

1. cria um nó na diretoria, com o tipo efémero e sequencial (e.g. "/app1/p_")
2. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário

Quando há uma alteração:

1. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário



ELEIÇÃO DO PRIMÁRIO: ALGORITMO

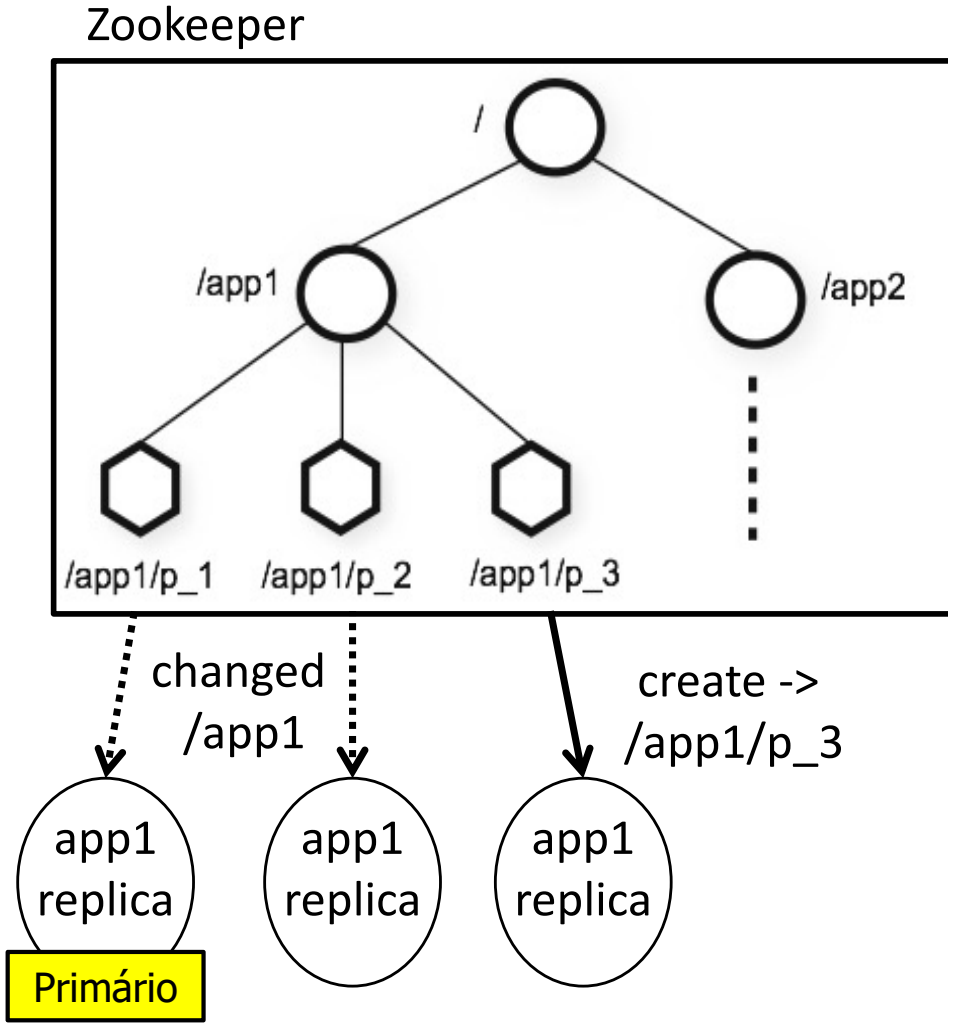
Cria-se diretoria para o serviço (e.g. "/app1")

Um servidor:

1. cria um nó na diretoria, com o tipo efémero e sequencial (e.g. "/app1/p_")
2. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário

Quando há uma alteração:

1. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário



ELEIÇÃO DO PRIMÁRIO: ALGORITMO

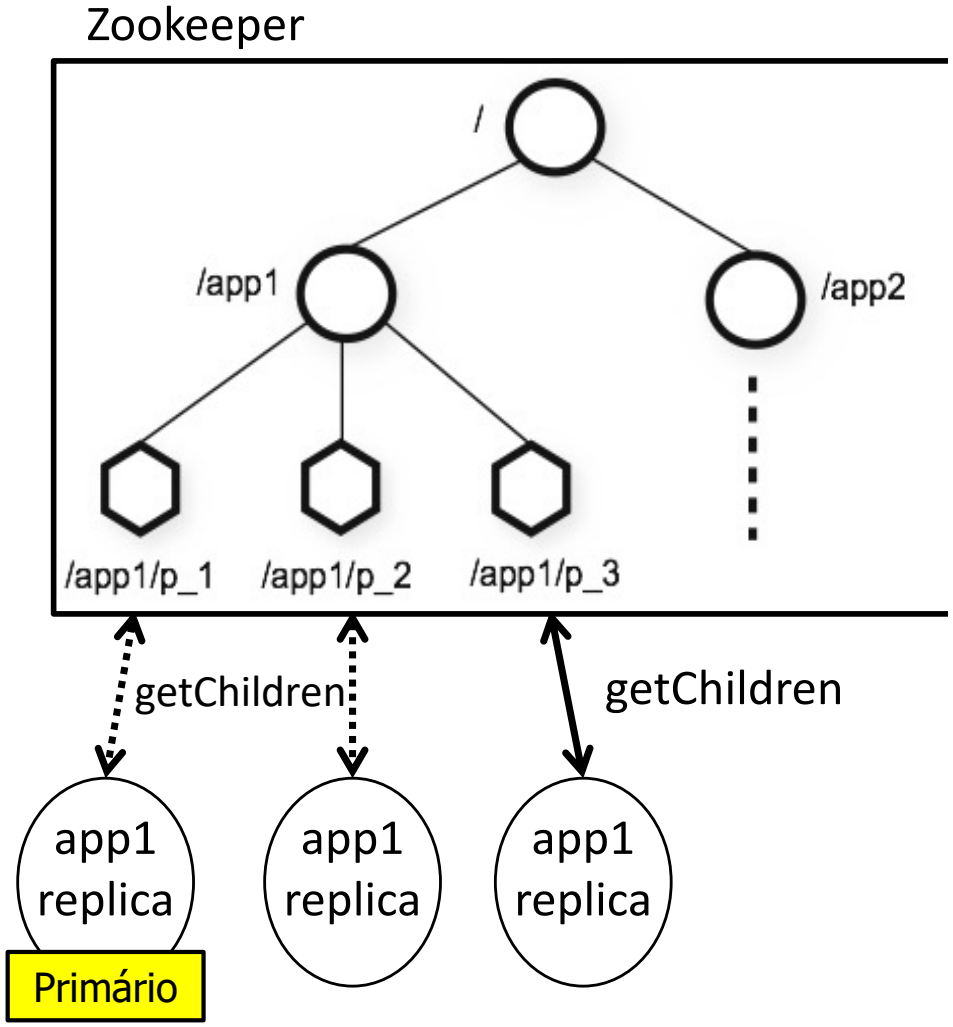
Cria-se diretoria para o serviço (e.g. "/app1")

Um servidor:

1. cria um nó na diretoria, com o tipo efémero e sequencial (e.g. "/app1/p_")
2. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário

Quando há uma alteração:

1. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário



ELEIÇÃO DO PRIMÁRIO: ALGORITMO

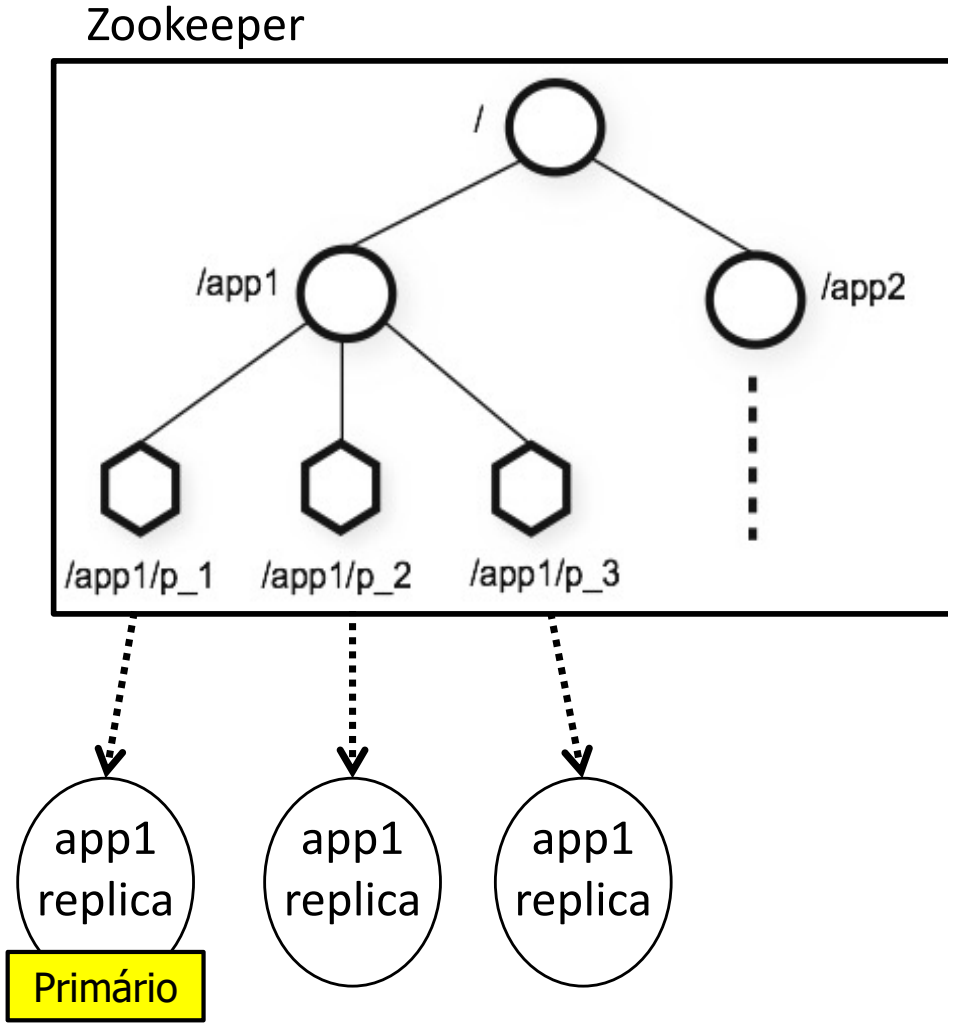
Cria-se diretoria para o serviço (e.g. "/app1")

Um servidor:

1. cria um nó na diretoria, com o tipo efémero e sequencial (e.g. "/app1/p_")
2. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário

Quando há uma alteração:

1. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário



ELEIÇÃO DO PRIMÁRIO: ALGORITMO

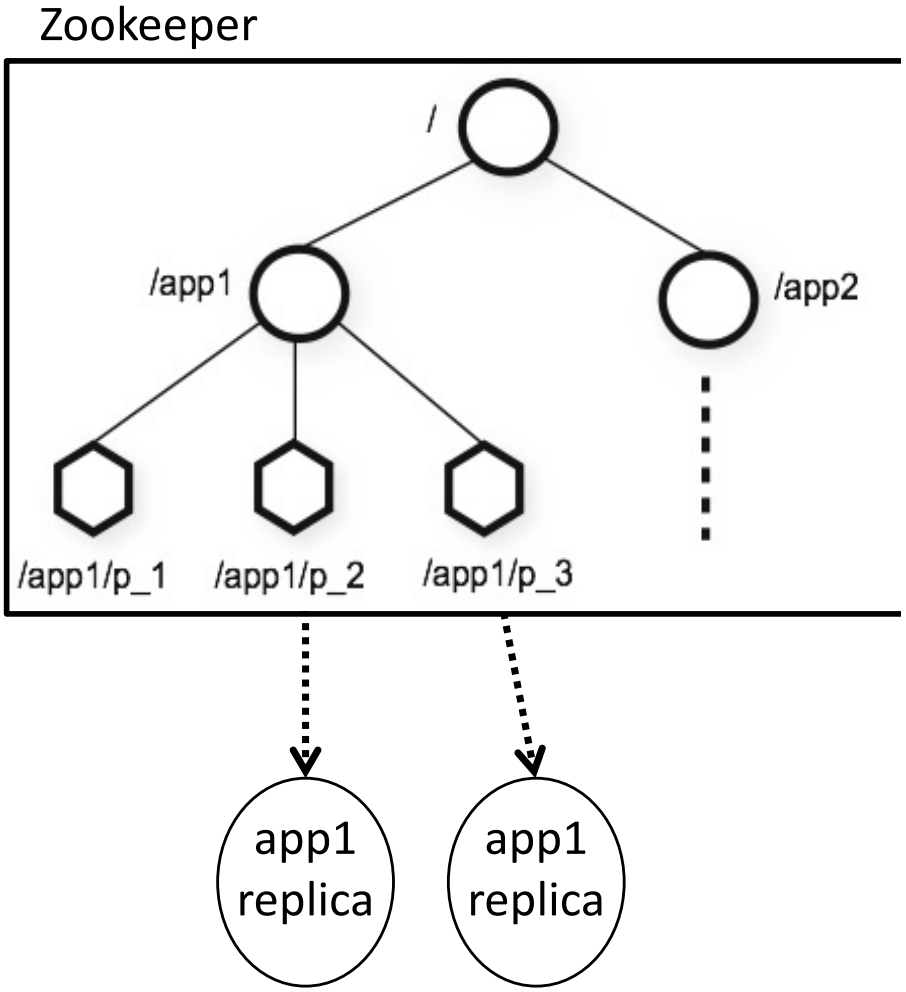
Cria-se diretoria para o serviço (e.g. "/app1")

Um servidor:

1. cria um nó na diretoria, com o tipo efémero e sequencial (e.g. "/app1/p_")
2. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário

Quando há uma alteração:

1. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário



ELEIÇÃO DO PRIMÁRIO: ALGORITMO

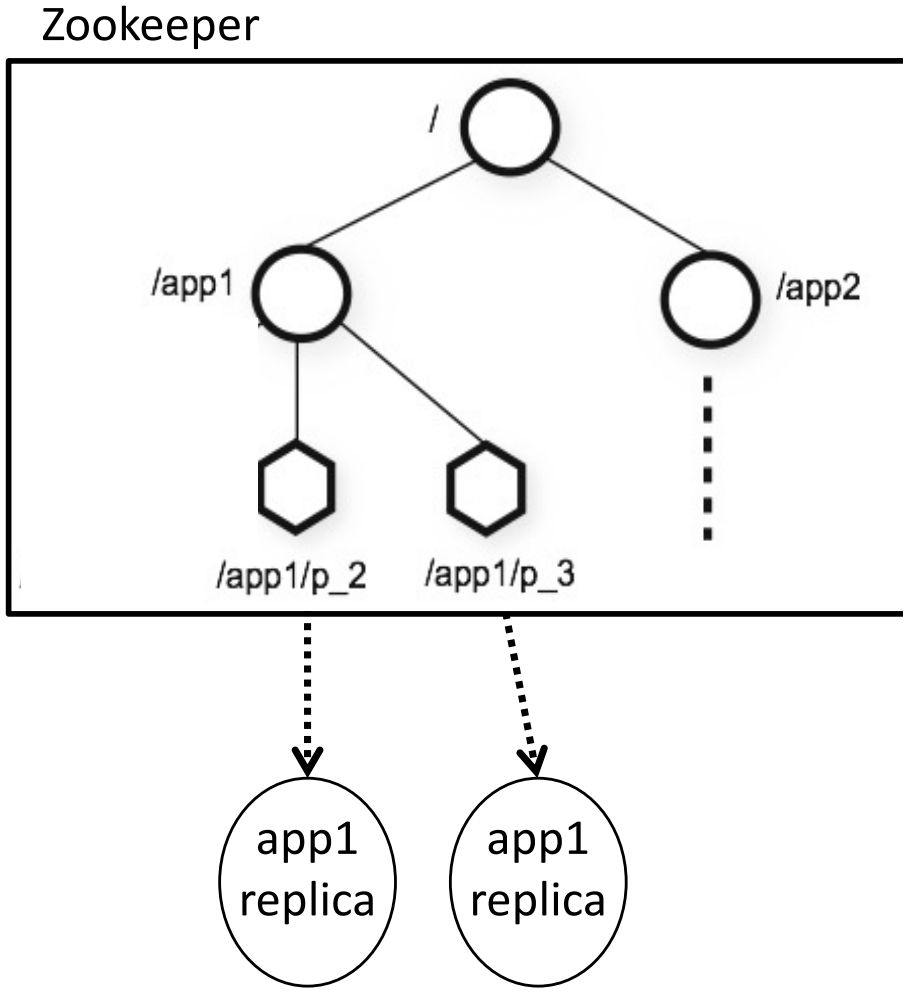
Cria-se diretoria para o serviço (e.g. "/app1")

Um servidor:

1. cria um nó na diretoria, com o tipo efémero e sequencial (e.g. "/app1/p_")
2. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário

Quando há uma alteração:

1. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário



ELEIÇÃO DO PRIMÁRIO: ALGORITMO

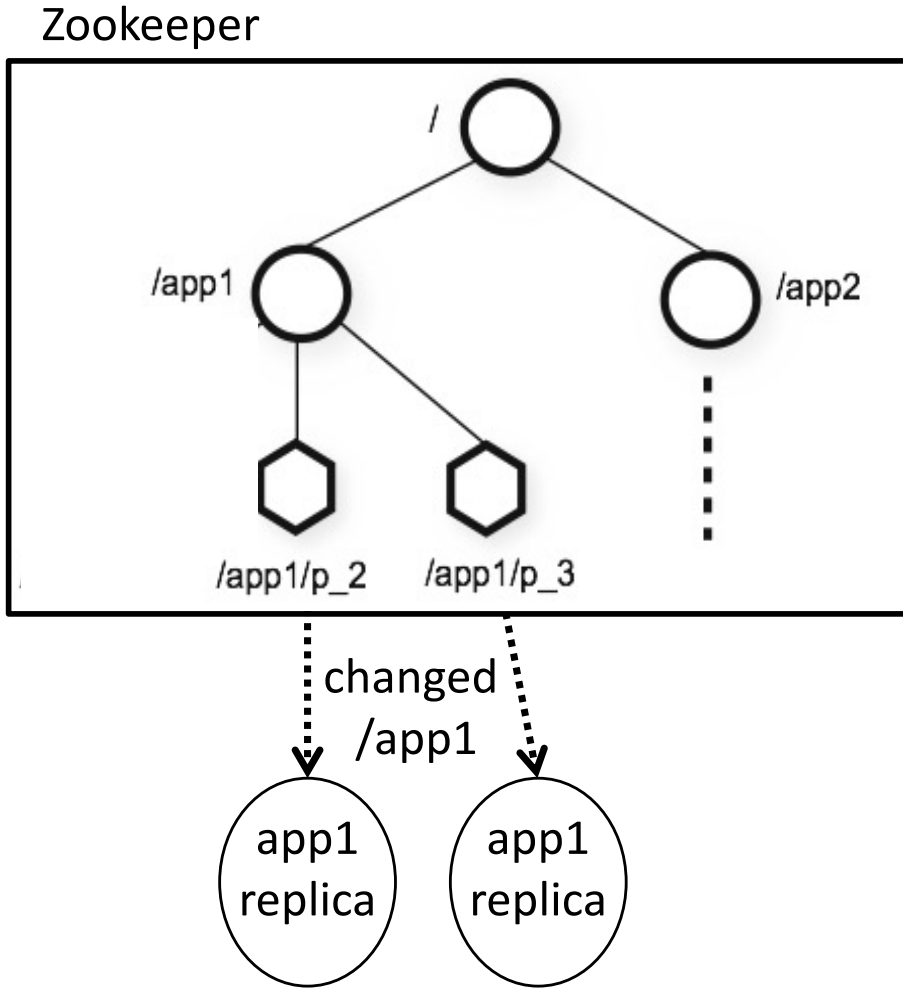
Cria-se diretoria para o serviço (e.g. "/app1")

Um servidor:

1. cria um nó na diretoria, com o tipo efémero e sequencial (e.g. "/app1/p_")
2. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário

Quando há uma alteração:

1. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário



ELEIÇÃO DO PRIMÁRIO: ALGORITMO

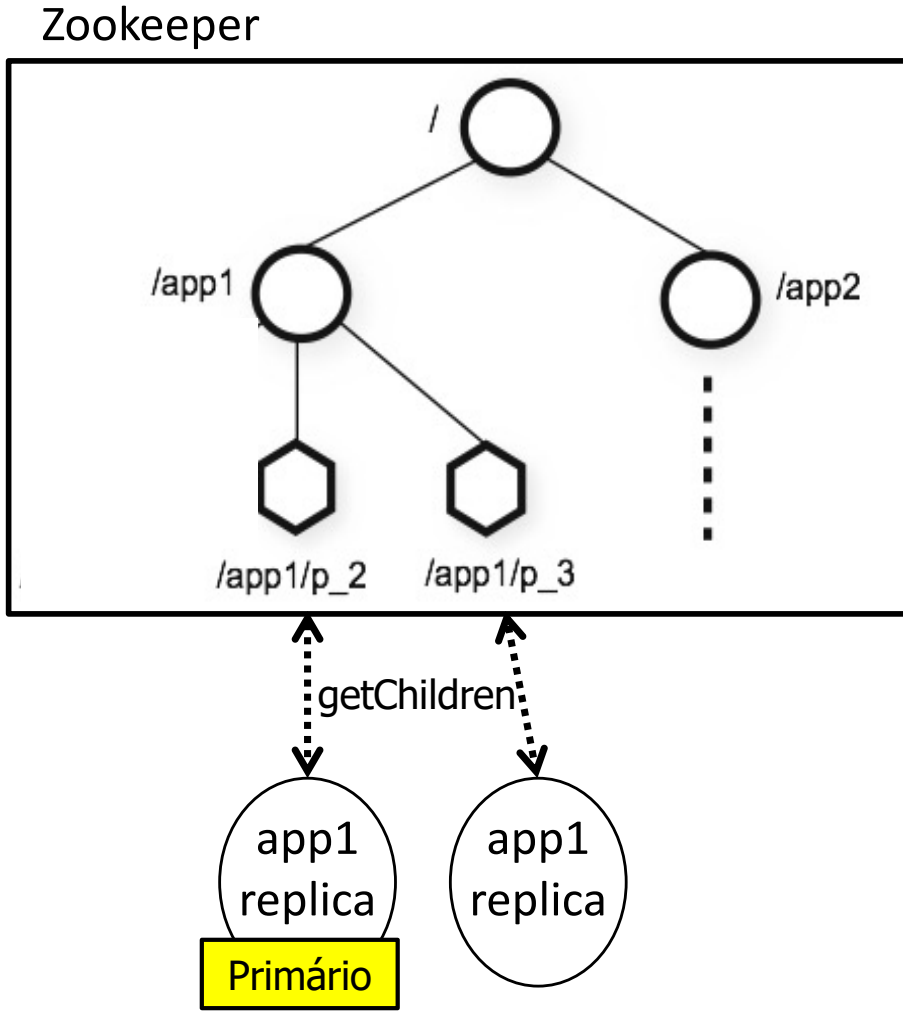
Cria-se diretoria para o serviço (e.g. "/app1")

Um servidor:

1. cria um nó na diretoria, com o tipo efémero e sequencial (e.g. "/app1/p_")
2. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário

Quando há uma alteração:

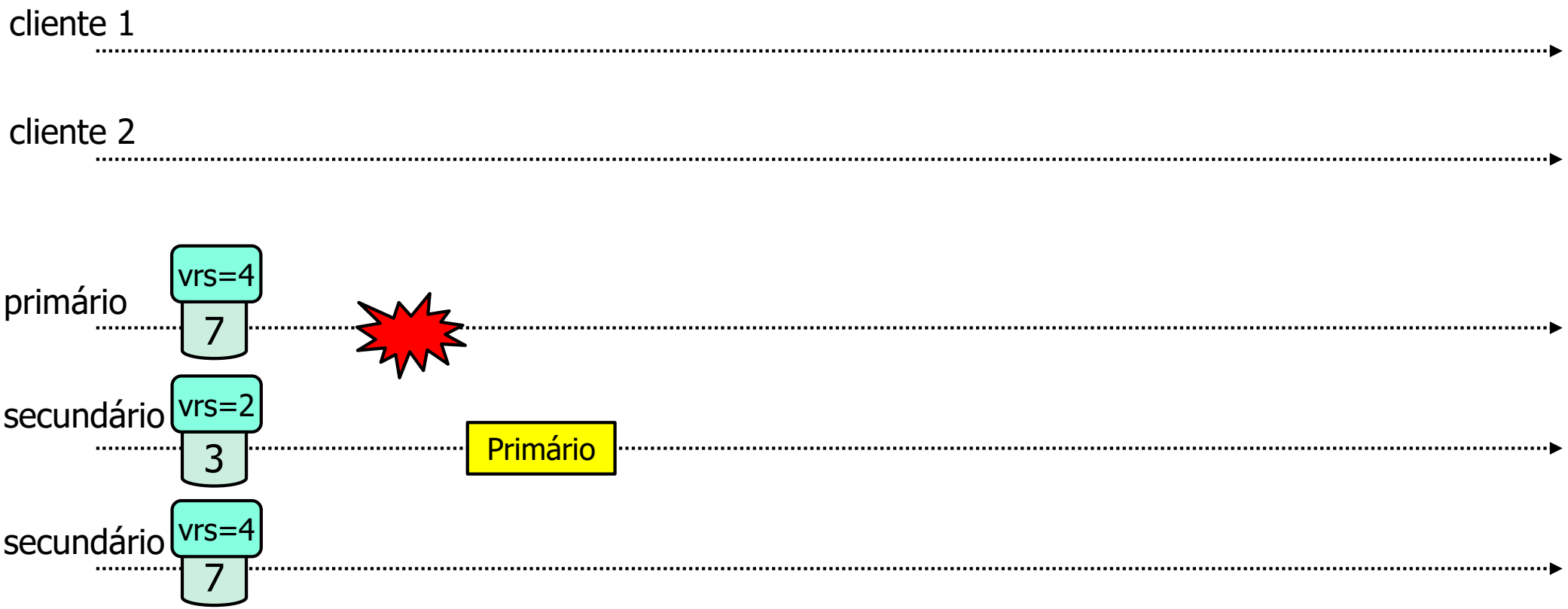
1. lista a diretoria e fica a observar alterações – se o seu nome for o menor, é o primário



PRIMÁRIO / SECUNDÁRIO : NOVO PRIMÁRIO

O que é que um nó faz quando descobre que há um novo primário?

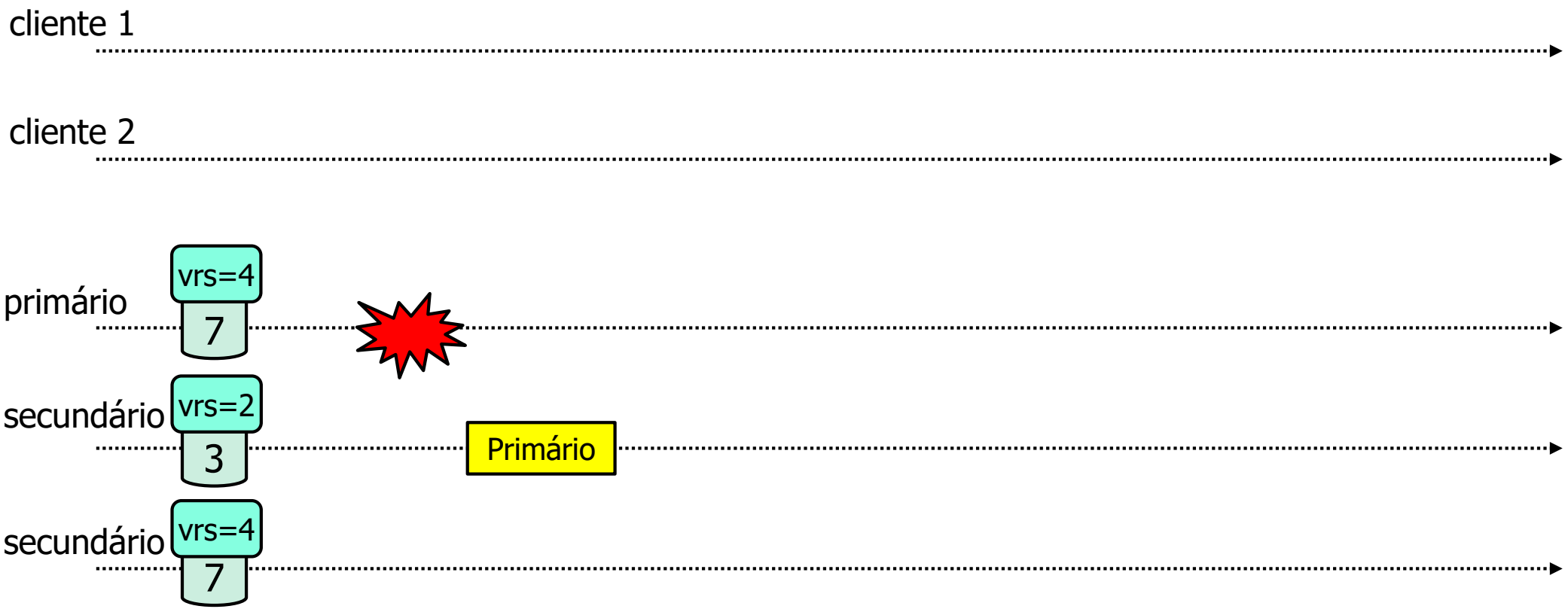
1. Deixa de receber mensagens do primário anterior (notem que o primário antigo pode não te falhado ou ter recuperado)



PRIMÁRIO / SECUNDÁRIO : NOVO PRIMÁRIO

O que é que o novo primário deve fazer?

1. Contactar nós para obter operações/versões que não tenha
2. Começar a executar protocolo normal



REPLICAÇÃO DE MÁQUINA DE ESTADOS

Como *garantir que todas as réplicas convergem para o mesmo estado* ?

Existem muitos outros algoritmos além do primário/secundário.

No próximo capítulo veremos como implementar replicação de máquina de estados usando um sistema de comunicação indireta.

RESUMO

Introdução à replicação

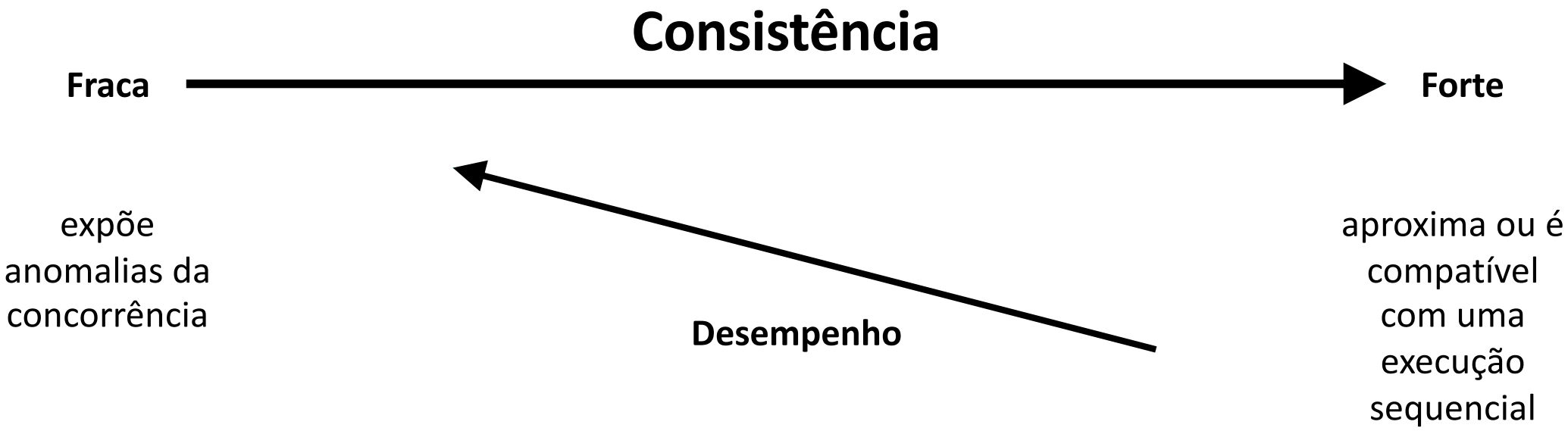
- Consistência forte. E.g.: primário-secundário
- **Consistência fraca.**

Caching

- Sistemas de ficheiros distribuídos
- Caching NFS
- Caching CIFS
- Caching Callback Promise

CONSISTÊNCIA

Qualidade que se refere à coerência dos dados face a atualizações concorrentes…(vs. uma execução puramente sequencial)



CONSISTÊNCIA "EVENTUAL" / FINA

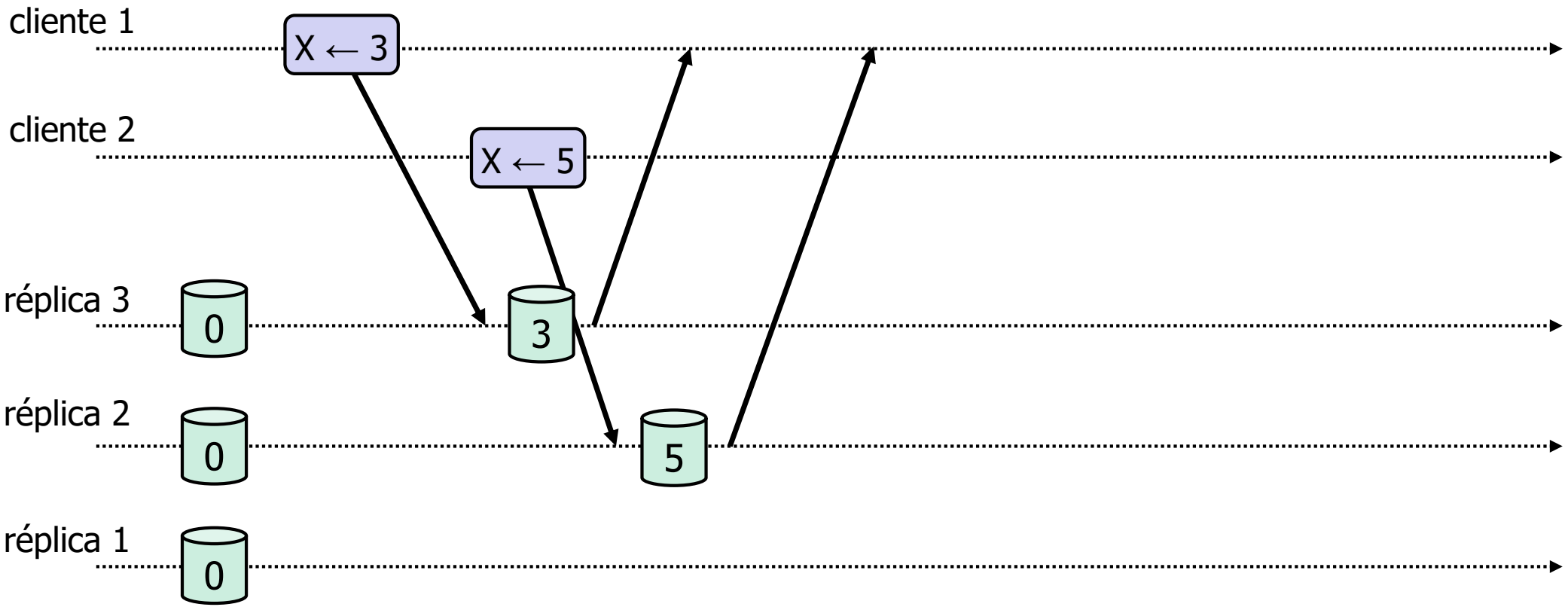
Cliente: os clientes podem **executar** as operações em **qualquer** servidor

Réplicas: **todos** os servidores podem **receber** operações e executá-las **imediatamente**

CONSISTÊNCIA "EVENTUAL" / FINAL

Cliente: os clientes podem **executar** as operações em **qualquer** servidor

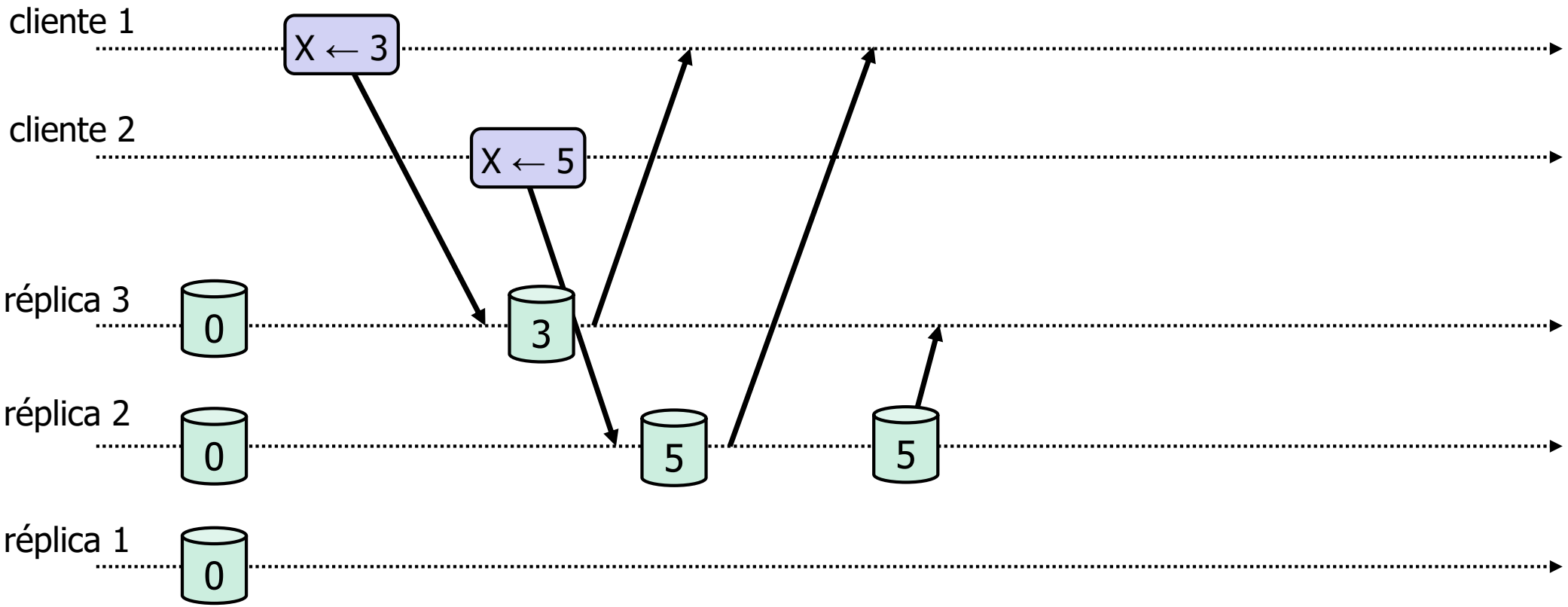
Réplicas: **todos** os servidores podem **receber** operações e executá-las **imediatamente**



CONSISTÊNCIA "EVENTUAL" / FINAL

Sincronização epidémica: réplicas comunicam periodicamente **trocando** as suas atualizações (podem enviar o estado ou a lista de operações)

- Como lidar com escritas concorrentes?
- **Possível solução**: última escrita ganha => ordem total sobre as escritas
- Como ordenar totalmente as operações?

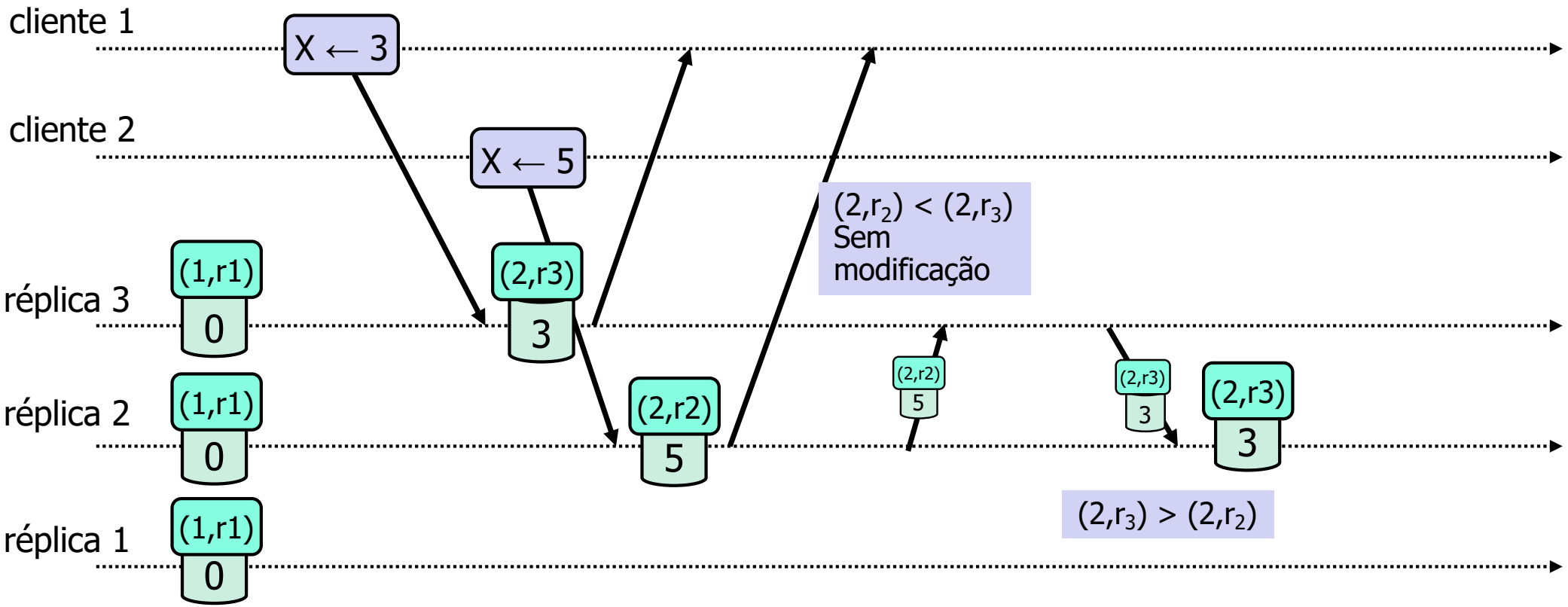


CONSISTÊNCIA "EVENTUAL" / FINAL

Como ordenar as operações? Uma hipótese...

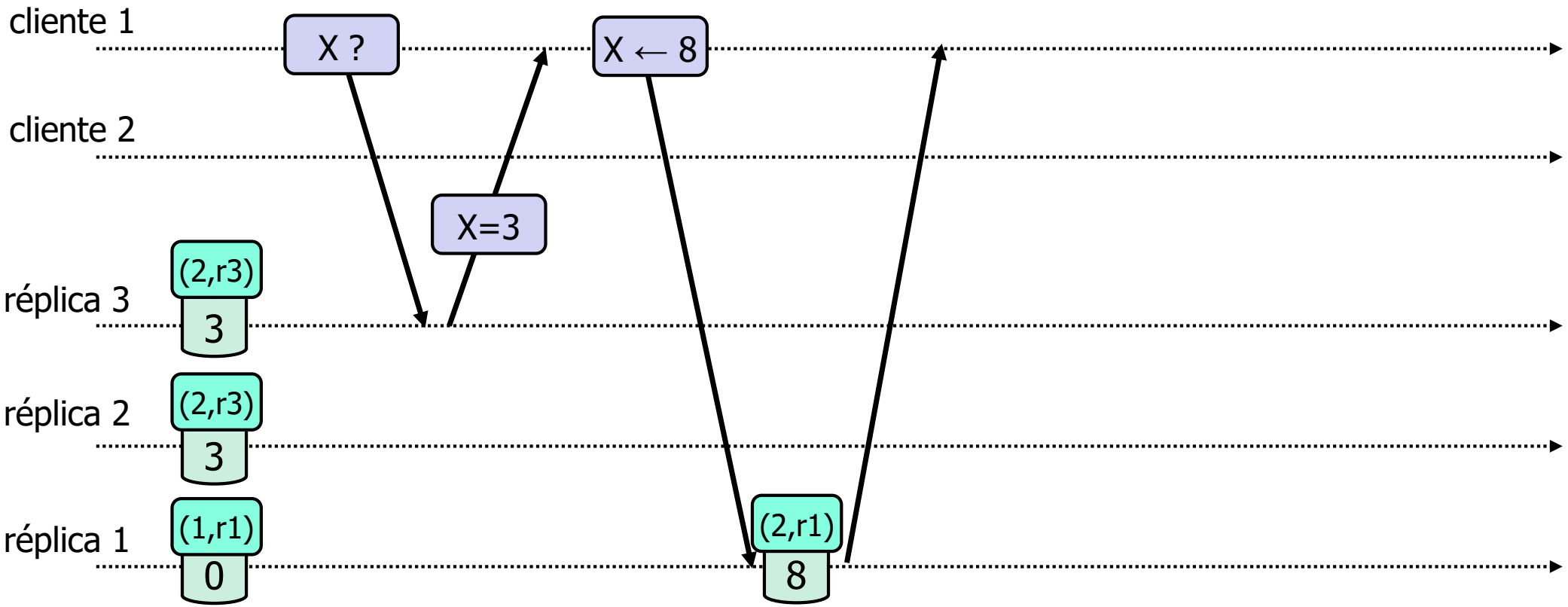
(relógio lógico de Lamport, id da réplica)

(t1,r1) < (t2,r2) sse t1 < t2 OR (t1 = t2 AND r1 < r2)



CONSISTÊNCIA "EVENTUAL" / FINAL

Problema?

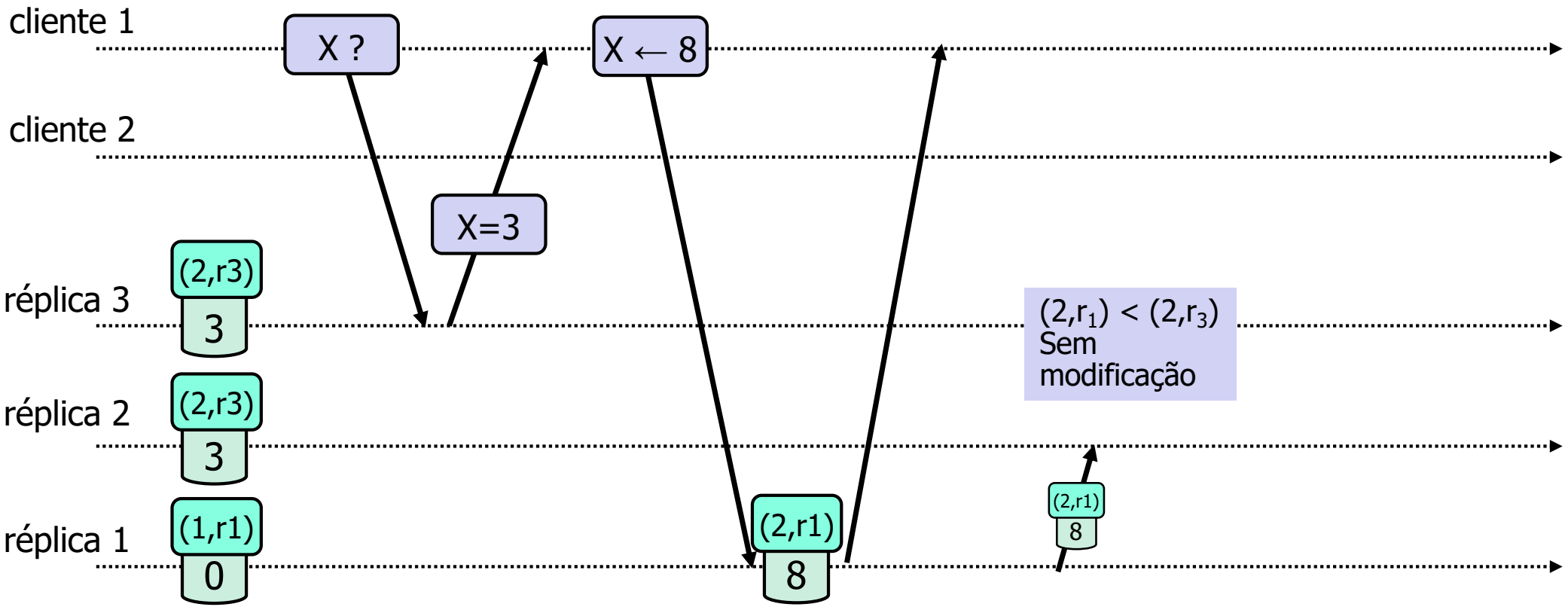


CONSISTÊNCIA "EVENTUAL" / FINAL

Problema?

Como $(2,r_1) < (2,r_3)$, a última escrita é preterida.

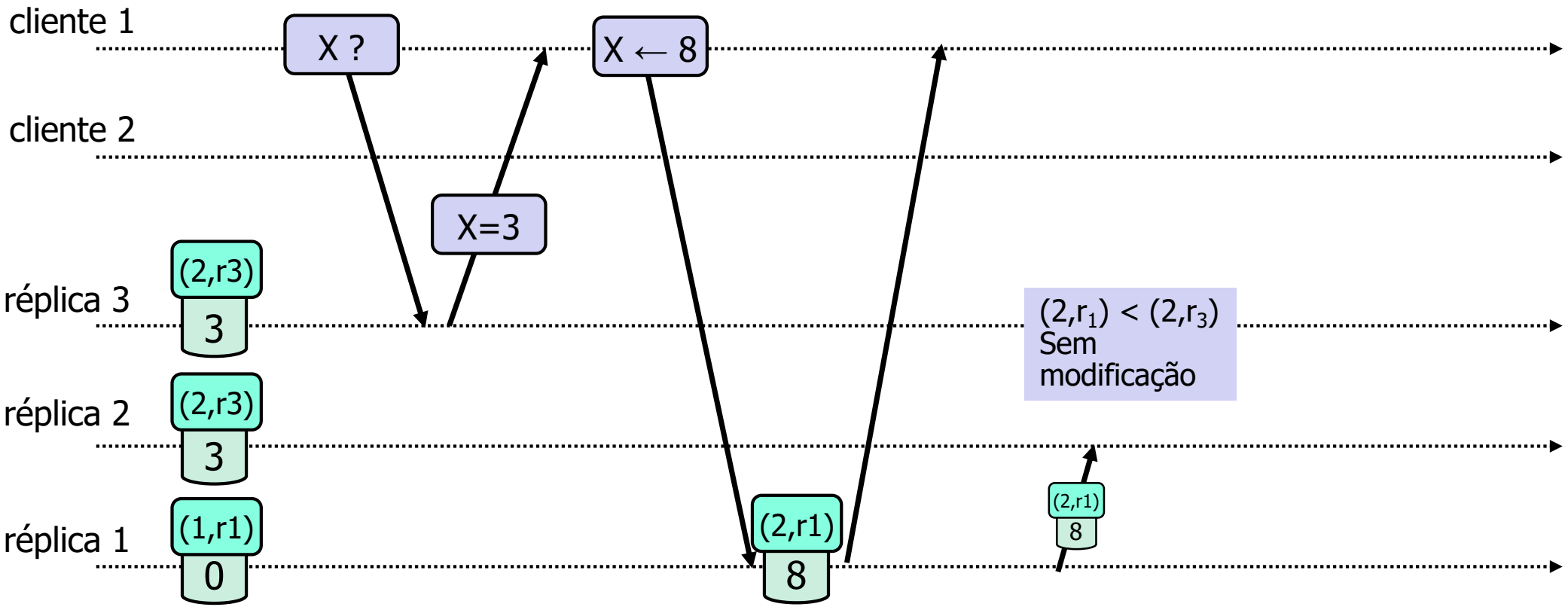
Dado que o cliente fez a escrita X=8 depois de ver o valor X=3, o resultado deveria ser X=9.



CONSISTÊNCIA "EVENTUAL" / FINAL

Propagar a causalidade é importante.

Solução?

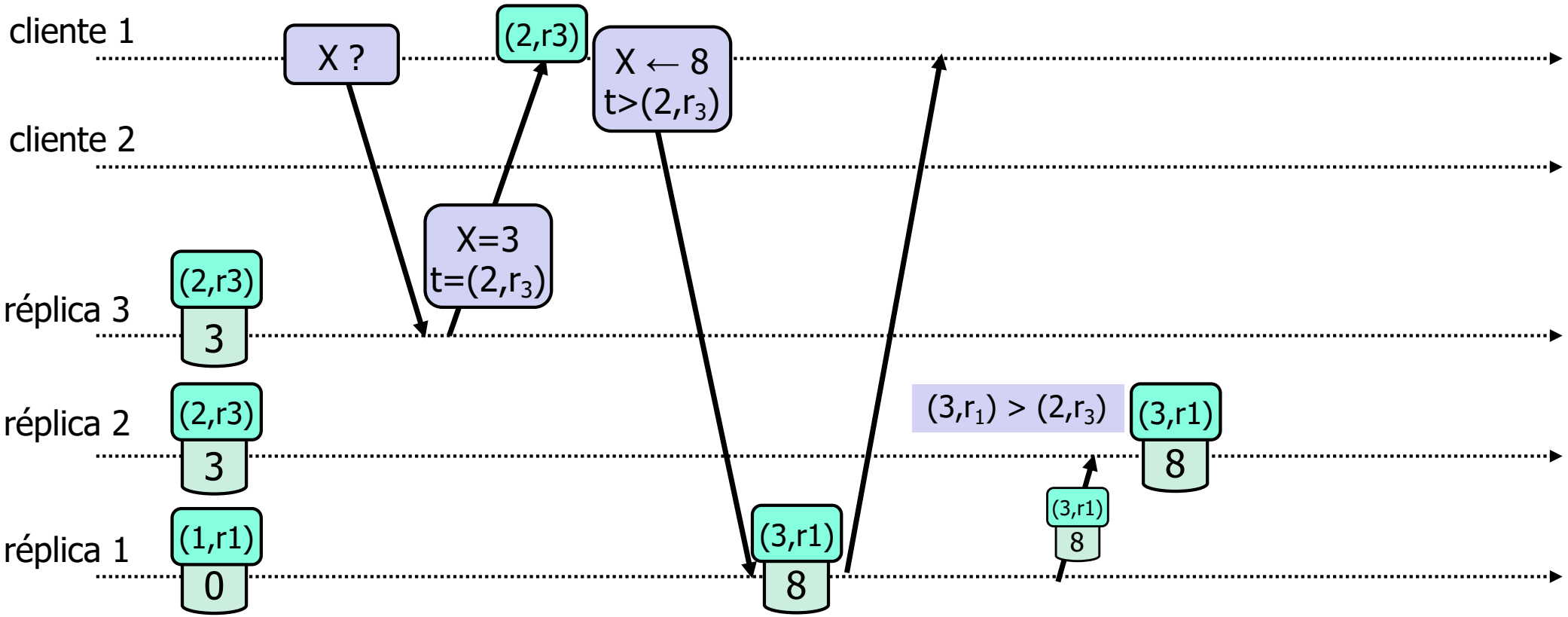


CONSISTÊNCIA "EVENTUAL" / FINAL

Propagar a causalidade é importante.

Solução?

Propagar relógios lógicos para o cliente.



CONSISTÊNCIA EVENTUAL

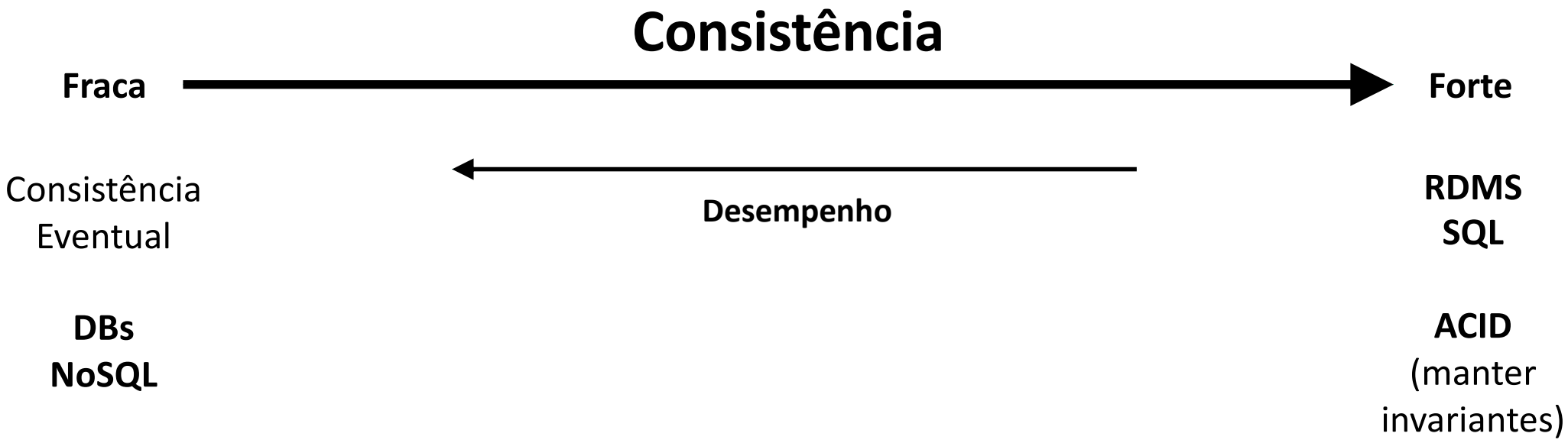
Replicação, cujo modelo de consistência entre réplicas promete **elevada disponibilidade** e que, **quando as atualizações terminarem**, as leituras acabarão por devolver **o mesmo valor**.

Ou seja, garante que **as réplicas irão convergir para um mesmo valor se não houver mais atualizações.**

Uma consequência é que **duas réplicas que tenham visto o mesmo conjunto de atualizações têm o mesmo estado.**

CONSISTÊNCIA

Qualidade que se refere à coerência dos dados face a atualizações concorrentes…(vs. uma execução puramente sequencial)



AMAZON DYNAMO (2007)

Sistema de bases de dados NoSQL

Replicação geográfica com elevada disponibilidade e desempenho

Operações que atualizam o estado podem executar sem ver os efeitos uma da outra.

"Eventually", todas as operações são propagadas a todas as réplicas.

DYNAMO: INTERFACE

get(key) → returns <list of values,context>

- Retorna lista de escritas mais recentes que não viram os efeitos uma da outra (de forma a nenhuma se perder)
- **Context** descreve o conjunto de escritas que são reflectidas na lista de valores retornada

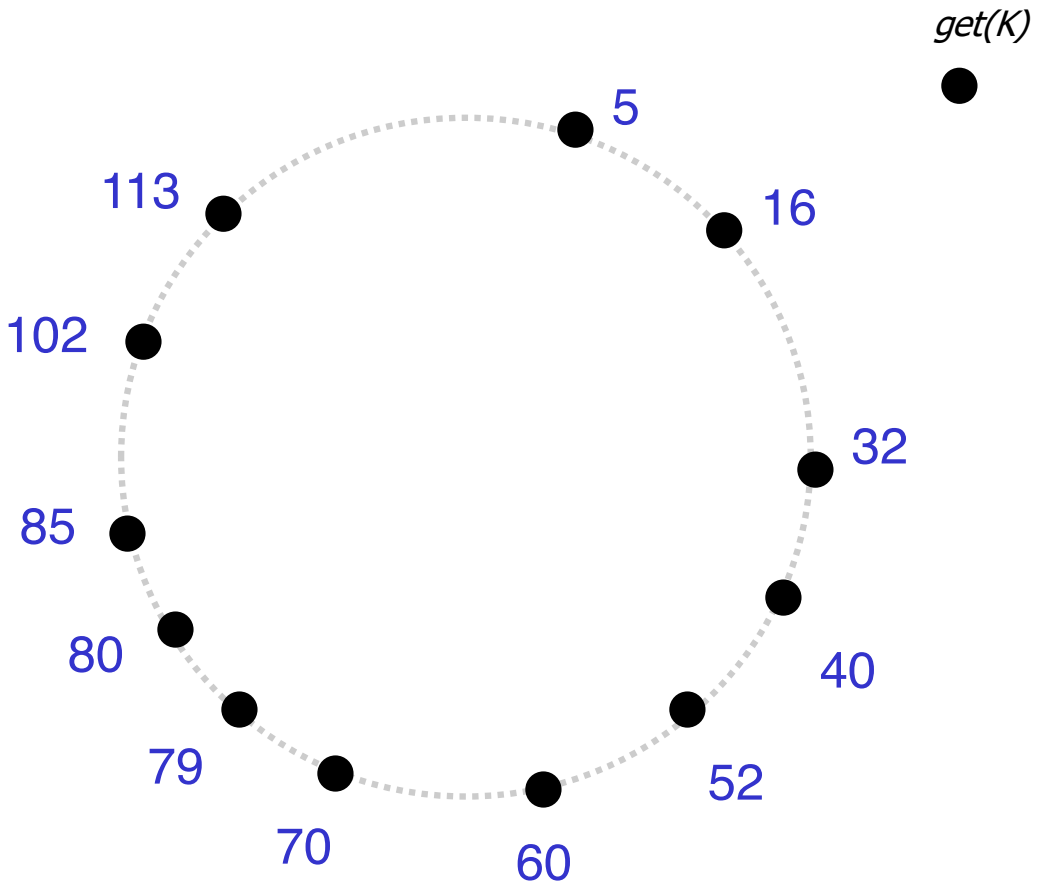
put(key,value,context) → returns "ack"

- Escreve o novo valor "value"
- Cliente deve passer o contexto associado ao get mais recente no qual a escrita se baseia

DYNAMO: IMPLEMENTAÇÃO

Nós do sistema organizados numa DHT one-hop, i.e., uma DHT em que um nó conhece todos os outros.

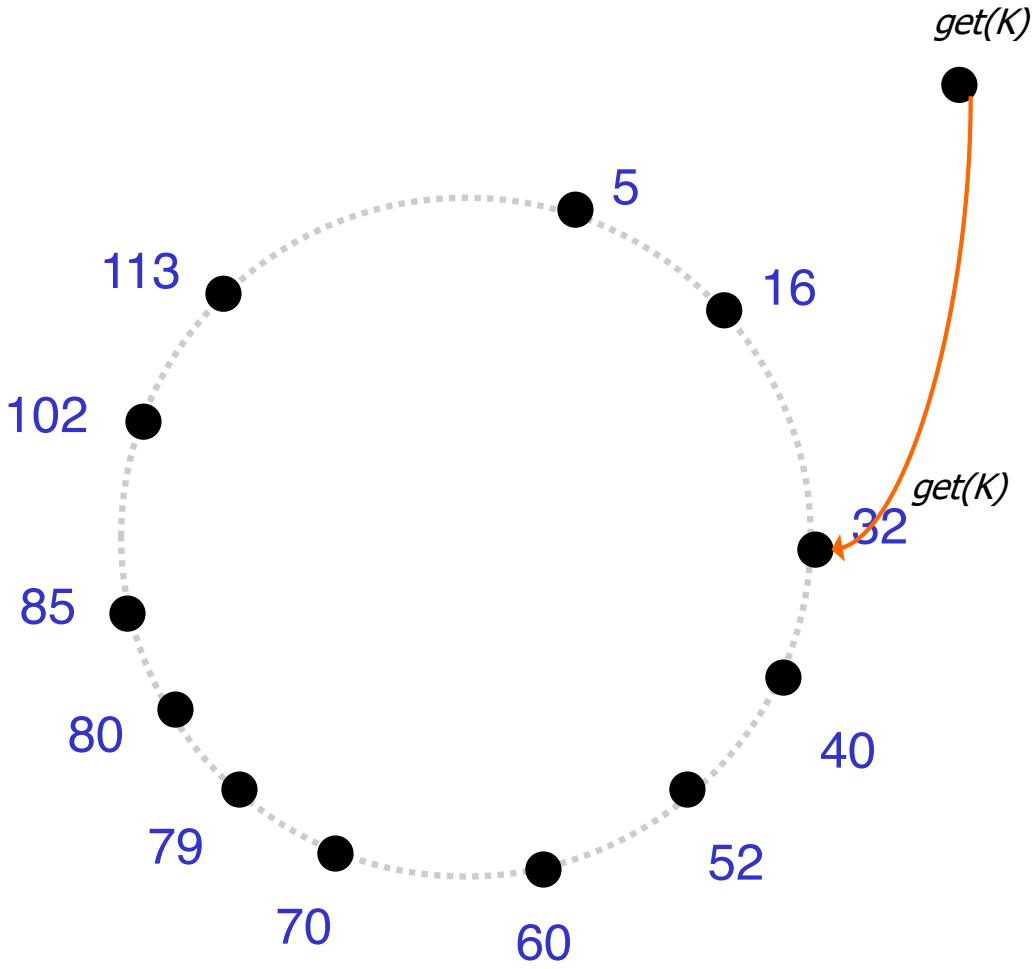
Quóruns de leitura e escrita: subconjuntos de R e W réplicas, com R+W<N (podem não se intersectar)



DYNAMO: IMPLEMENTAÇÃO

Operações executam-se numa única fase:

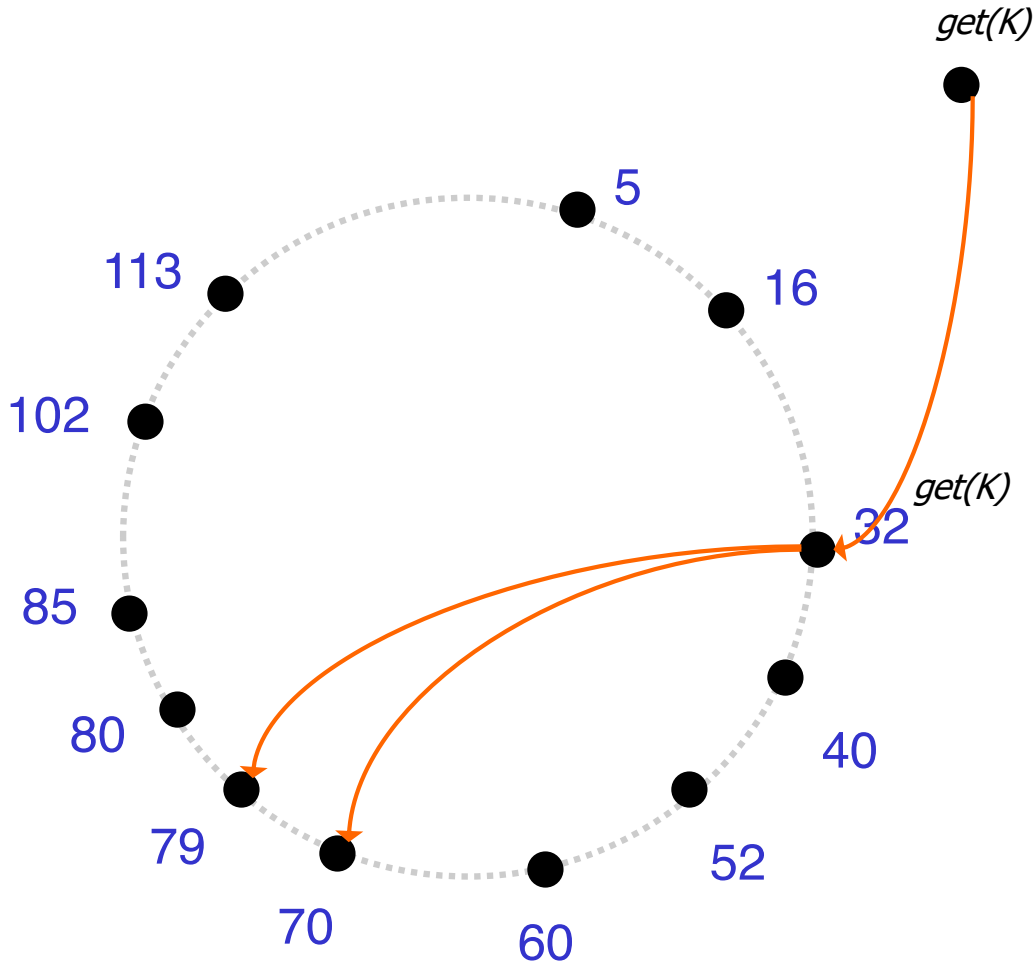
1. Cliente contacta uma das réplicas escolhida aleatoriamente



DYNAMO: IMPLEMENTAÇÃO

Operações executam-se numa única fase:

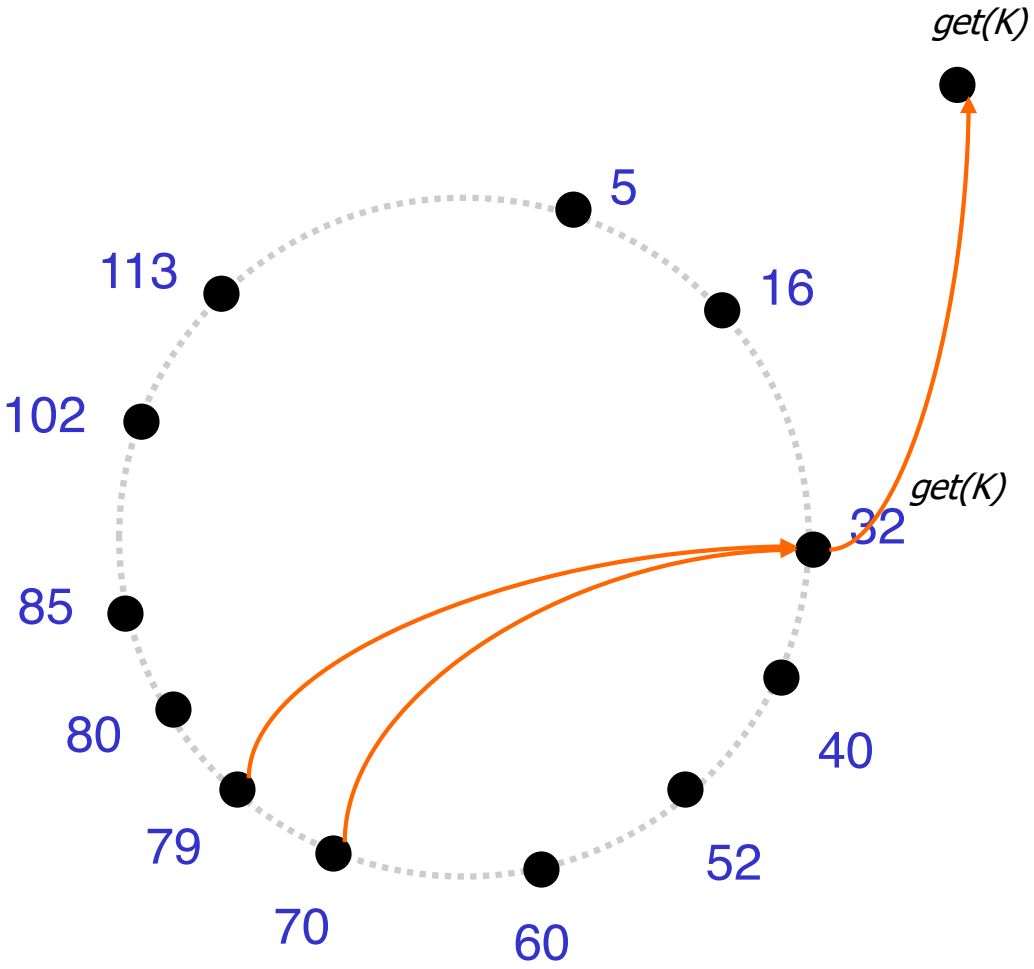
1. Cliente contacta uma das réplicas escolhida aleatoriamente
2. Réplica propaga pedido de get/put para N réplicas



DYNAMO: IMPLEMENTAÇÃO

Operações executam-se numa única fase:

1. Cliente contacta uma das réplicas escolhida aleatoriamente
2. Réplica propaga pedido de get/put para N réplicas
3. Espera resposta de R/W réplicas antes de retornar ao cliente
4. No caso do get, retorna o(s) valor(es) mais recente(s) – mais do que um no caso de escritas concorrentes



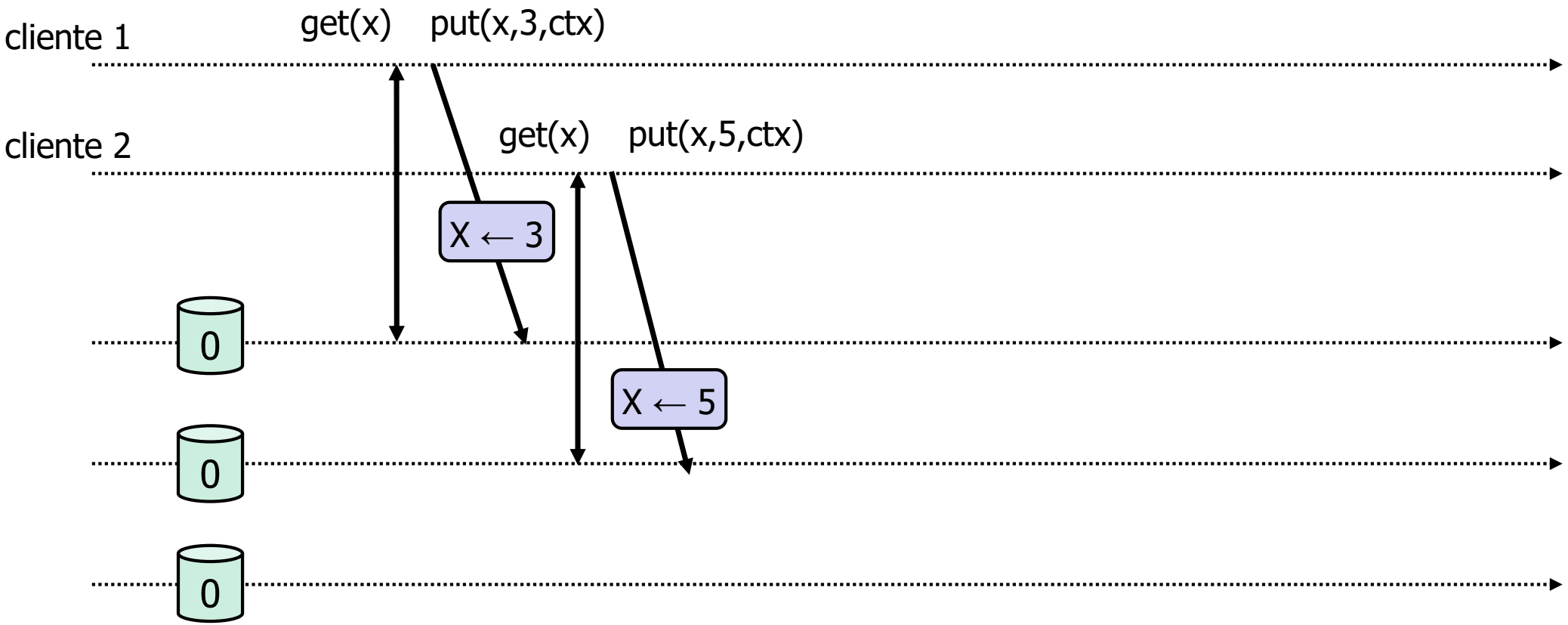
DYNAMO: ESCRITAS CONCORRENTES

No Dynamo podem ocorrer escritas concorrentes porque:

- Vários clientes podem escrever ao mesmo tempo

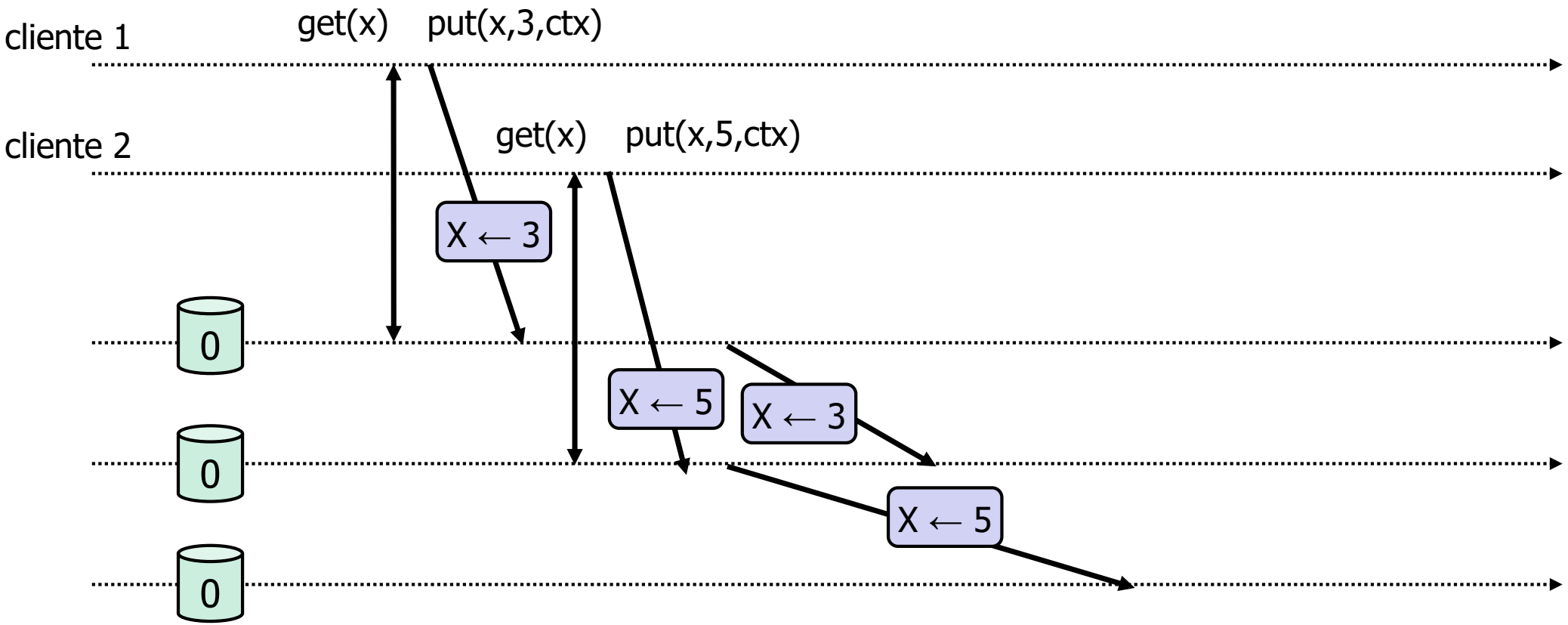
DYNAMO: ESCRITAS CONCORRENTES

Dois clientes fazem escritas concorrentes, i.e., sem conhecerem a escrita do outro.



DYNAMO: ESCRITAS CONCORRENTES

Dois clientes fazem escritas concorrentes, i.e., sem conhecerem a escrita do outro.



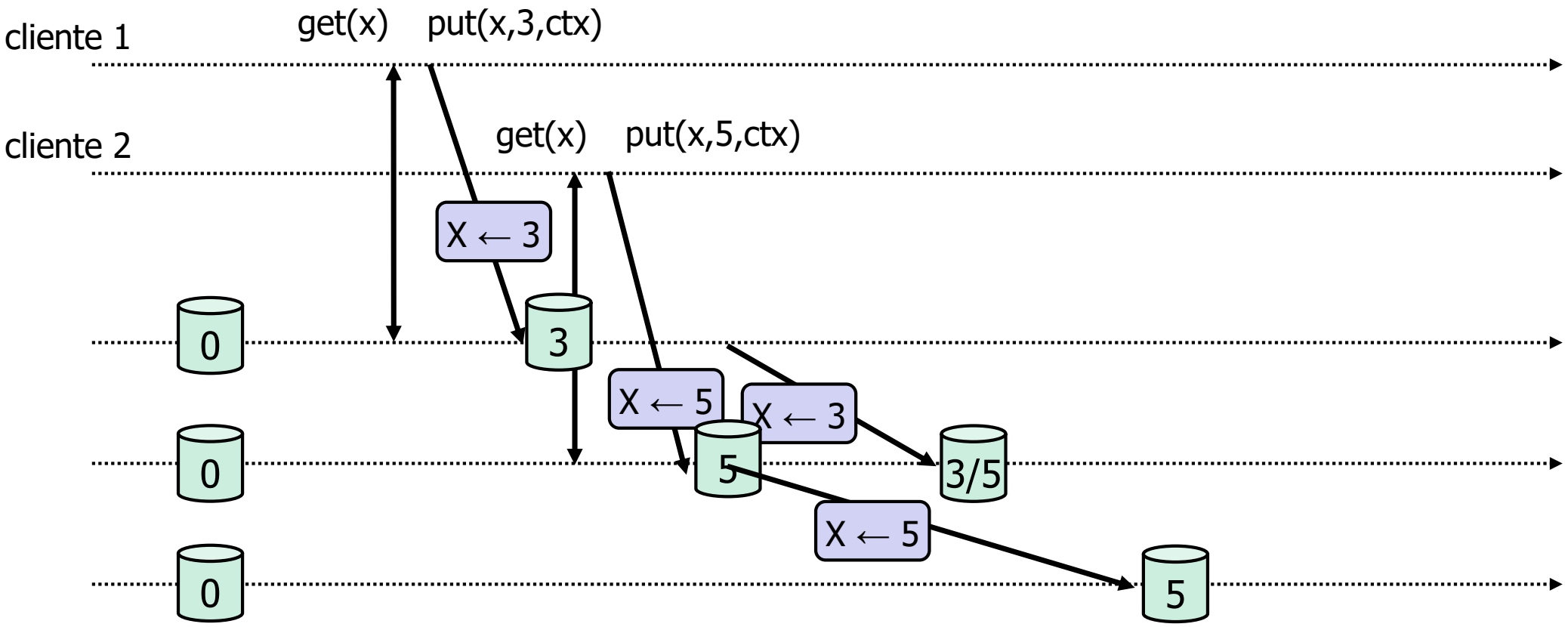
DYNAMO: ESCRITAS CONCORRENTES

No Dynamo podem ocorrer escritas concorrentes porque:

- Vários clientes podem escrever ao mesmo tempo
- Sistema permite que se escreve em servidores que não replicam a chave (sloppy quorums)

DYNAMO: COMO LIDAR COM ESCRITAS CONCORRENTES?

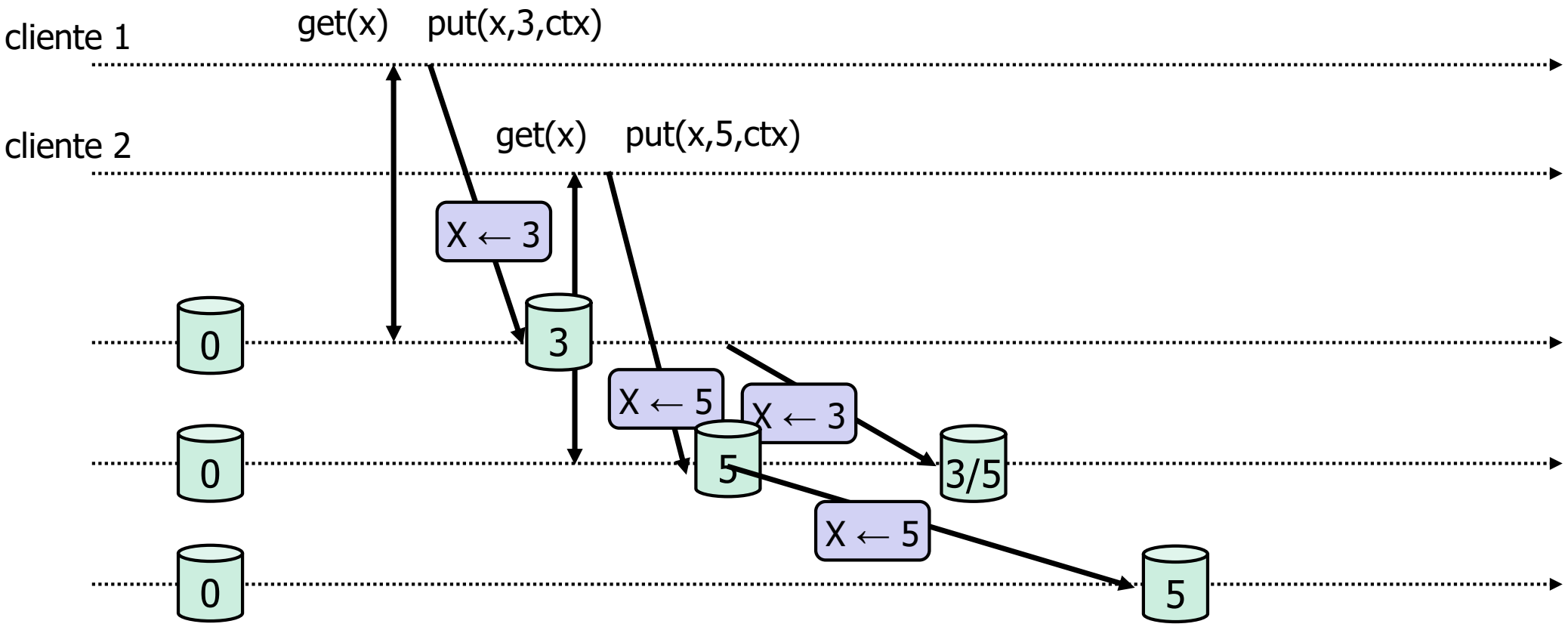
Sistemas mantém as várias escritas concorrentes e devolve-as ao cliente na leitura.



DYNAMO: ESCRITAS CONCORRENTES

Como detetar escritas concorrentes?

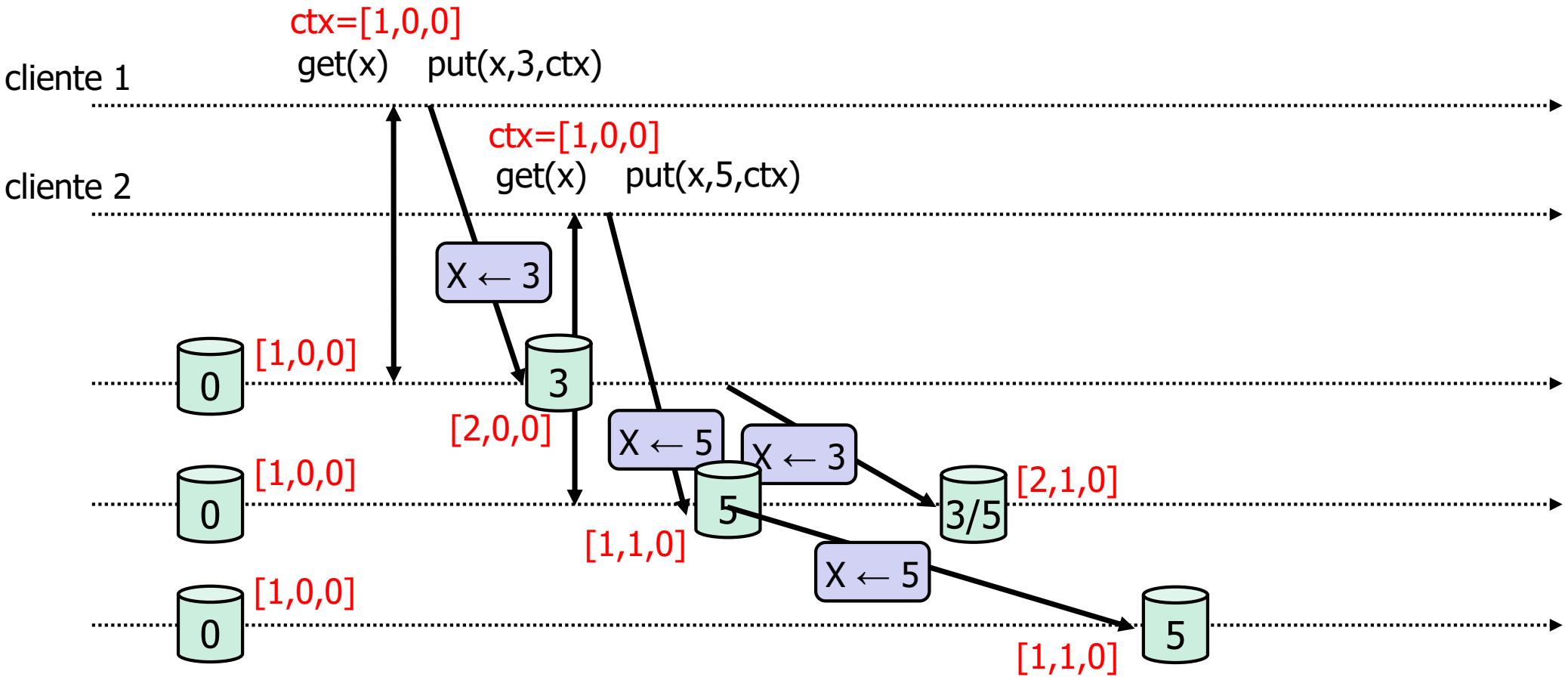
Usando vetores versão.



DYNAMO: ESCRITAS CONCORRENTES

Como detetar escritas concorrentes?

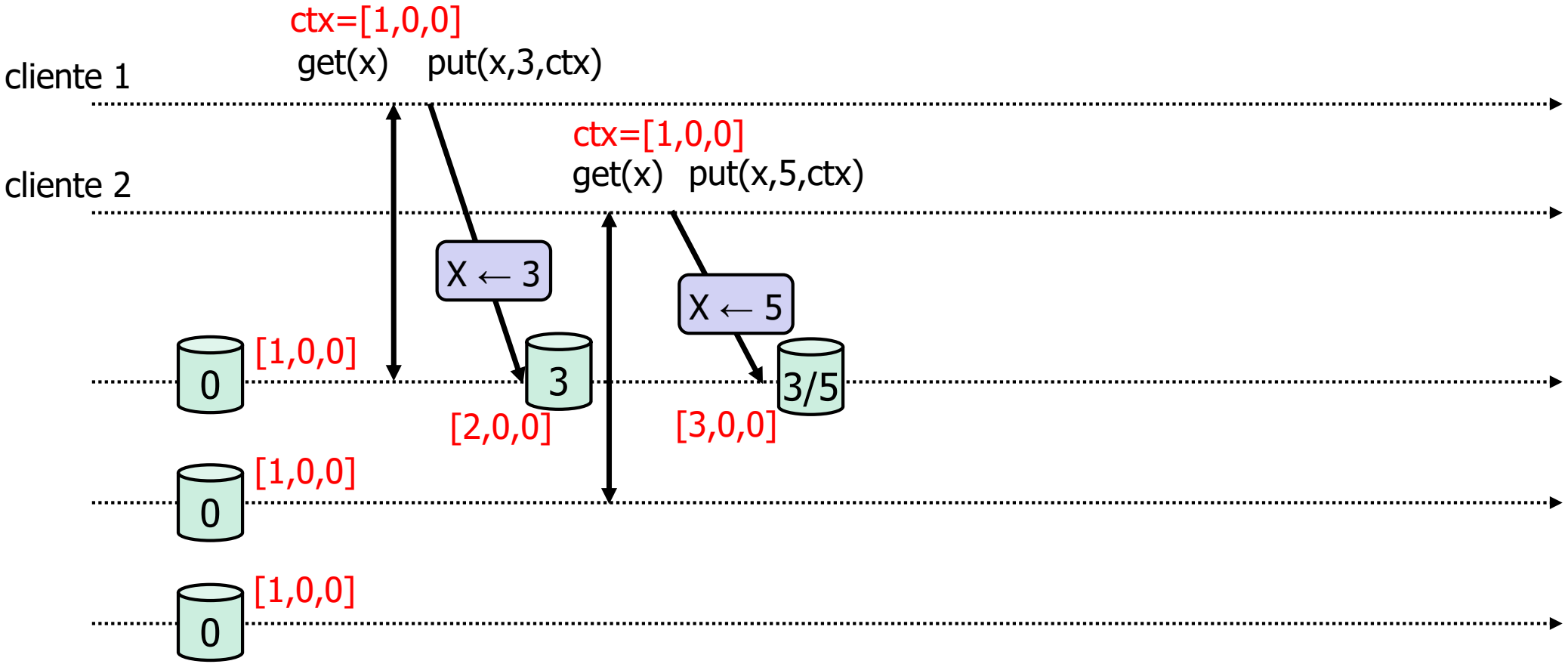
Usando vetores versão.



DYNAMO: ESCRITAS CONCORRENTES (ALTERNATIVA 2)

Como detetar escritas concorrentes?

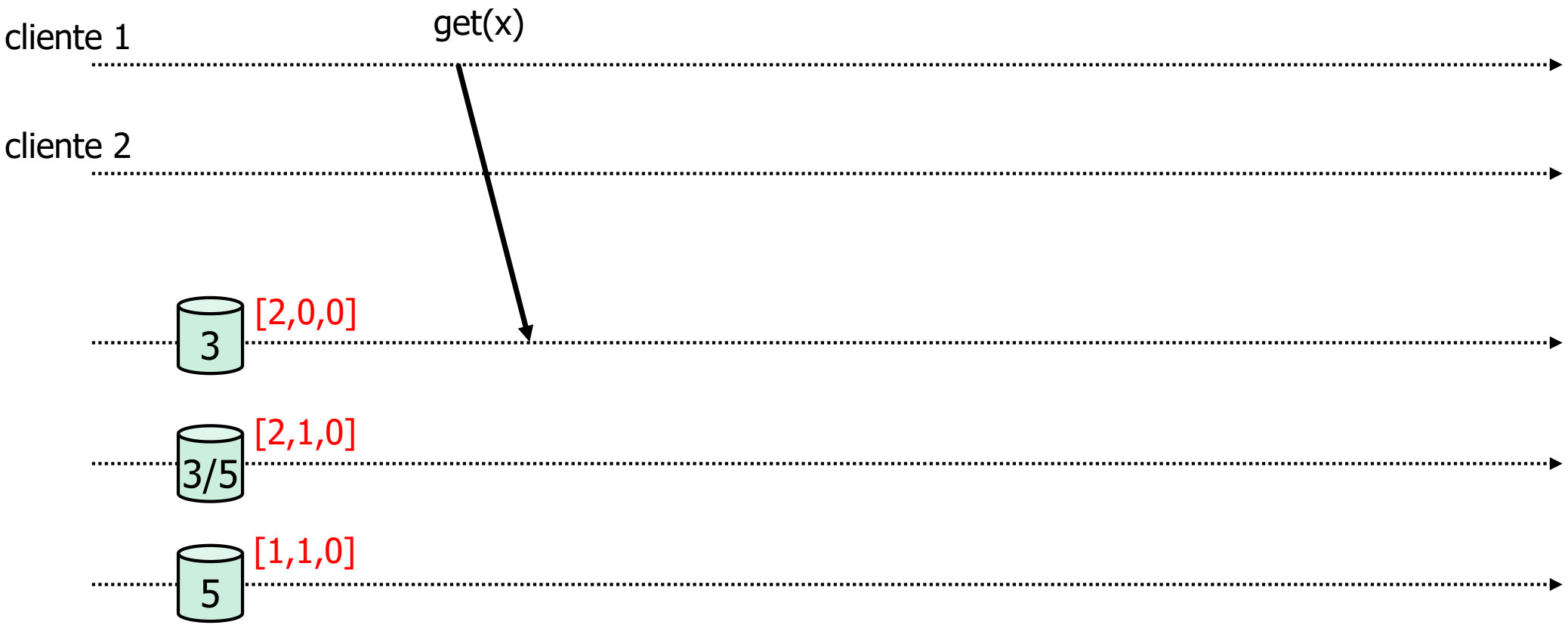
Quando se recebe uma escrita, se contexto menor que contexto atual é porque houve escrita – e.g. [1,0,0] < [2,0,0]



DYNAMO: ESCRITAS CONCORRENTES

Como é que a leitura devolve escritas concorrentes?

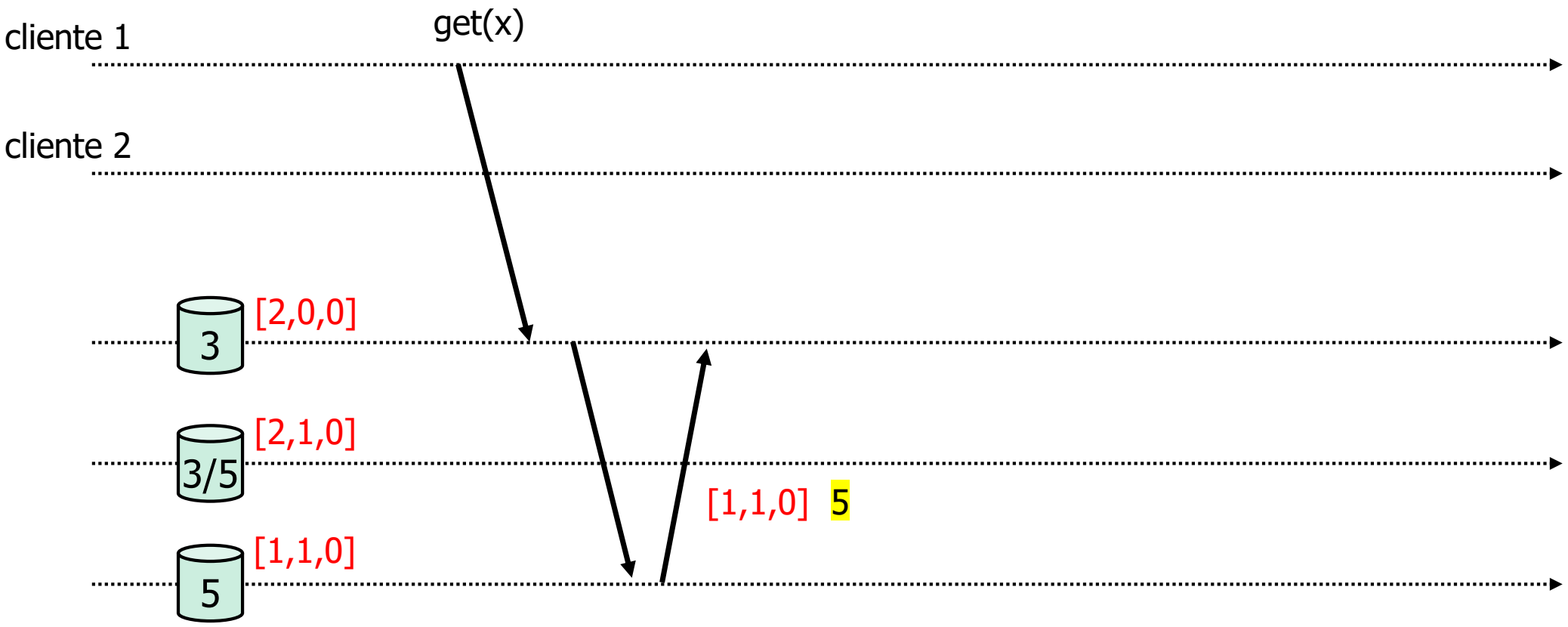
1. Cliente contacta réplica



DYNAMO: ESCRITAS CONCORRENTES

Como é que a leitura devolve escritas concorrentes?

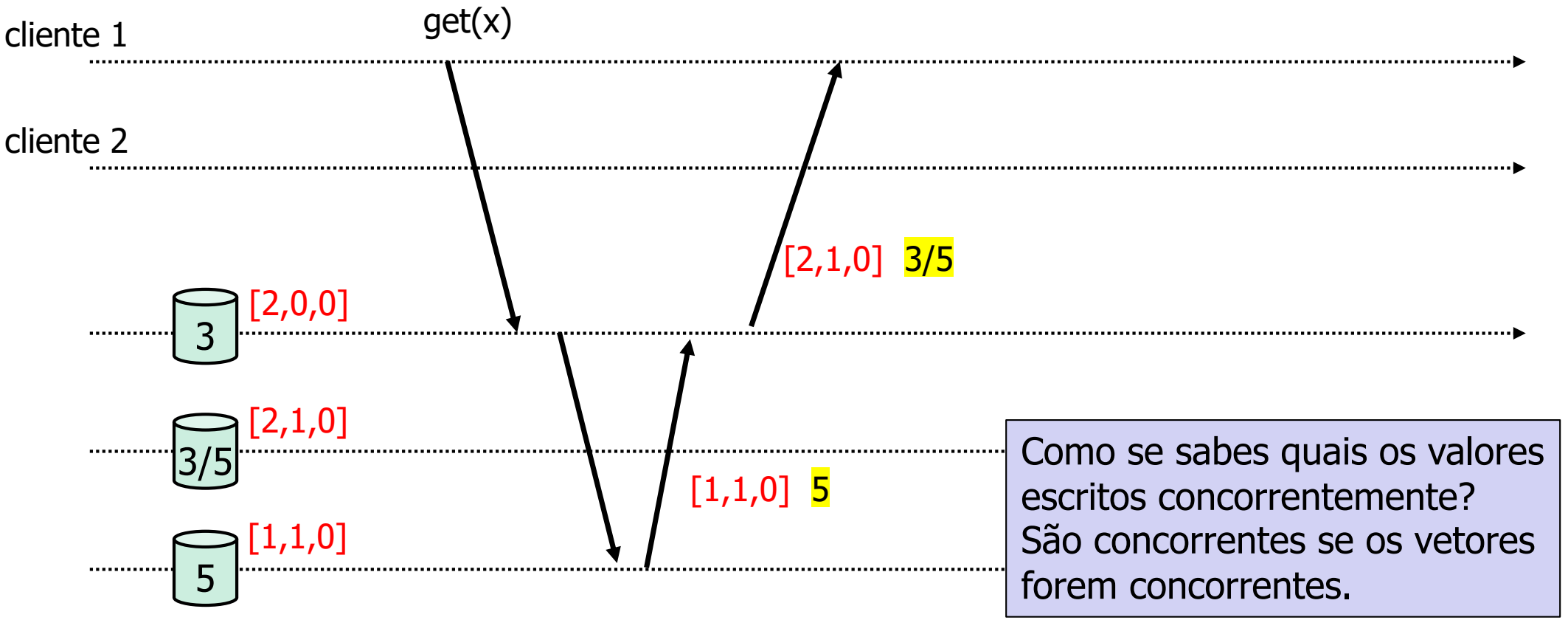
2. Réplica contacta R-1 réplicas



DYNAMO: ESCRITAS CONCORRENTES

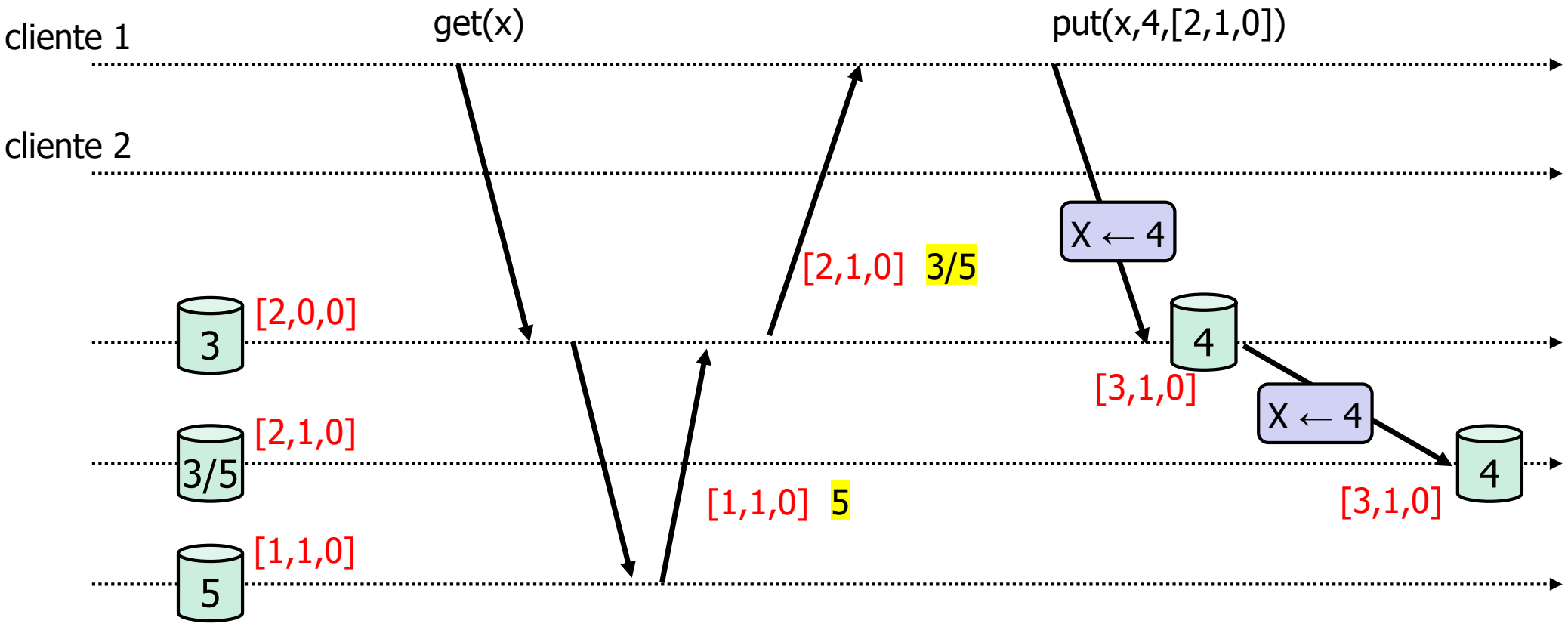
Como é que a leitura devolve escritas concorrentes?

3. Réplica devolve lista de valores escritos concorrentes e vetor versão que unifica vetor versão das réplicas contactadas.



DYNAMO: RESOLUÇÃO DE CONFLITOS

Cliente pode escrever valor que resolve conflito.



PROBLEMA: RECONCILIAR DIVERGÊNCIA

Problema: como lidar com escritas que executaram de forma concorrente?

- E.g. dado um carrinho de compras com {batatas}, o que acontece se um cliente escrever o novo estado: {batatas,cebolas} e outro cliente escrever concorrentemente: {nabos}

PROBLEMA: RECONCILIAR DIVERGÊNCIA

Problema: como lidar com escritas que executaram de forma concorrente?

- E.g. dado um carrinho de compras com {batatas}, o que acontece se um cliente escrever o novo estado: {batatas,cebolas} e outro cliente escrever concorrentemente: {nabos}

Dynamo: cliente responsável por resolver conflito

Artigo do Dynamo sugere que se faça a união dos carrinhos de compras -> {batatas,cebolas,nabos}

PROBLEMA: RECONCILIAR DIVERGÊNCIA

Problema: como lidar com escritas que executaram de forma concorrente?

- E.g. dado um carrinho de compras com {batatas}, o que acontece se um cliente escrever o novo estado: {batatas,cebolas} e outro cliente escrever concorrentemente: {nabos}

Cassandra (clone Dynamo open-source): "last-writer-wins"

Resultado seria {batatas,cebolas} ou {nabos}, dependendo da estampilha da escrita.

PROBLEMA: RECONCILIAR DIVERGÊNCIA

Problema: como lidar com escritas que executaram de forma concorrente?

- E.g. dado um carrinho de compras com {batatas}, o que acontece se um cliente escrever o novo estado: {batatas,cebolas} e outro cliente escrever concorrentemente: {nabos}

CRDTs (conflict-free replicated data types): tipos de dados com política que unifica atualizações e inclui política de resolução de conflitos pré-definida

Resultado seria {cebolas,nabos}, no pressuposto que o primeiro cliente adicionou cebolas e o sergundo retirou as batatas.