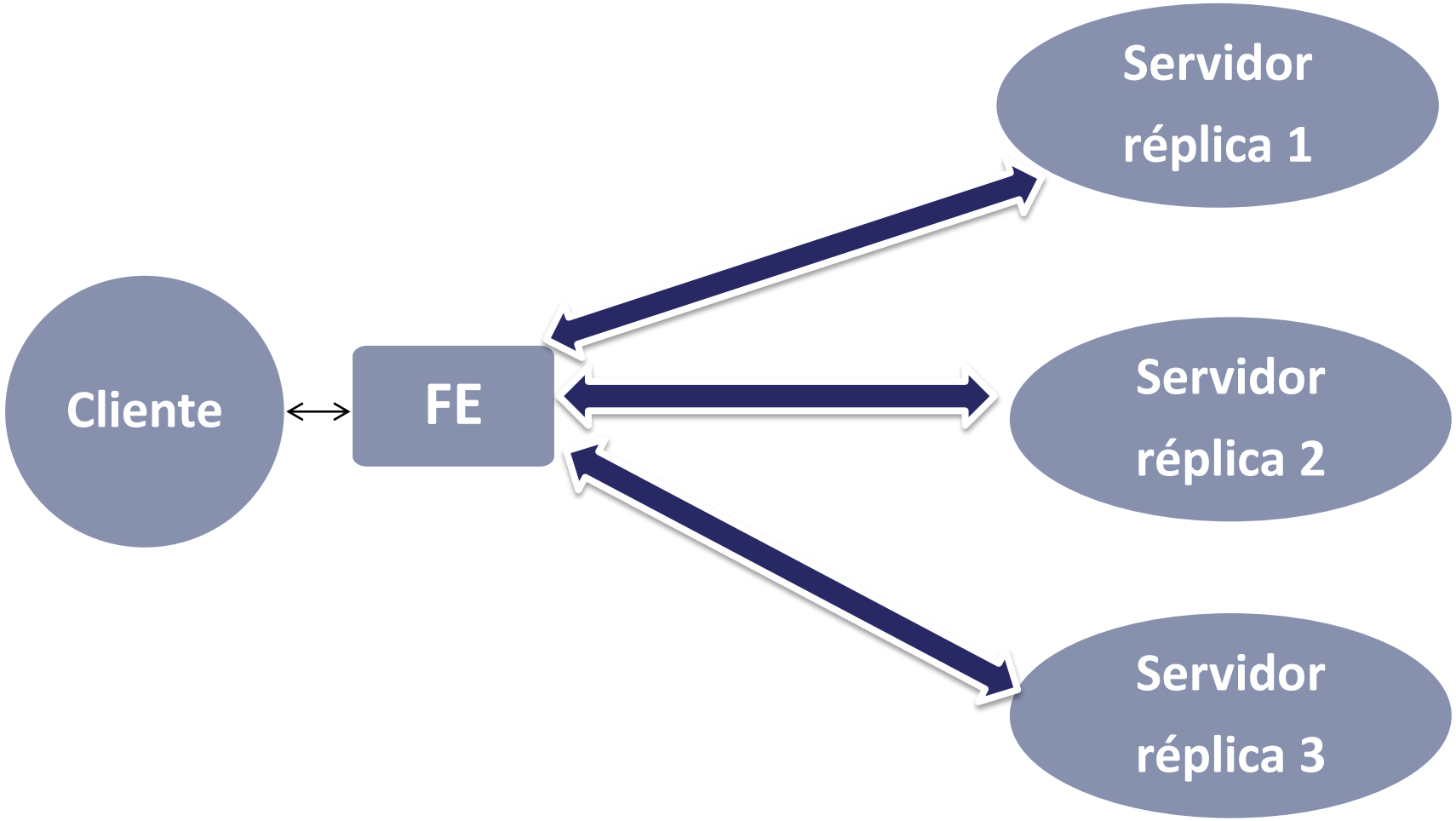
Replicação Ativa

Replicação Ativa



Protocolos de replicação ativa

- As réplicas são todas idênticas e executam em paralelo o mesmo serviço como máquinas de estado determinísticas
- *Front-End* (FE) do cliente envia mensagem aos gestores de réplica
- Cada gestor executa a mensagem e envia a resposta
- O FE espera por um conjunto de respostas e retorna uma delas

Por quantas respostas precisa o FE esperar?

Se chegarem respostas diferentes, qual escolher?

Tentemos construir um protocolo de replicação ativa

Simplificação: Interface leitura/escrita de registo

- Sistema replica apenas um registo
  - Por exemplo, um inteiro
- Suporta apenas duas operações
  - Leitura do registo replicado
    ```
    val = read();
    ```
  - Escrita no registo replicado
    ```
    ack = write(new_val);
    ```
- Queremos garantir consistência sequencial

Relembrar:
Consistência sequencial

Um sistema replicado diz-se **sequencialmente consistente** sse:

- Existe uma **serialização virtual** que:
  - É **correta** segundo a especificação dos objetos, e
  - Respeita ~~o **tempo real**~~ a **ordem do programa de cada cliente**
- A execução observada por cada cliente é consistente com essa serialização virtual (para todos os clientes)

- Se os pressupostos forem
  - Nós com falha silenciosa
  - Rede síncrona e sem falhas
- O sistema seria parecido ao *primary-backup* com o tempo de recuperação praticamente nulo
  - Grau de replicação: f+1
  - Tempo de resposta: ideal
  - Tempo de recuperação: **nulo**

Modelo de faltas que assumiremos

- Sistema assíncrono, em que:
  - **Mensagens não se perdem,**
    mas podem chegar arbitrariamente atrasadas
  - **Não há garantia de receção FIFO**

A alteração deste pressuposto em relação
ao *primary-backup* tem enorme importância

Uma solução hipotética:

Protocolo *write-all-available*

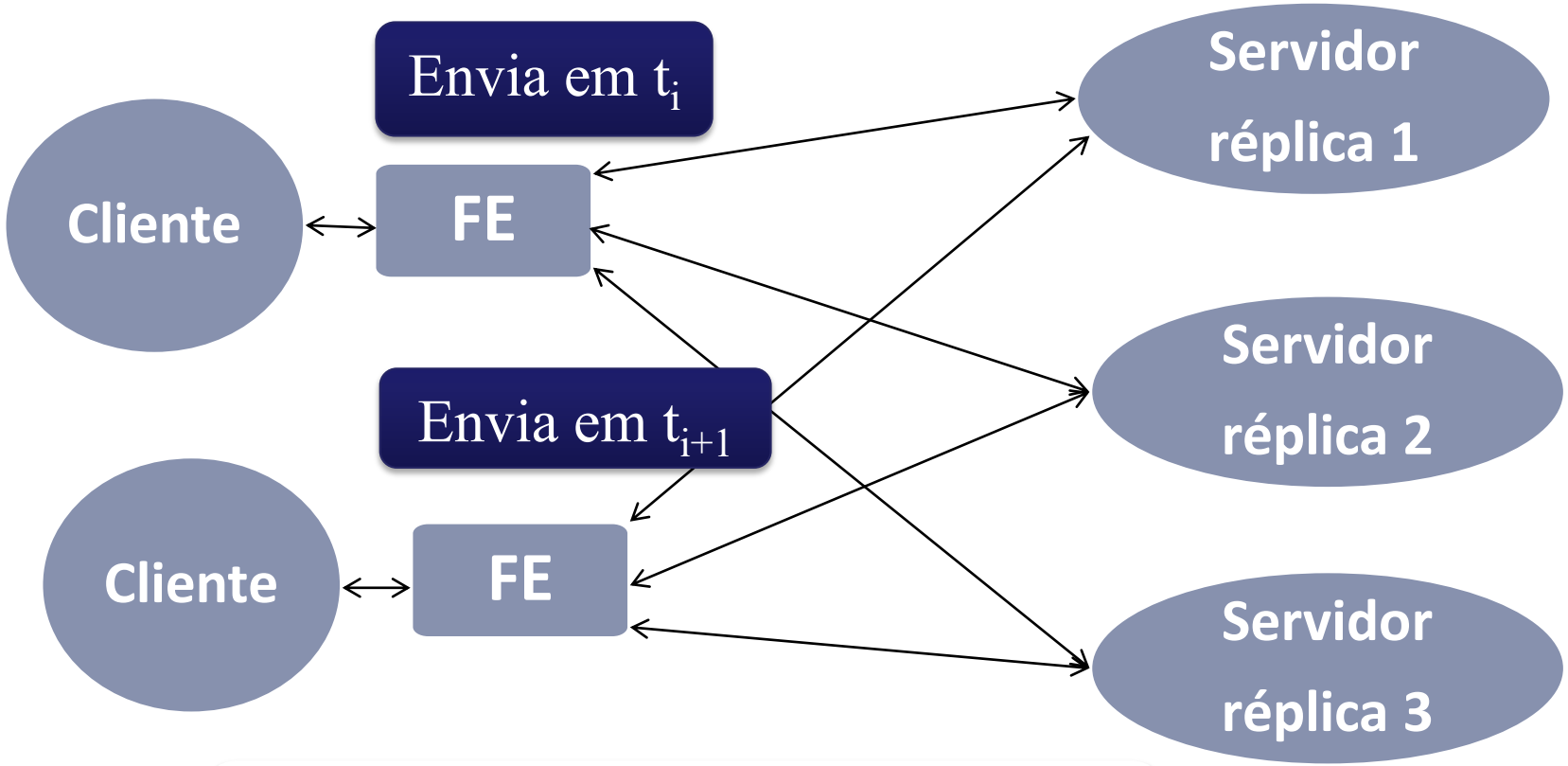
Protocolo *write-all-available*

- Para escrever, o *Front-End* (FE):
  - Envia pedido de escrita para todas as réplicas
  - Cada réplica que recebe o pedido,
    escreve o novo valor sobre o seu registo local e responde “ack”
  - Quando receber “ack” de pelo menos uma réplica,
    FE dá a escrita como terminada
- Para ler, o FE:
  - Envia pedido de leitura para todas as réplicas
  - Cada réplica que recebe o pedido responde com
    o valor atual do registo local
  - Cliente **espera pela primeira resposta** e retorna-a

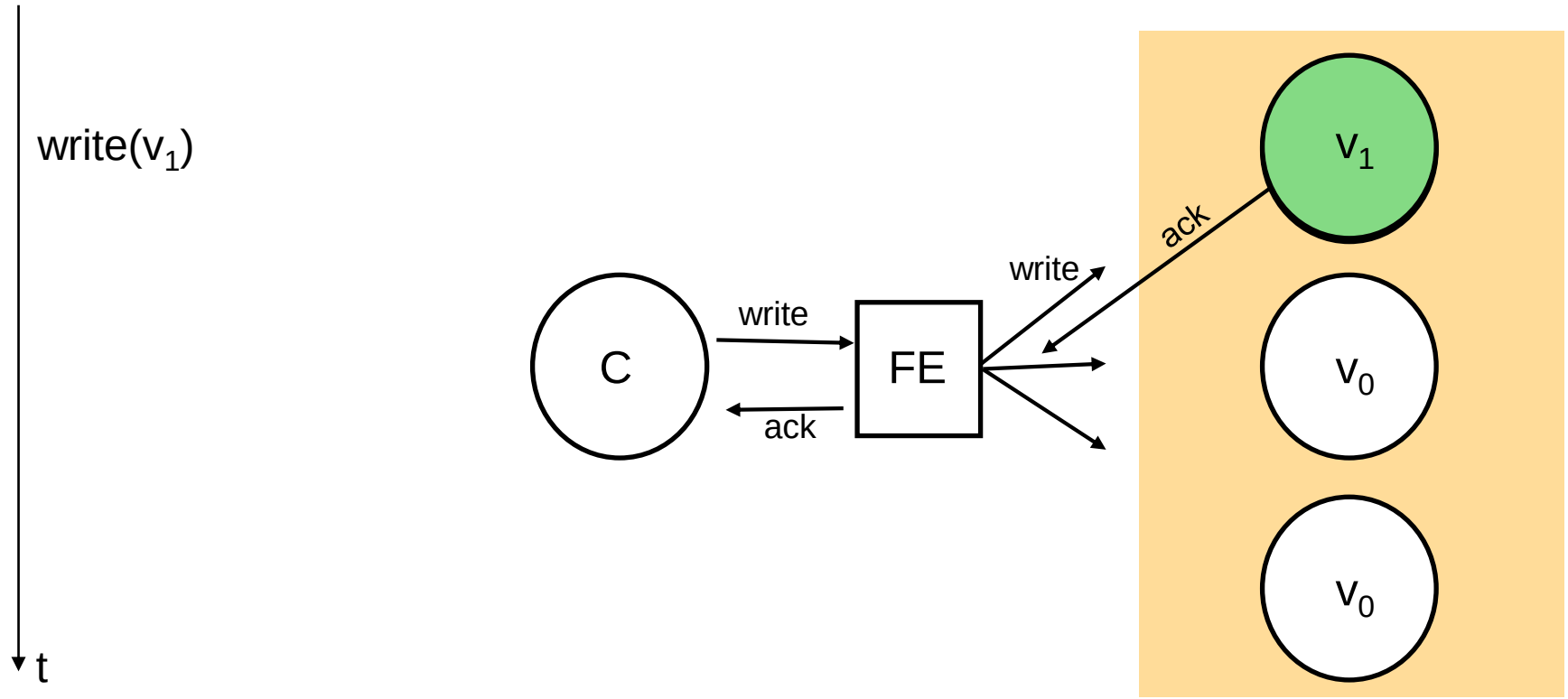
Write-all-available: vantagens aparentes

- Grau de replicação ótimo:
  - f+1 réplicas toleram f falhas
    - Tanto para efetuar escritas como leituras
- Operações muito rápidas
  - Basta receber a primeira resposta e FE retorna
  - Tipicamente, a primeira resposta chega da réplica mais próxima do cliente e/ou menos sobrecarregada
    - Um caso particularmente interessante é quando existe uma réplica na mesma máquina do cliente

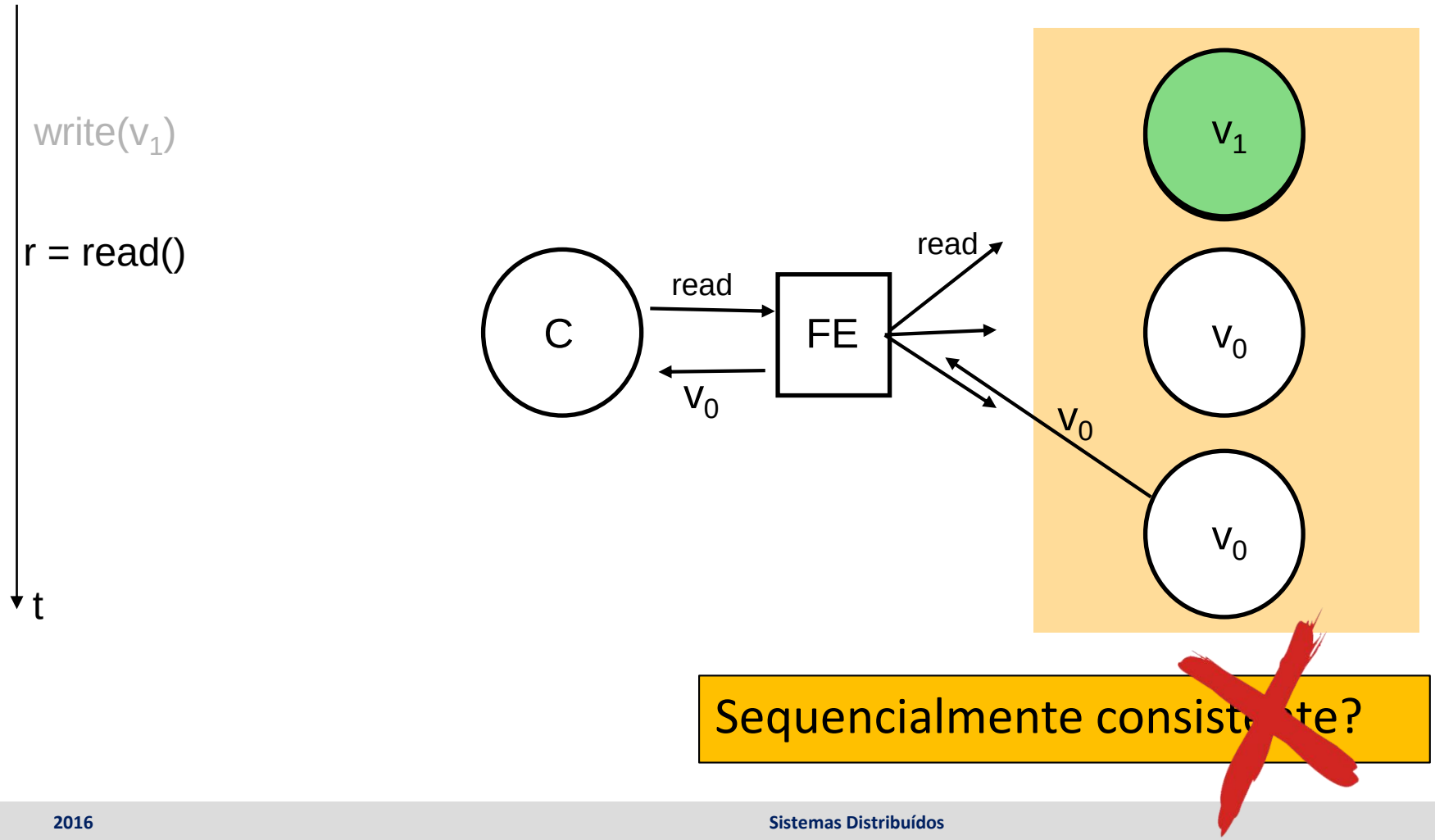
O problema da Ordem



Write-all-available parece ter grau de replicação ótimo mas…



Write-all-available parece ter grau de replicação ótimo mas…



Write-all-available parece ter grau de replicação ótimo mas…

- O que pode acontecer caso uma réplica não receba um pedido de escrita e depois responda a leitura?
  - Em caso de mensagem **atrasada ou falha temporária da réplica**

Réplica pode levar FE a retornar leitura inconsistente!

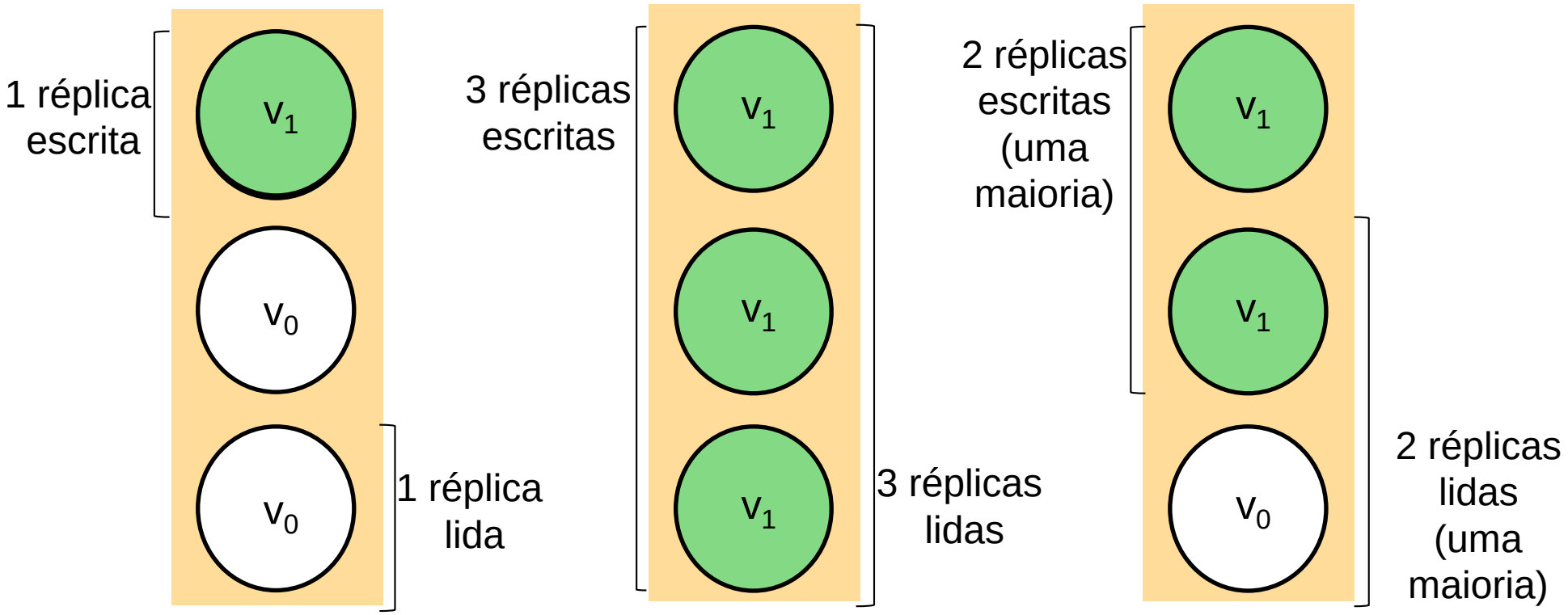
  - Mesmo quando não há falhas, o que pode correr mal?

Se houver múltiplos pedidos de escrita concorrentes,
réplicas podem ficar inconsistentes e
leituras posteriores retornam valores diferentes!

Uma melhor solução:

Protocolo *Quorum Consensus*

De quantas réplicas ler/escrever?



Não é seq. consistente

Não tolera falhas

Protocolo Quorum Consensus

- Sistema de Quóruns:
  - Conjunto de subconjuntos das réplicas,
    tal que quaisquer dois subconjuntos se intersetam
  - Por exemplo: N réplicas → Quórum: qualquer **maioria**: |Q|>N/2
- Cada réplica guarda:
  - Valor do objeto (*registo*)
  - Respetivo *timestamp*

Modelo de faltas:
Nós – falta por paragem
Rede – atrasa ou omite mensagens

Grau de replicação: 2f+1

Protocolo Quorum Consensus
Réplicas

- Cada réplica guarda:
  - *val* - valor do objeto (registo)
  - *tag* - que identifica a versão, composta por:
    - *seq* (número de sequência da escrita que deu origem à versão)
    - *cid* (identificador do cliente que escreveu)
- Dizemos que *tag2* é maior que *tag1* sse:
  - seq2 > seq1, ou
  - seq2=seq1 e cid2>cid1

Protocolo Quorum Consensus
Leituras

- *Front-end*:
  - Envia `read()` para todos os gestores de réplica
    - No-máximo-1-vez
  - Aguarda por Q respostas
  - Seja maxVal o valor que recebeu com maior tag
  - Retorna maxVal

- Gestor de réplica:
  - Ao receber `read()`, responde com <val,tag>

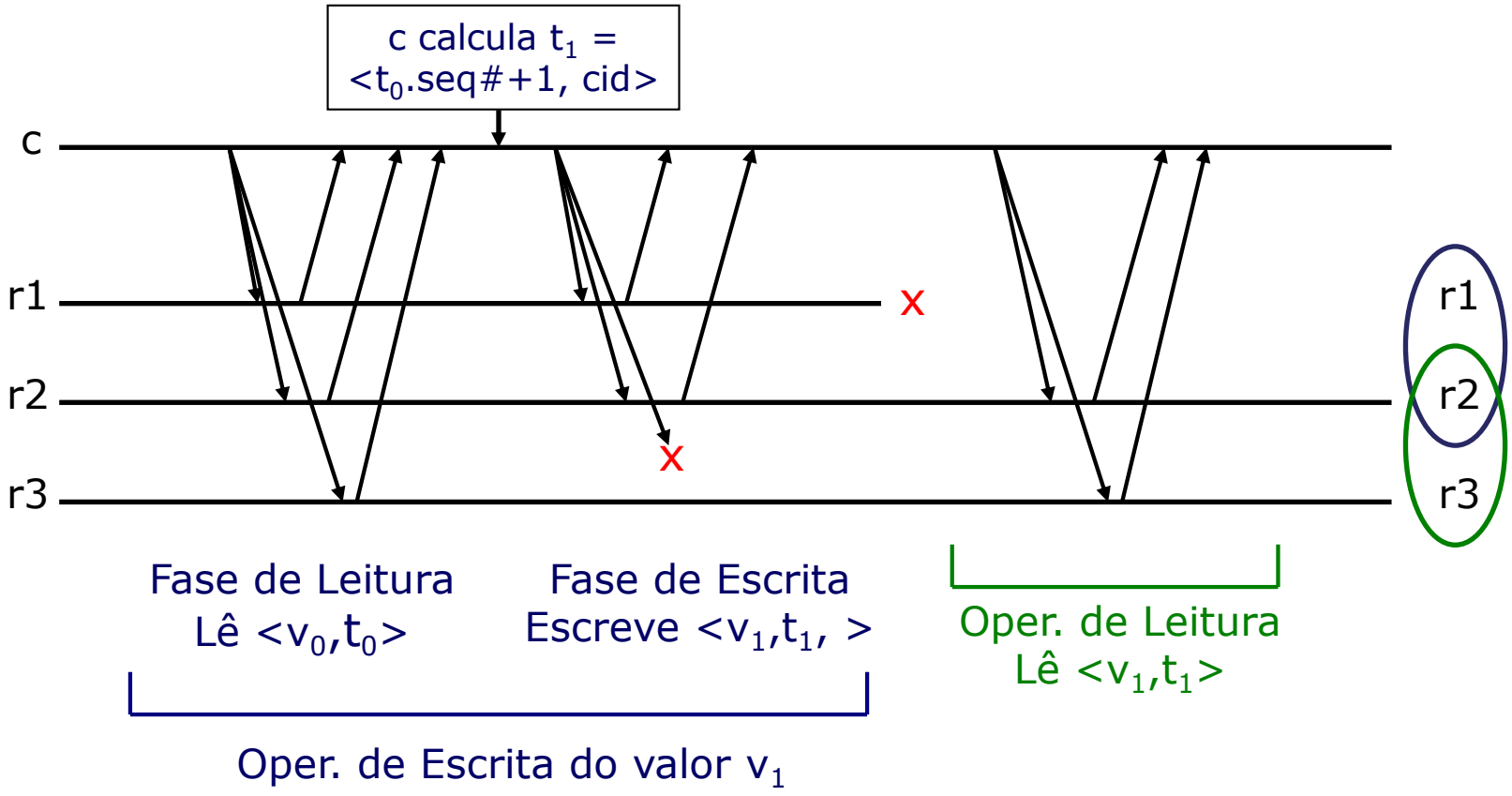
Protocolo Quorum Consensus
Escritas

- Front-end:
  - Executa leitura para obter *maxTag=<seq,cid>*
  - *newTag = <seq+1, mycid>*
  - Envia `write(val, newtag)` a todos os gestores de réplica
  - Espera por Q *acks*
  - Retorna ao cliente

- Gestor de réplica:
  - Ao receber `write(v,t)`:

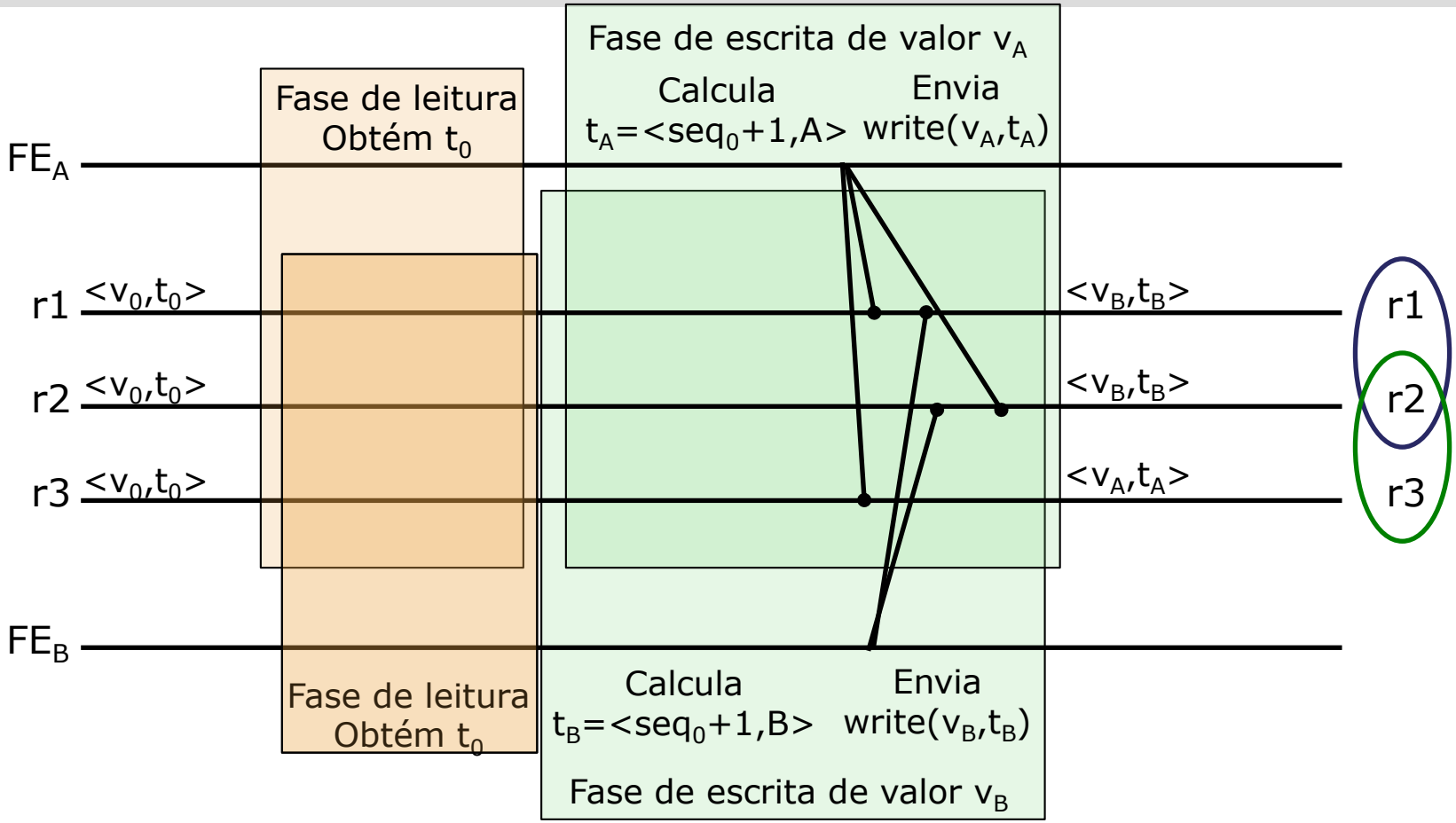
```
se (t > tag) {
  val = v
  tag = t
}
Responde ack
```

Garante consistência sequencial?
Exemplo 1

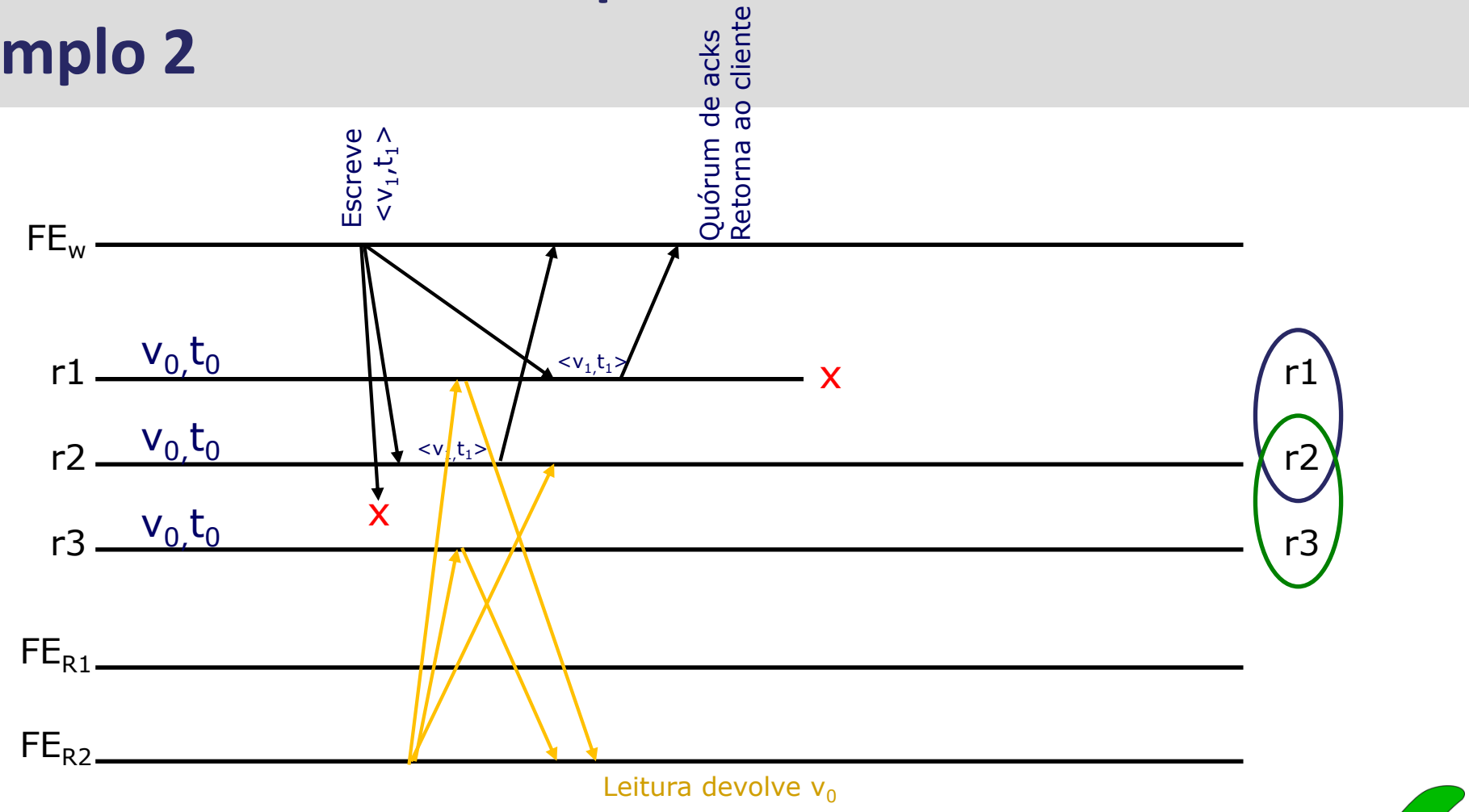


Sequencialmente consistente

Garante consistência sequencial?
Exemplo 2: escritas concorrentes



Garante consistência sequencial?
Exemplo 2

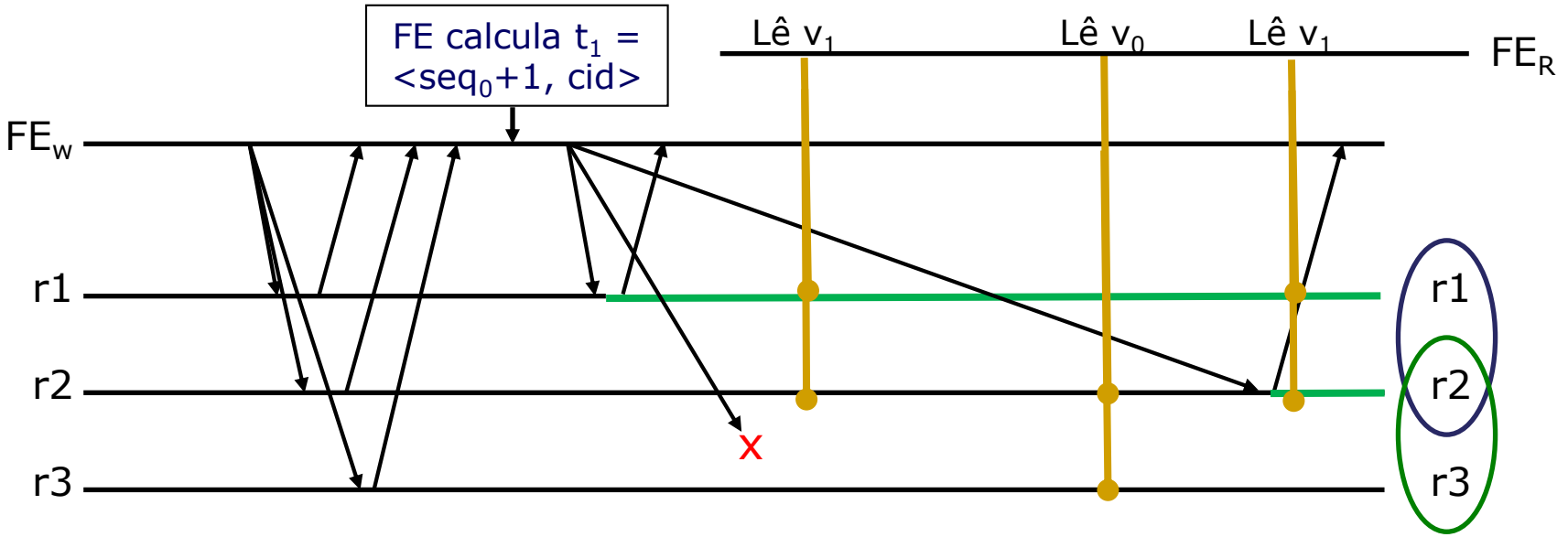


Sequencialmente consistente

Quorum Consensus
Versão múltiplos escritores/leitores

- Problema:
  - Duas escritas concorrentes podem escolher o mesmo *timestamp*
  - Solução: *timestamp* = <Nº seq., client-id>
- *Timestamp* passa a ser <Nº seq., client-id>
  - Assume-se que cada FE tem um client-id único
  - Ao comparar de *timestamps*, comparamos o nºseq; em caso de empate, client-id desempata

Garante consistência sequencial?
Exemplo 3: ler durante escrita incompleta



Sequencialmente consistente?

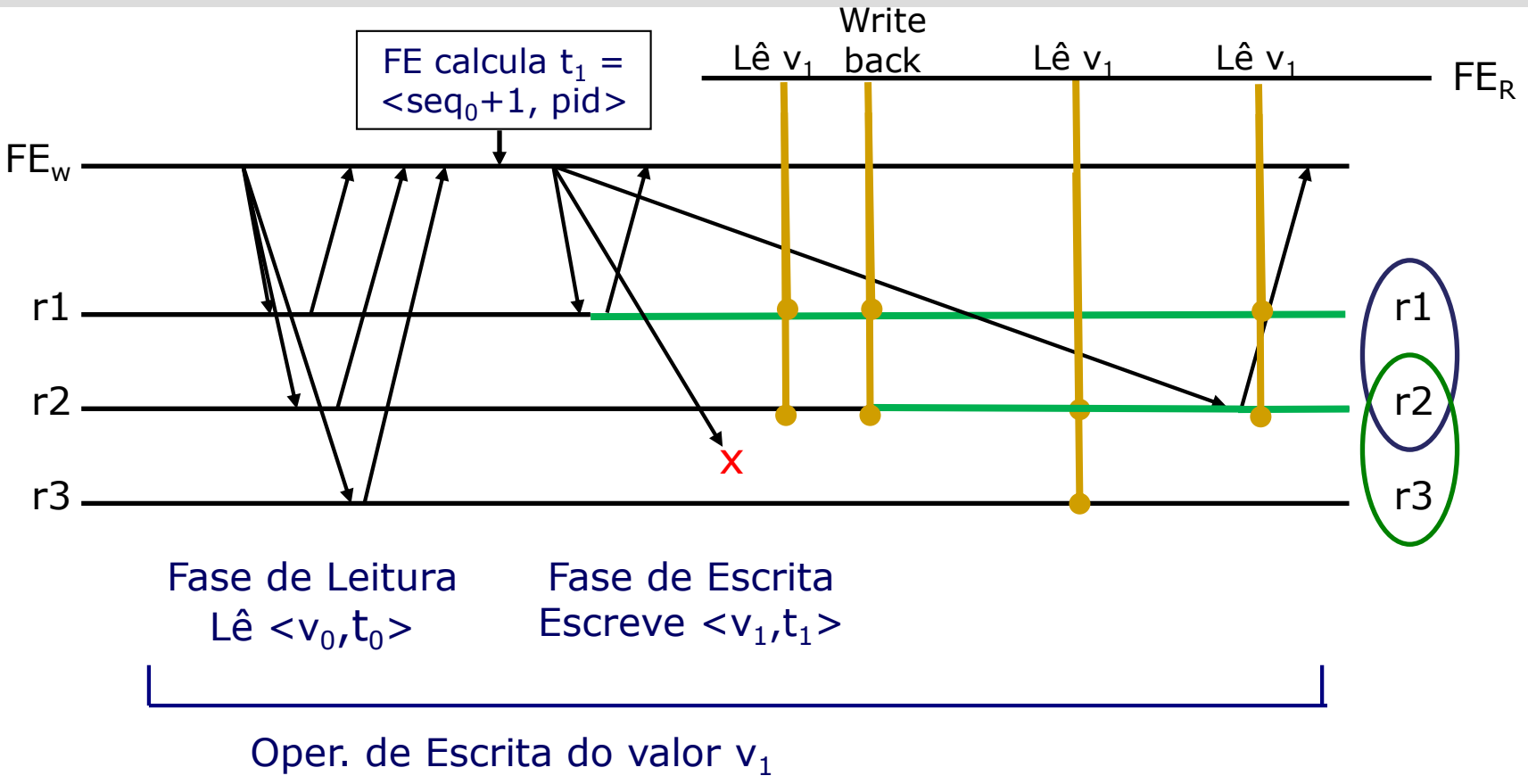
Protocolo Quorum Consensus: variante ABD
Leituras (agora, com *writeback*)

- Frontend:
  - Envia read() para todos os gestores de réplica
    - No-máximo-1-vez
  - Aguarda por Q respostas
  - Seja maxVal o valor que recebeu com maior tag (maxTag)
  - writeback
    - Envia write(maxVal,maxTag) a todos os gestores de replica
    - Espera por Q *acks*
  - Retorna maxVal

- Gestor de réplica:
  - Ao receber read(), responde com <val,tag>

Caso o valor mais recente seja de escrita que não chegou ainda a Q réplicas, assegura que essa escrita chega a Q

Exemplo 3: ler durante escrita incompleta (agora com *writeback*)



Sequencialmente consistente

Variantes ao protocolo: pesos variáveis

- Cada réplica tem um peso não negativo
  - Soma total de pesos é conhecida *a priori*
- Um quórum passa a ser qualquer conjunto de réplicas tal que a soma do peso do quórum é superior a (peso total do sistema)/2
- Interessante porque permite dar maior peso a réplicas mais fiáveis, com melhor conectividade ou maior poder computacional

Variantes ao protocolo: Quóruns de leitura e escrita

- O peso exigido para cada tipo de operação passa a ser distinto:
  - *read threshold* (RT) para leituras
  - *write threshold* (WT) para escritas
- Estes parâmetros têm de assegurar que:
  - 2WT > peso total do sistema, e
  - RT + WT > peso total do sistema
- Interessante porque permite otimizar uma operação, à custa da outra
  - Por exemplo, em sistemas em que as leituras são mais frequentes, podemos ter RT << WT

Quorum Consensus: Vantagens?

- Primeiro protocolo que aprendemos que tolera falhas silenciosas em sistemas assíncronos
- Réplica que falhe temporariamente e recupere está imediatamente pronta para participar
  - Ficará naturalmente atualizada quando receber próximo pedido de escrita

Quorum Consensus: Desvantagens?

- Protocolo “caro”: exige muitas réplicas para tolerar número curto de falhas
  - Com quóruns de maioria, precisamos de 2*f+1 réplicas para tolerar f falhas de réplicas
- Leituras implicam respostas de múltiplas réplicas
  - Em muitos sistemas, leituras são predominantes, logo o ideal seria permitir que leitura retornasse após resposta de uma réplica apenas
  - Uma hipótese para conseguir isso seria definir WT máximo, mas aí deixaríamos de tolerar qualquer falha em escritas

Para além do Quorum Consensus (I)

- Melhor eficiência nos acessos de leitura
  - Combinar técnicas de protocolos anteriores
  - Exemplo: protocolos de *virtual partition*
    - Na operação normal, usa-se *write-all* dentro do conjunto de gestores de réplica
    - *Quorum consensus* usado apenas quando há suspeita de falha de gestor de réplica
      - Usado para que restantes gestores acordem em retirar o gestor suspeito de falha do grupo
      - Grupo chama-se *view* e este caso chama-se *view change*

Para além do Quorum Consensus (II)

- Tolerância a *f* gestores de réplica bizantinos
  - Várias soluções disponíveis, normalmente baseadas em replicação ativa
    - Quóruns maiores, pois é necessário acautelar o pior caso: mesmo que o quórum contenha as *f* réplicas bizantinas, há também réplicas corretas em número suficiente no quórum
    - Mensagens autenticadas, para evitar que réplicas bizantinas enviem mensagens em nome de réplicas corretas

Para além do Quorum Consensus (III)

- Suporte a interfaces genéricas
  - Uso de transações distribuídas
    - Veremos mais à frente no semestre