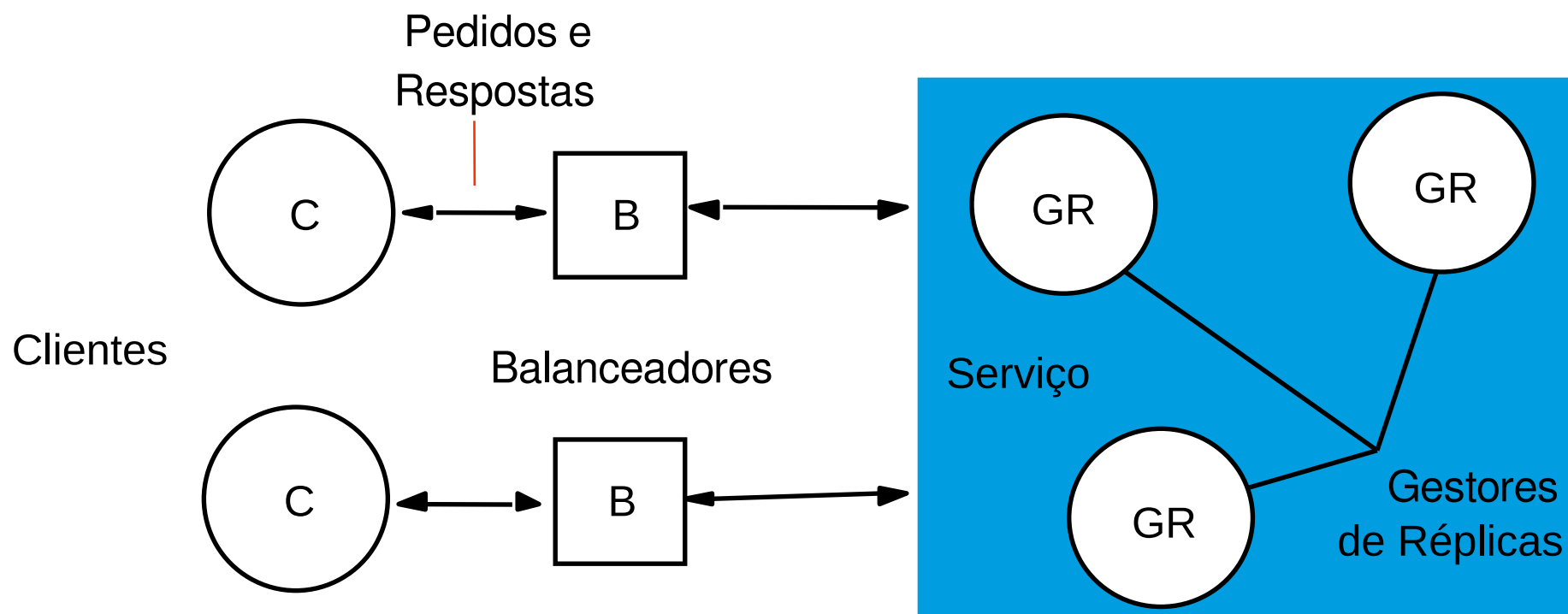


# Breve introdução aos sistemas replicados

# Replicação

- Conceito simples:
  - Manter **cópias** dos dados e do software do serviço em vários computadores



# Replicação: que benefícios?

- Melhor **desempenho e escalabilidade**
  - Clientes podem aceder às cópias mais próximas de si
    - Caso extremo: cópia na própria máquina do cliente (*cache*)
  - Algumas operações podem ser executadas apenas sobre algumas das cópias
    - Distribui-se carga, logo maior escalabilidade
- Melhor **disponibilidade**
  - O sistema mantém-se disponível mesmo quando:
    - Alguns nós falham
    - A rede falha, tornando alguns nós indisponíveis

Para já: discutiremos algoritmos de replicação sem tolerância a faltas => foco no 1º benefício.  
Mais tarde abordaremos sistemas replicados tolerantes a faltas => 2º benefício.

# Coerência

- **Coerência**

- Idealmente, um cliente que leia de uma das réplicas deve sempre ler **valor mais atual**
  - Mesmo que a escrita mais recente tenha sido solicitada sobre outra réplica
- Um critério de coerência: linearizabilidade

## Linearizability: A Correctness Condition for Concurrent Objects

MAURICE P. HERLIHY and JEANNETTE M. WING  
Carnegie Mellon University

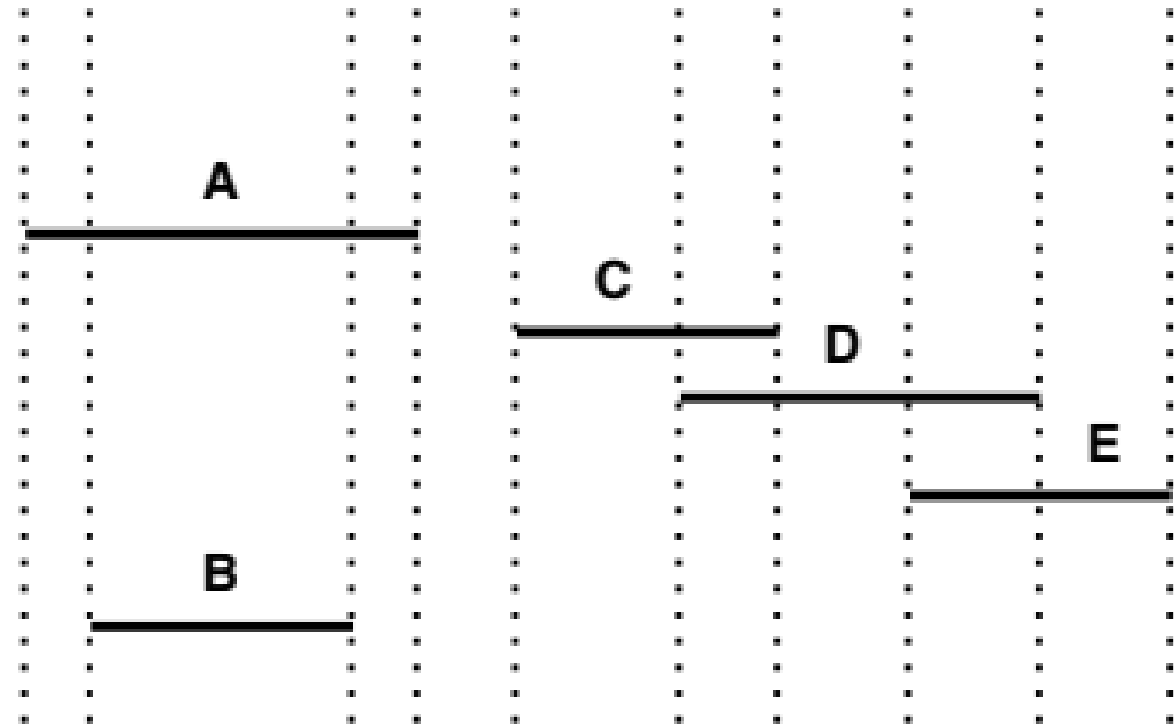
A concurrent object is a data object shared by concurrent processes. Linearizability is a correctness condition for concurrent objects that exploits the semantics of abstract data types. It permits a high degree of concurrency, yet it permits programmers to specify and reason about concurrent objects using techniques from the sequential domain. Linearizability provides the illusion that each process takes effect instantaneously at some point between its operations. This paper compares it to other correctness conditions, and

# Nota prévia sobre ordenar operações

- As operações sobre um sistema replicado não são instantâneas:
  - Uma operação é invocada por um cliente, executada e mais tarde o cliente recebe a resposta
- Se uma operação X começa **depois** de outra operação Y acabar, a operação X ocorre **depois** de Y
- Se uma operação X começa **antes** de outra operação Y acabar, a operação X é **concorrente** com Y

# Ordenar operações

- A concorrente com B
- A anterior a C
- B anterior a C
- C concorrente com D
- C anterior a E
- D concorrente com E



# Linearizabilidade

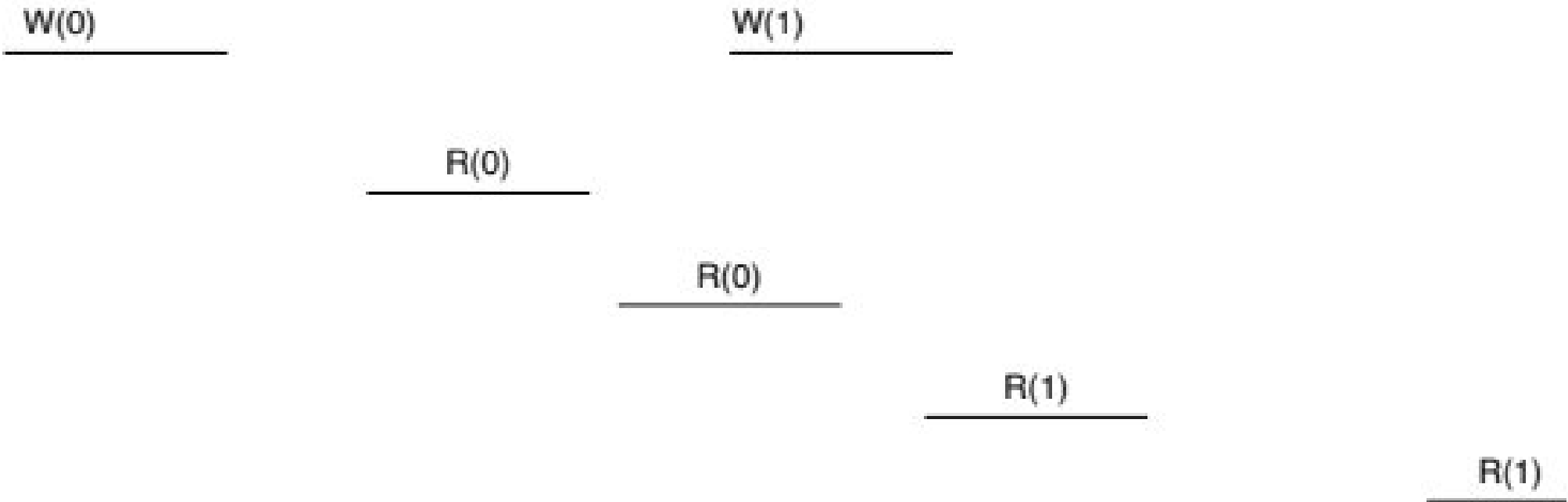
Um sistema replicado diz-se **linearizável** sse (se e só se):

1. Existe uma **serialização virtual** que respeita o **tempo real** em que as operações foram invocadas, isto é:

**Se  $op1$  ocorre antes de  $op2$  (em tempo real), então  $op1$  tem de aparecer antes de  $op2$  na serialização virtual**

- Nota: se  $op1$  e  $op2$  concorrentes, a serialização virtual pode ordená-las arbitrariamente
2. A execução observada por cada cliente é coerente com essa serialização virtual (para todos os clientes):
    - Isto é, os valores retornados pelas leituras feitas por cada cliente refletem as operações anteriores na serialização virtual

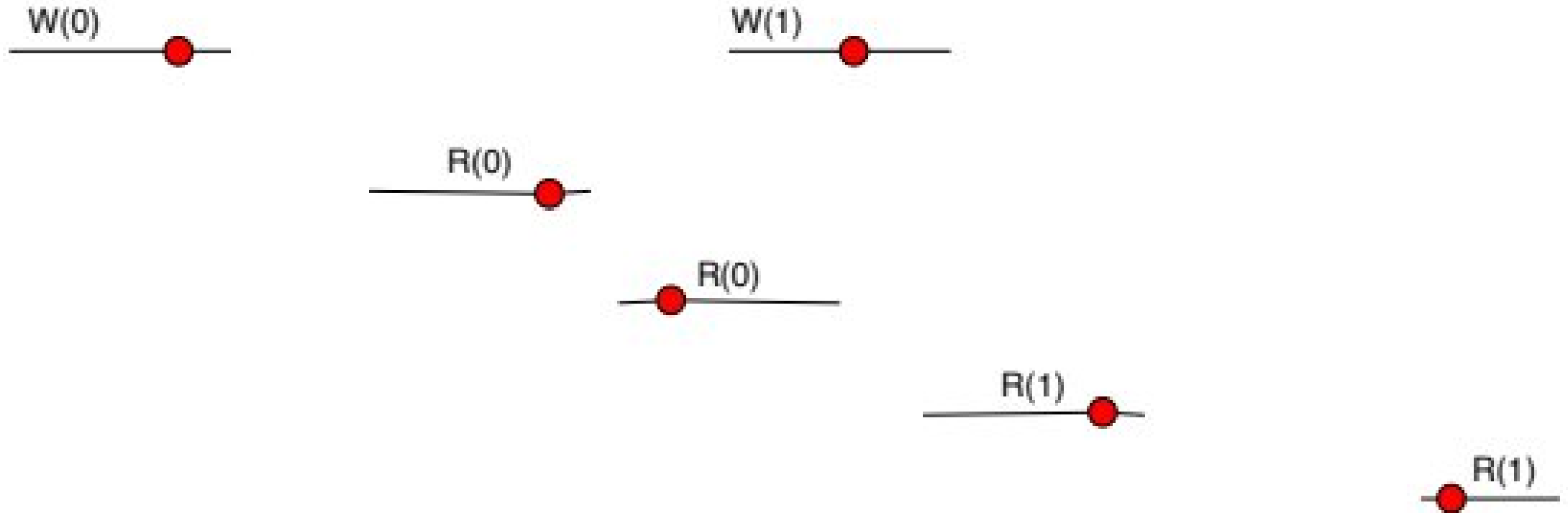
# Exemplo I: clientes que escrevem sobre um inteiro replicado



Esta execução é linearizável?

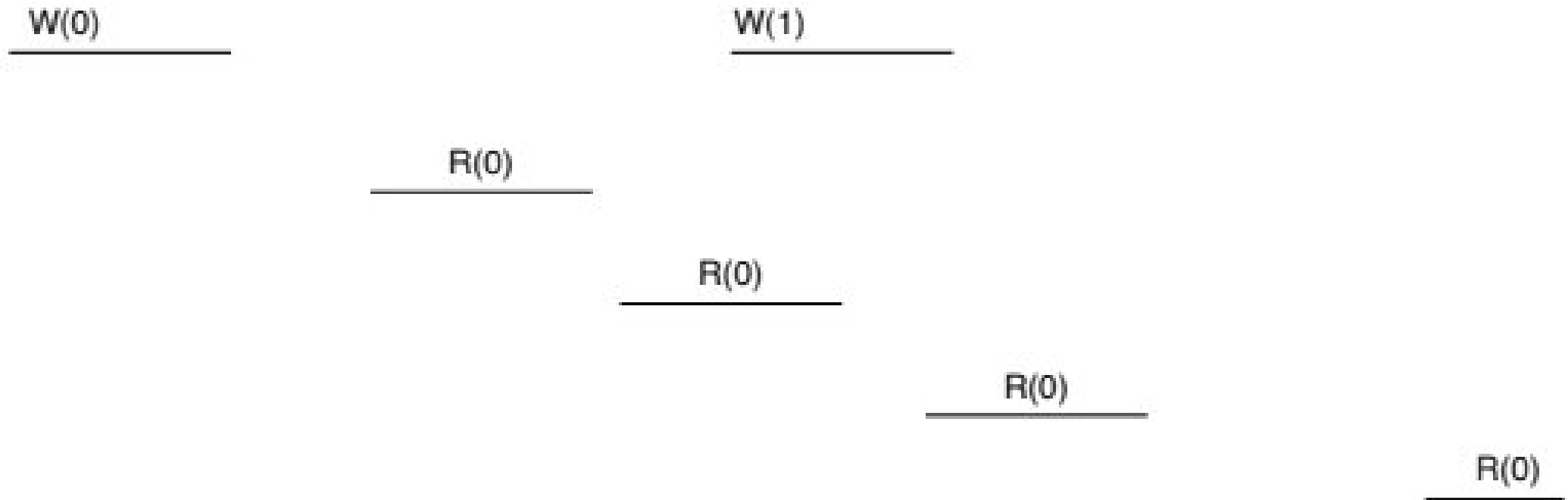


# Exemplo I: clientes que escrevem sobre um inteiro replicado



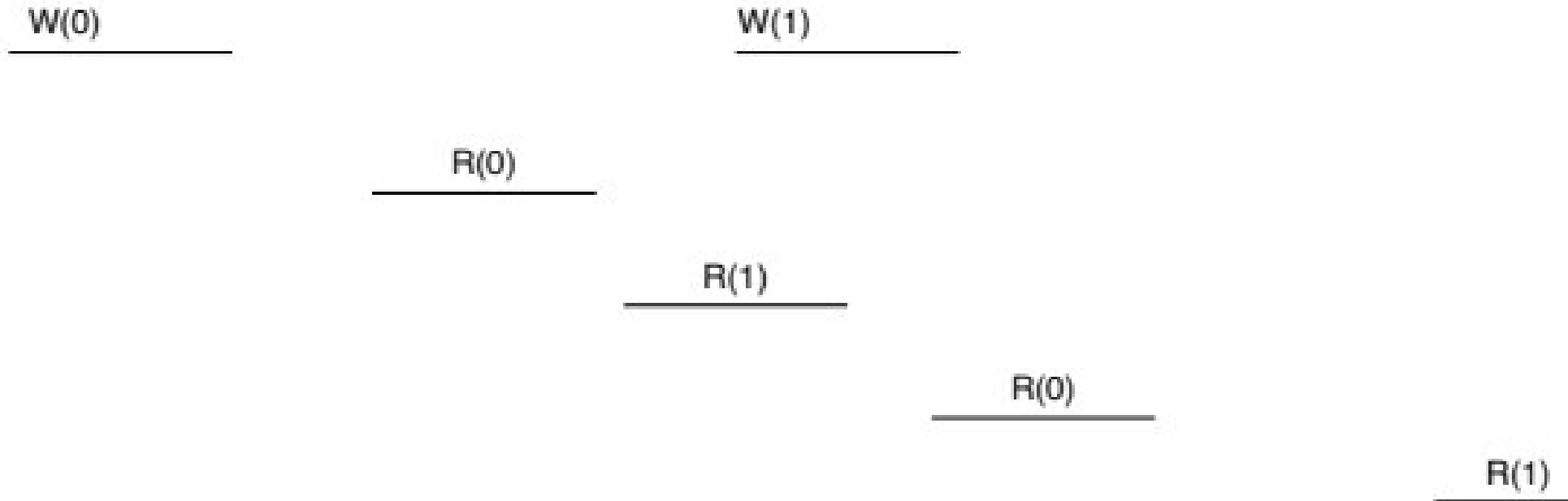
Sim, é linearizável

# Exemplo II



Esta execução é linearizável?

# Exemplo III



Esta execução é linearizável?

Esta aula: como podemos replicar **estruturas de dados específicas**, registo e espaço de tuplos?

Mais tarde, estudaremos como replicar objetos mais **genéricos**

# Registos partilhados (e replicados)

# Registos

- Como criar a ilusão de que existe um registo partilhado?
- Duas dimensões:
  - Qual é o comportamento esperado de um registo partilhado?
  - Que algoritmo usado para assegurar esse comportamento?

# Registos

- Duas operações:
  - Escrita
  - Leitura
- Uma nova escrita substitui o valor da escrita anterior:
  - Isto difere do caso em que os objectos aceitam operações arbitrárias (por exemplo, incrementar duas vezes é diferente de incrementar apenas uma vez)
- Múltiplos clientes podem ler do registo, mas só um cliente pode escrever:
  - Ou seja, as escritas são totalmente ordenadas
  - *Há algoritmos que suportam múltiplos escritores, mas ficam fora da cadeira*

# Registos

- Lamport definiu três modelos de coerência para registos:

- Atomic

Equivalente a linearizabilidade quando aplicada a registos

- Regular

- Safe (*menos relevante em SD, logo não estudaremos este*)



# Registos

8

---

## On Interprocess Communication

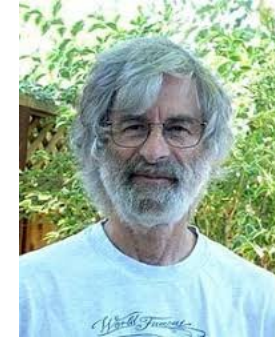
---

Leslie Lamport

---

December 25, 1985

---



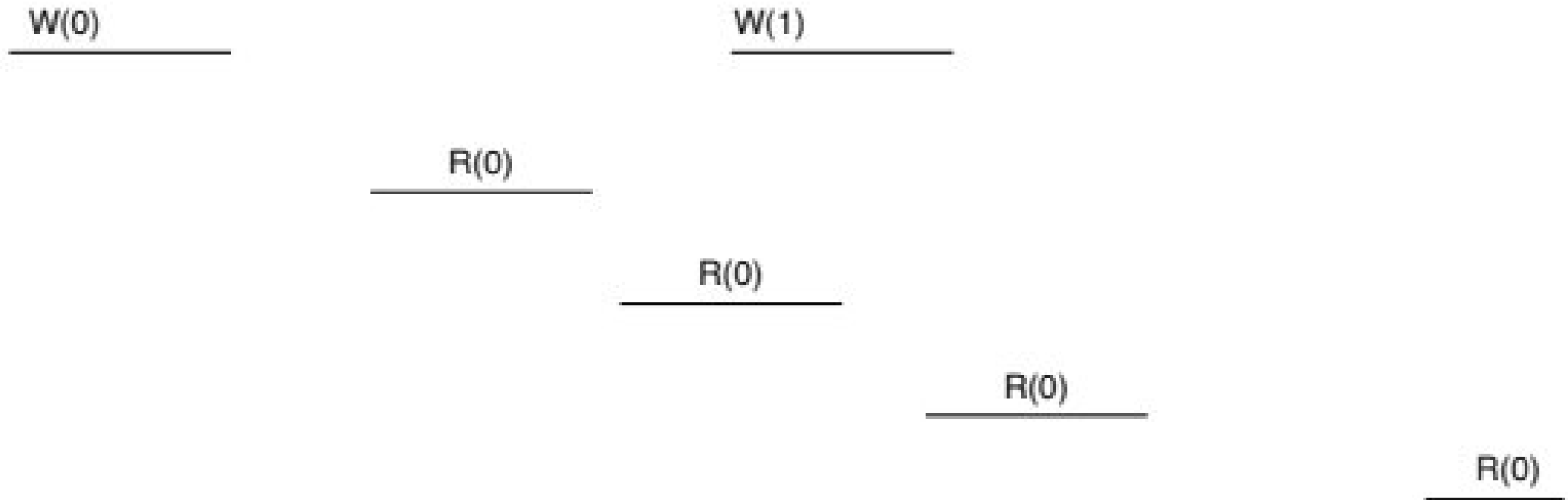
# Registos “Atomic”

- Equivalente a linearizabilidade quando aplicada a registos.
- *O resultado da execução é equivalente ao resultado de uma execução em que todas as escritas e leituras ocorrem instantaneamente num ponto entre o início e o fim da operação.*

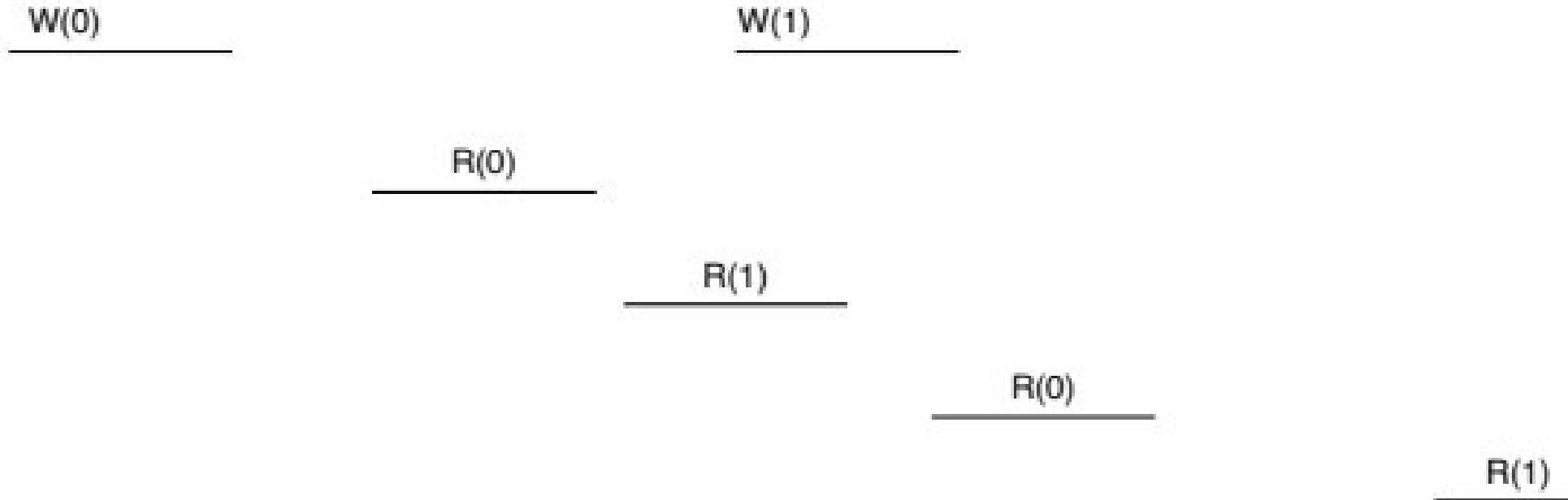
# Registos “Regular”

- Se uma leitura não for concorrente com uma escrita, lê o último valor escrito.
- Se uma leitura for conconcorrente com uma escrita ou retorna o valor anterior ou o valor que está a ser escrito
- Porque não é tão forte como registo atómico (linearizável)?
- R: Enquanto uma escrita está a decorrer, permite que leituras seguidas (uma a seguir à outra) leiam uma sequência incoerente de valores (primeiro ler o novo valor, depois ler o valor antigo)

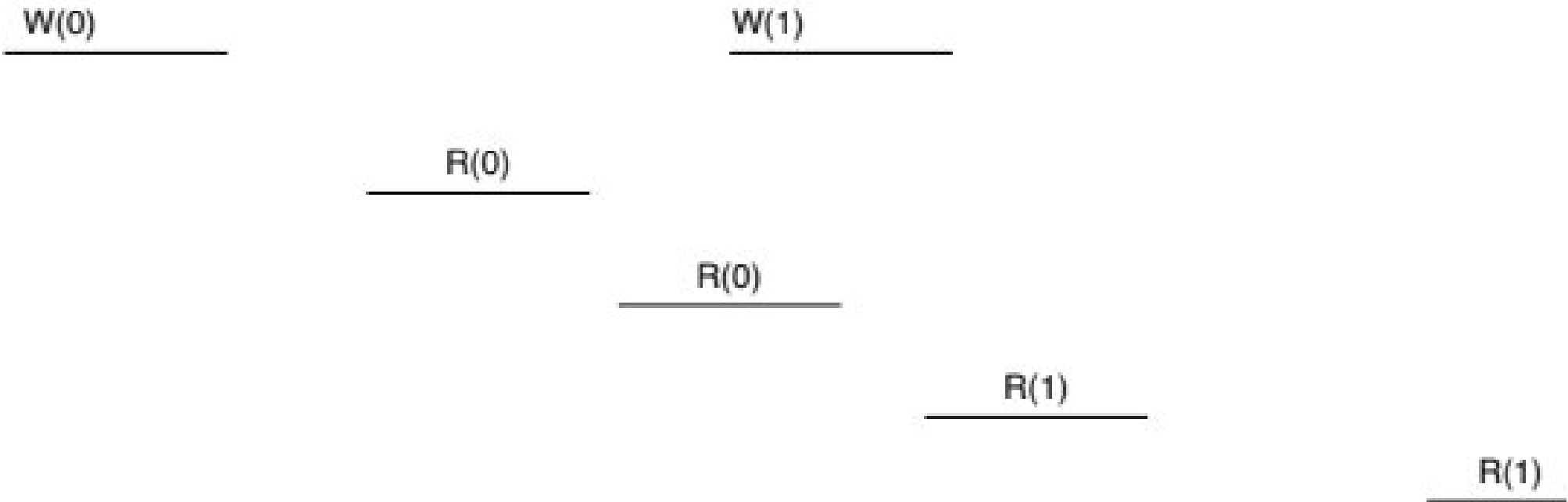
# Registo “unsafe”



# Registo “Regular” (mas não “atomic”)



# Registo “Atomic”

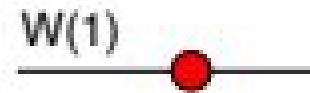


# Registo “Atomic”

W(0)



W(1)



R(0)



R(0)



R(1)



R(1)



# Como concretizar registos distribuídos?

- Cada processo mantém uma cópia do registo
- Cada registo mantém um tuplo **<valor, versão>**
- Para executar uma escrita ou uma leitura, cada processo troca mensagens com os outros processos
- É possível fazer isto de forma tolerante a faltas e não bloqueante!



# ABD



## Sharing Memory Robustly in Message-Passing Systems

Hagit Attiya<sup>1</sup>

Amotz Bar-Noy<sup>2</sup>

Danny Dolev<sup>3</sup>

February 16, 1990

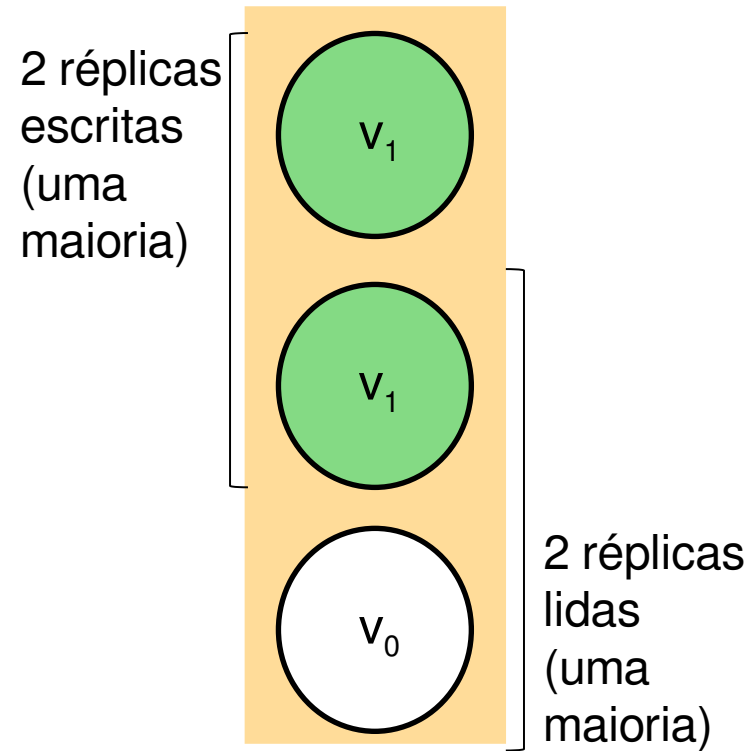
# Registo Regular (só com um escritor)

- Escrita:
  - O escritor incrementa o número de versão e envia o tuplo **<valor, versão>** para todos os processos.
  - Ao receber esta mensagem, os outros processos actualizam a sua cópia do registo (se a versão for superior à que possuem) e enviam uma confirmação o escritor.
  - A operação de escrita considera-se terminada quando o escritor receber resposta de uma maioria

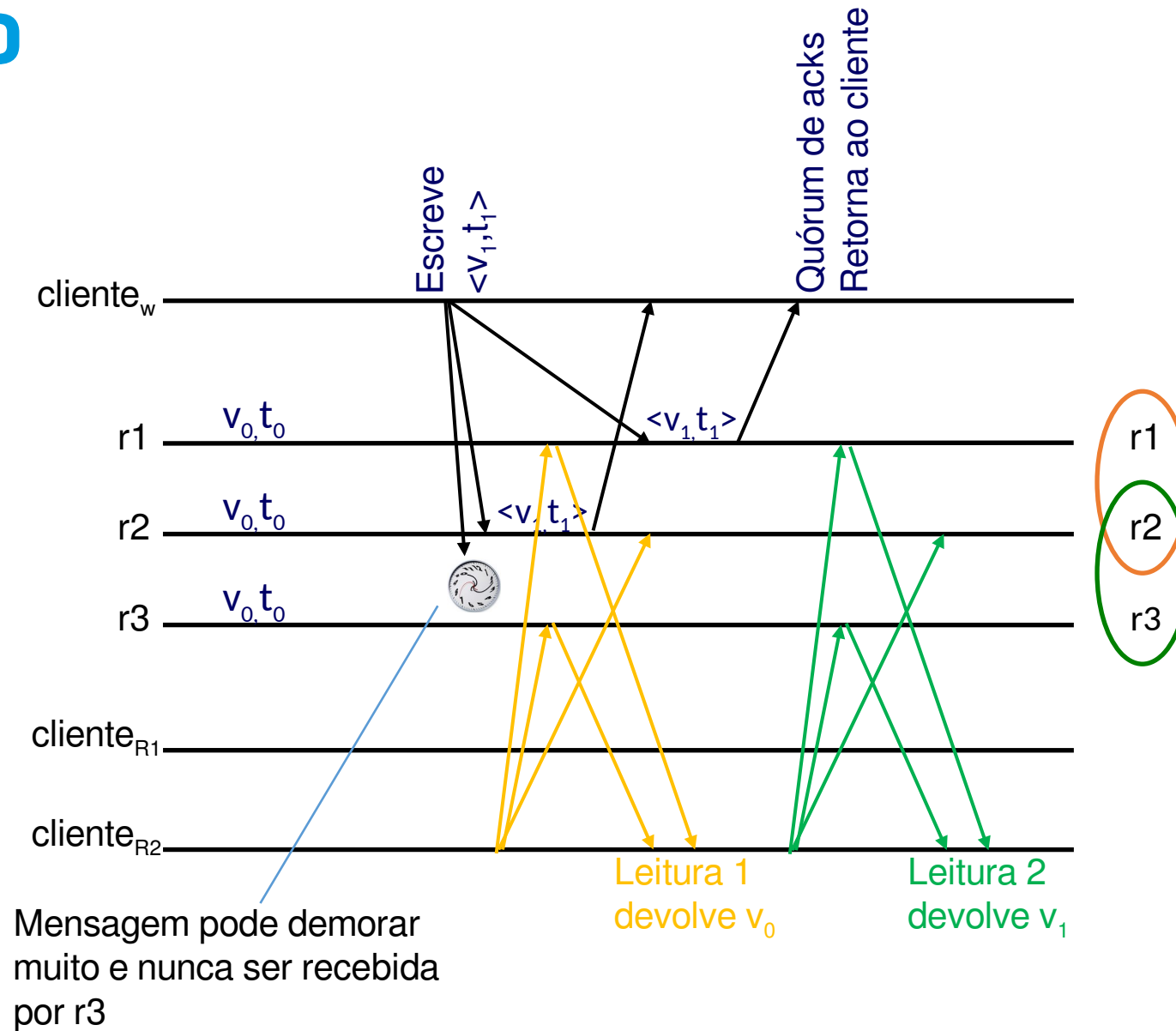
# Registo Regular (só com um escritor)

- Leitura:
  - O leitor envia uma mensagem a todos os processos solicitando o tuplo mais recente
  - Cada processo envia o seu tuplo **<valor, versão>**
  - Após receber resposta de uma maioria, o leitor retorna o valor com a versão mais recente (e actualiza o seu próprio tuplo, caso necessário)

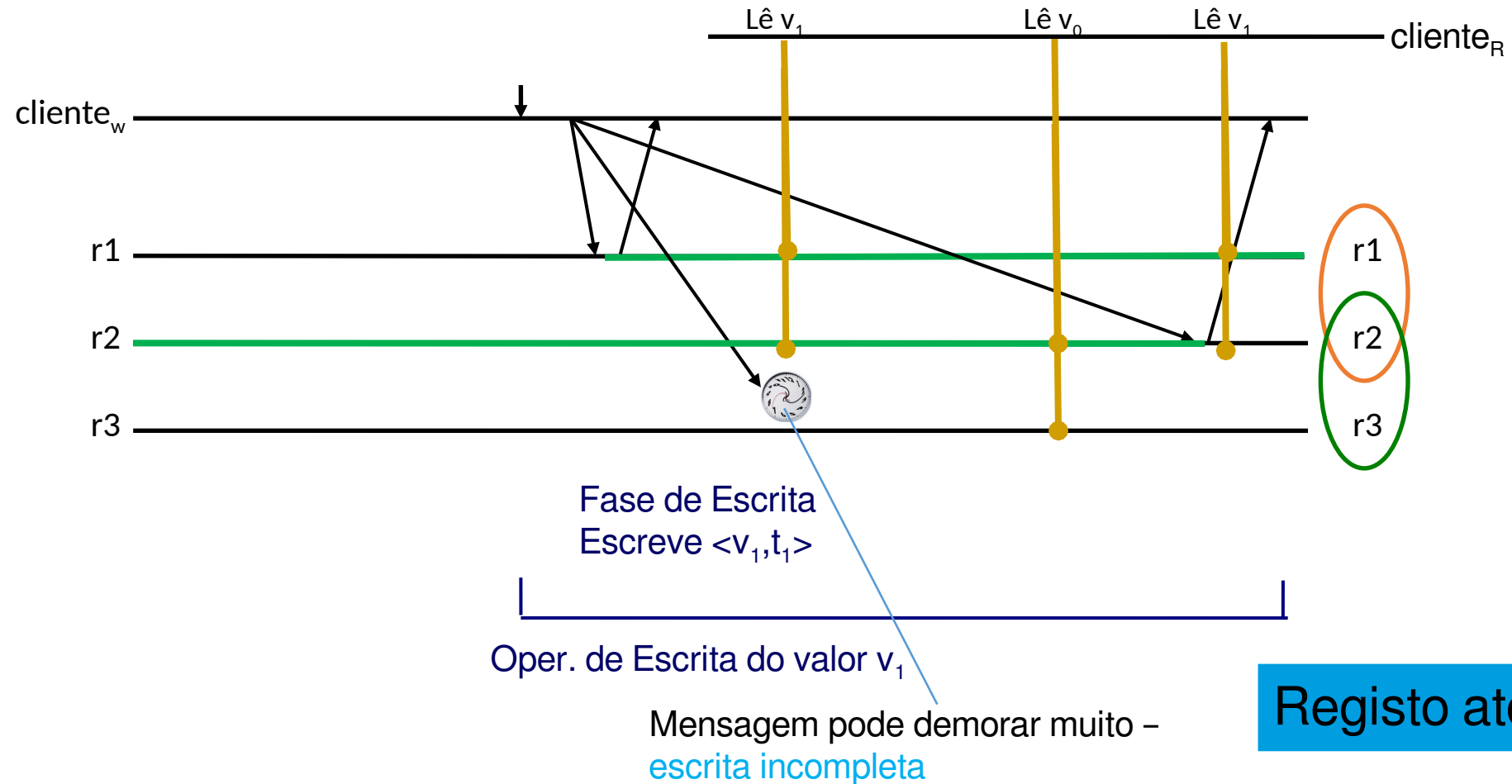
# Intuição do algoritmo



# Exemplo



# Outro exemplo



Registo atómico?

# Registo regular => registo atómico?

- É possível concretizar um registo atómico de forma não bloqueante, isto é, sem impedir as leituras durante uma escrita?

# Registo atómico

- Escrita
  - Semelhante ao anterior
- Leitura:
  - Executa o algoritmo de leitura anterior mas não retorna o valor
  - Executa o algoritmo de escrita, usando o valor lido
  - Apenas retorna o valor lido após a escrita ter terminado