Replication and Consistency Models

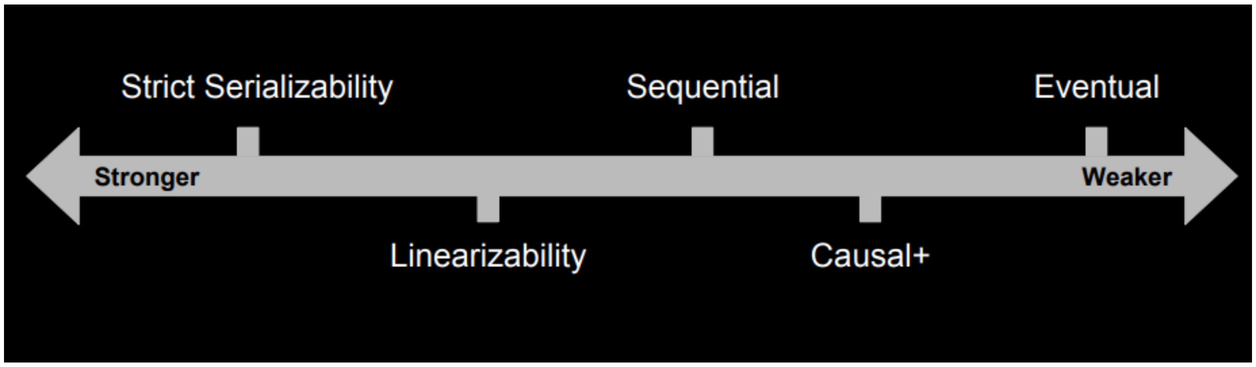
Content

Data replication

The data replication allow us to have:

- **Reliability:** since the data cannot be lost, unless the data is lost in all replicas
- **Availability:** the data is always available, unless the all the replicas are offline.
- **Scalability:** we can have a balance load across nodes

The challenge is to have consistency between replicas.



Strong consistency

Eventual Consistency vs Strong Consistency

Eventual Consistency Eventual consistency is a theoretical guarantee that, provided no new updates to an entity are made, all reads of the entity will eventually return the last updated value. The

https://medium.com/system-design-blog/eventual-consistency-vs-strong-consistency-b4de1f92534d

“Simply put, data has to be locked during the period of update or replication process to ensure that no other processes are updating the same data.”

It has deterministic updates: **same initial state** leads to **same result.**

Sequential consistency model

Sequential consistency in newbie terms?

A program running in a sequentially consistent and distributed environment will behave as if all the instructions are interleaved in a sequential manner. This means multiple execution paths are

https://stackoverflow.com/questions/54493739/sequential-consistency-in-newbie-terms

The operations executed by any thread (in any processor) appear in the order they are executed, in order words, in a sequential order.

Linearizable

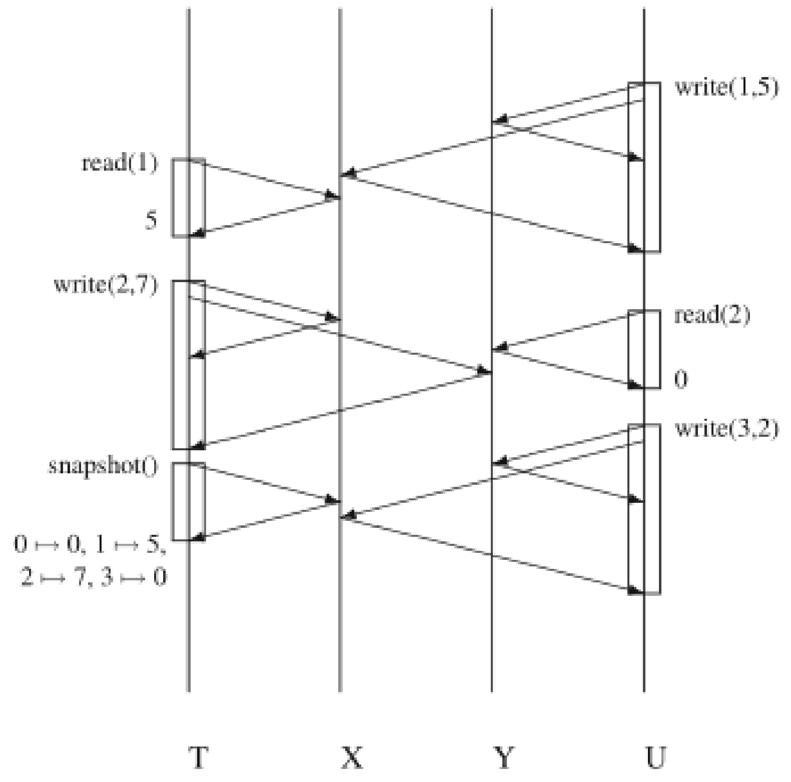
Charron-Bost, Bernadette, Fernando Pedone, and André Schiper, eds. *Replication: Theory and Practice*. Vol. 5959. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. https://doi.org/10.1007/978-3-642-11294-2. [chap 1.3]

We can say that an execution *E* is linearizable if **there is a sequence** the obeys the following rules:

[L1] The linear sequence H contains all the operations E has, each paired wit the return value received in E.
[L2] the real-time total order of operations in H is compatible with the partial order

[L3] H is a legal history of the sequential data type that is replicated.

Now consider that following sequence:



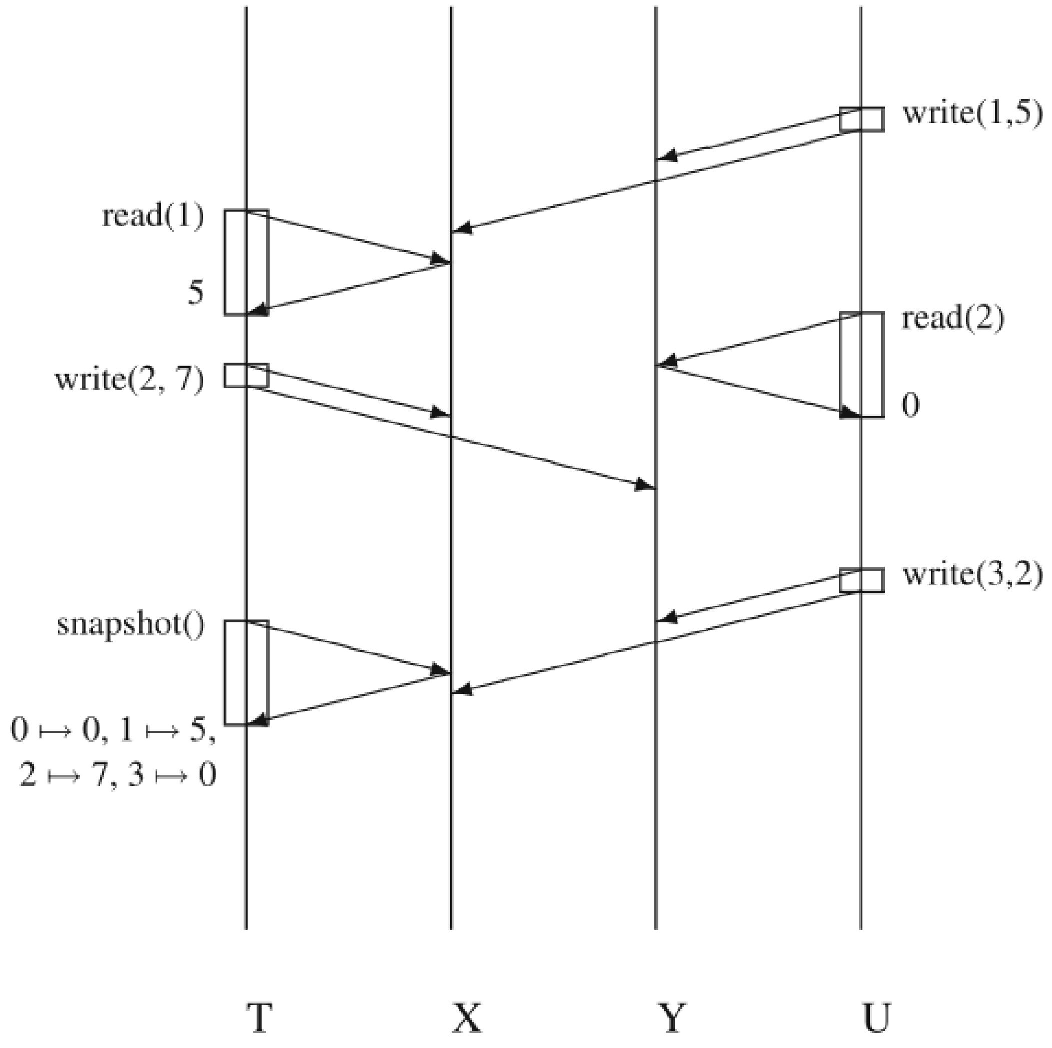
We have that the *write(1)* occurs before *write(2), read(2), snapshot() and write(3).*
Also, *read(1)* occurs before *write(2), read(2), snapshot and write(3).*
read(2) occurs before *snapshot().*

Thus a candidate for a linear order is:

write1, read1, read2, write2, snapshot and write3.

This order obeys the rules [L1], [L2] and [L3].

This system below, however, is **not** linearizable, because at this time the *write* operation might returns before the replicas were actually modified. Thus, there's no guarantee of the linearizability:



Notice that in real-time partial order **write(3)** must be before snapshot(), **because it returns before**. But the result returned by snapshot() is not compatible with this order. In other words, write(3) took place before snapshot(), but the effects were just visible after the snapshot().

TIPS!

https://www.cs.princeton.edu/courses/archive/fall19/cos418/docs/precept8_consistency.pdf

1) A completed write appears to all future reads.

2) Once a read sees a new value, all future reads reads must return the new value

Sequential consistency

Sequential Consistency

Sequential consistency is a strong safety property for concurrent systems. Informally, sequential consistency implies that operations appear to take place in some total order, and that that order is consistent with the order of operations on each individual process.

https://jepsen.io/consistency/models/sequential

Informally, sequential consistency implies that operations appear to take place in some total order, and that that order is consistent with the order of operations on each individual process.

In fact the system above is not linearizable, but it has sequential consistency. Consider $<_{E,rc}$ is the client partial order symbol used. Then, we can say in $p <_{E,rc} q$, **p** and **q** occur in the same client and cronologically **p** returns before **q.**

The rules to a system have sequential consistency are similar to a system that is linearizable. We can say that there's a sequential consistency if there's a sequence of the system that obeys:

[L1] H has the same operations of E, each paired with the return value received in E

[L2] **H obeys the client partial order**

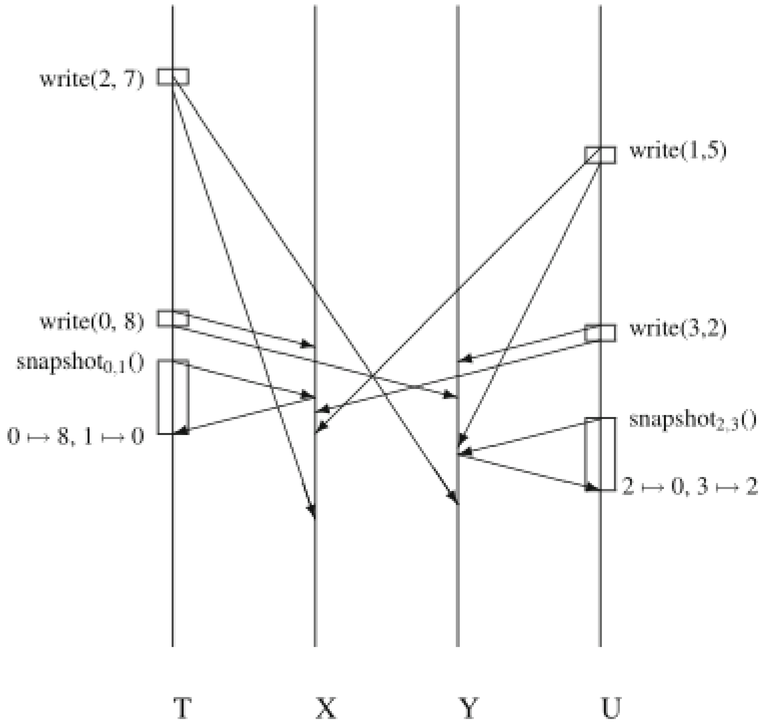
[L3] H is a legal history of the sequential data type that is replicated

The client partial order is a weaker version of the standard partial order. Thus a **every linearizable system also has sequential consistency.**

Since the client partial-order just points that there's an order in the same client, it's ok to place the snapshot before the write(3).

Sequential Consistency does not Compose

We may have two systems that are sequential consistent in separate (U and T), but the composition of partial sequential consistency systems, we not always have a sequential consistent system.



This execution is not sequentially consistent.

The sequence must be:

- **T:**
  write(2,7)
  write(0,8)
  snapshot0,1() [0→8, 1→0]
- **U:**
  write(1,5)
  write(3,2)
  snapshot2,3() [2→0, 3→2]

The snapshots tells us that:

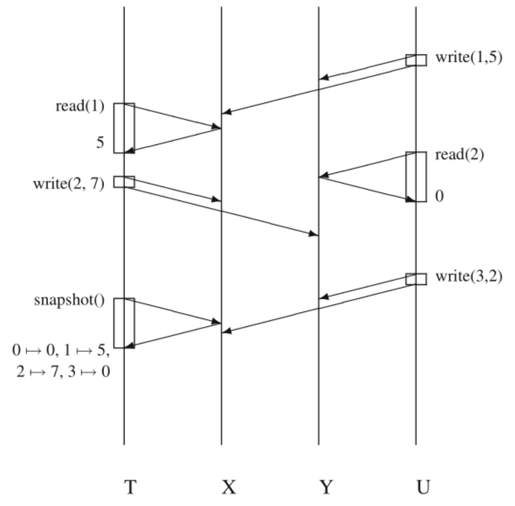
- write(0) $<_{e,ct}$snapshot0,1$<_{e,ct}$write(1)
- write(3) $<_{e,ct}$snapshot2,3$<_{e,ct}$write(2)

The relation with the client partial order does not hold. Because

write(2)$<_{e,ct}$write(0)$<_{e,ct}$write(1), but write(3) is after write(1) and before write(2).

> If we were able to merge it into just one timeline and this timeline H obeys the client partial order, then this would be sequential consistent.

In the following example:
We can say that the order might be:
- write(1)
- read(1)
- read(2)
- write(2)
- snapshot()
- write(3)
And this order obeys the client partial order.



One-copy serializability (transaction-based system)

- The execution of a set of transaction has the same result as if it was executed in a single processor.
- It’s like we have a total order, all operations are transactions and provides isolation
- Nowadays the systems provide weaker consistency to improve performance

Questions

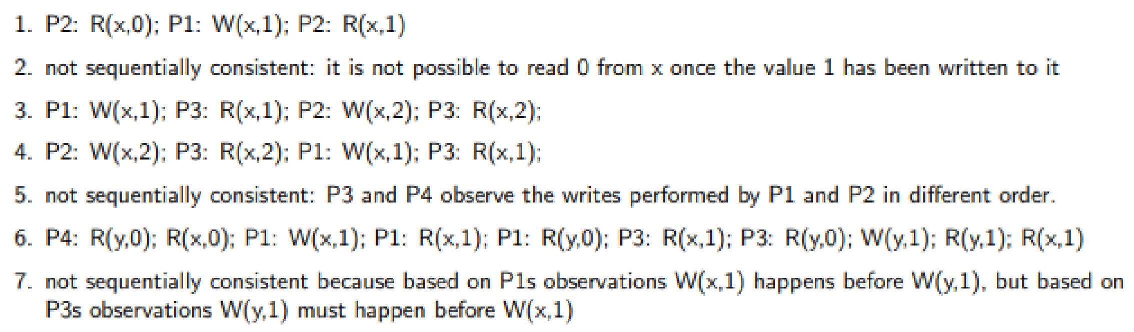
1) What are the rules of linearizability?

2) For the following executions, please indicate if they are sequentially consistent. All variables are initially set to 0.

1. P1: W(x,1);
   P2: R(x,0); R(x,1)

2. P1: W(x,1);
   P2: R(x,1); R(x,0);
3. P1: W(x,1);
   P2: W(x,2);
   P3: R(x,1); R(x,2);
4. P1: W(x,1);
   P2: W(x,2);
   P3: R(x,2); R(x,1);
5. P1: W(x,1);
   P2: W(x,2);
   P3: R(x,2); R(x,1);
   P4: R(x,1); R(x,2);
6. P1: W(x,1); R(x,1); R(y,0);
   P2: W(y,1); R(y,1); R(x,1);
   P3: R(x,1); R(y,0);
   P4: R(y,0); R(x,0);
7. P1: W(x,1); R(x,1); R(y,0);
   P2: W(y,1); R(y,1); R(x,1);
   P3: R(y,1); R(x,0);

▼ Answer



1. P2: R(x,0); P1: W(x,1); P2: R(x,1)
2. not sequentially consistent: it is not possible to read 0 from x once the value 1 has been written to it
3. P1: W(x,1); P3: R(x,1); P2: W(x,2); P3: R(x,2);
4. P2: W(x,2); P3: R(x,2); P1: W(x,1); P3: R(x,1);
5. not sequentially consistent: P3 and P4 observe the writes performed by P1 and P2 in different order.
6. P4: R(y,0); R(x,0); P1: W(x,1); P1: R(x,1); P1: R(y,0); P3: R(x,1); P3: R(y,0); W(y,1); R(y,1); R(x,1)
7. not sequentially consistent because based on P1s observations W(x,1) happens before W(y,1), but based on P3s observations W(y,1) must happen before W(x,1)

Explaining the question 7 explicitly:

W1(x,1) < R1(x,1) < R1(y,0)

W2(y,1) < R2(y,1) < R2(x,1)

R3(y,1) < R3(x,0)

R1(y,0) < W2(y,1) && R3(x,0) < W1(x,1) && R3(y,1) < W2(y,1)

But W2(y,1) > Any event in P1. So, it's impossible to have R3(y,1) before R3(x,0).

3) The memory locations x and y are originally 0. The processors execute the following sequence:
P1: W(x,1); R(y,0);
P2: W(y,1); R(y,1); R(x,1);
P3: R(y,1); R(x,0);

▼ Answer

1) Trying to create a total order:

P1: W(x,1)

P1: R(y,0)

P2: W(y,1)

P2: R(y,1)

P2: R(x,1)

P3: R(y,1)

P3: R(x,0) ⇒ Not possible

From this we have a feeling that it's not sequentially consistent.

2) Why?

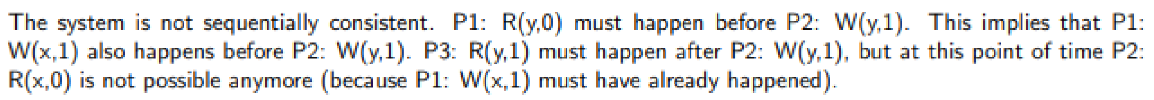
W1(x,1) < R1(y,0) < W2(y,1)

R3(x,0) < W1(x,1) ⇒ Not possible...

This is not possible, since W2(y,1) < R1(y,1) and W1(x,1) < W2(y,1).

In other words, R3(x,0) occurred before W1(x,1), but the W2(y,1) just occurred after it. Thus, R3(y,1) must be after R3(x,0).

The official solution:

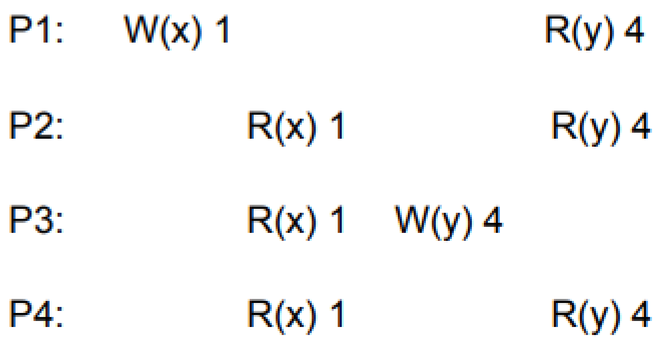


The system is not sequentially consistent. P1: R(y,0) must happen before P2: W(y,1). This implies that P1: W(x,1) also happens before P2: W(y,1). P3: R(y,1) must happen after P2: W(y,1), but at this point of time P2: R(x,0) is not possible anymore (because P1: W(x,1) must have already happened).

Exercises sources

https://spcl.inf.ethz.ch/Teaching/2013-dphpc/sequential_consistency_solution.pdf

4) Is this sequence linearizable? And sequentially consistent?

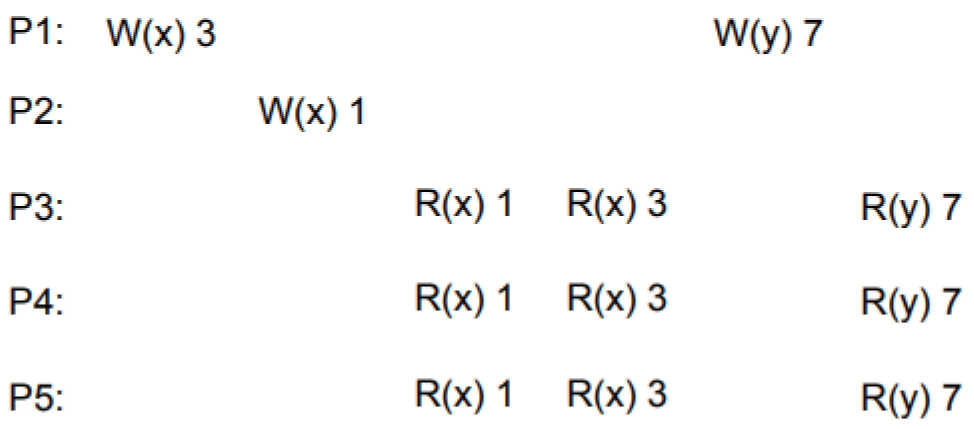


```
P1:   W(x) 1                    R(y) 4

P2:          R(x) 1              R(y) 4

P3:          R(x) 1   W(y) 4

P4:          R(x) 1              R(y) 4
```

▼ Answer

It’s both.

5) Is this sequence linearizable? And sequentially consistent?

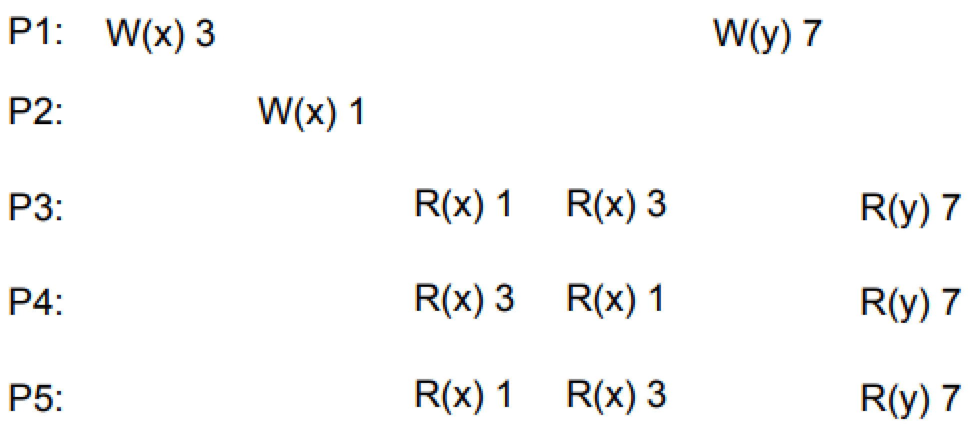


P1:	W(x) 3				W(y) 7	
P2:		W(x) 1				
P3:			R(x) 1	R(x) 3		R(y) 7
P4:			R(x) 1	R(x) 3		R(y) 7
P5:			R(x) 1	R(x) 3		R(y) 7

▼ Answer

It’s not linearizable, because the R(x)1 occurred before R(x)3, but the W(x)3 occurred before W(x)1. But it has sequential consistency.

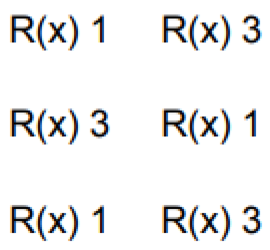
6) Is this sequence linearizable? And sequentially consistent?



P1:	W(x) 3				W(y) 7	
P2:		W(x) 1				
P3:			R(x) 1	R(x) 3		R(y) 7
P4:			R(x) 3	R(x) 1		R(y) 7
P5:			R(x) 1	R(x) 3		R(y) 7

▼ Answer

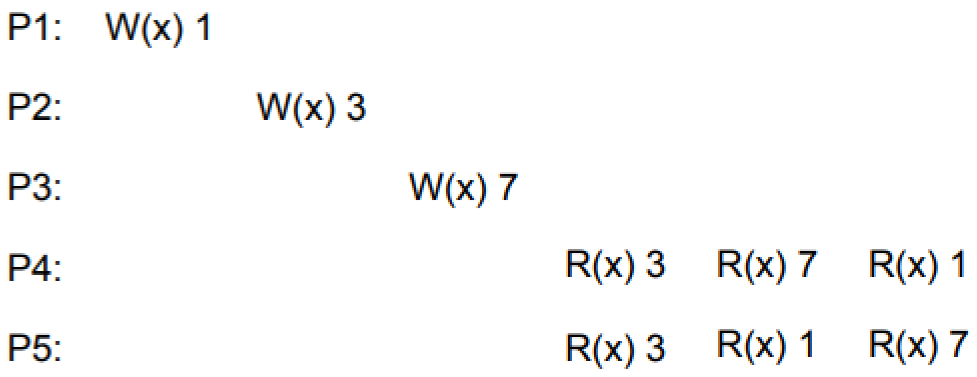
None of both. It’s not sequentially consistency because of these:



R(x) 1	R(x) 3
R(x) 3	R(x) 1
R(x) 1	R(x) 3

And not linearizable, because R(x)1 occurred before R(x)3.

7) And these?



P1:	W(x) 1					
P2:		W(x) 3				
P3:			W(x) 7			
P4:				R(x) 3	R(x) 7	R(x) 1
P5:				R(x) 3	R(x) 1	R(x) 7

▼ Answer

No, same reason from previous questions.