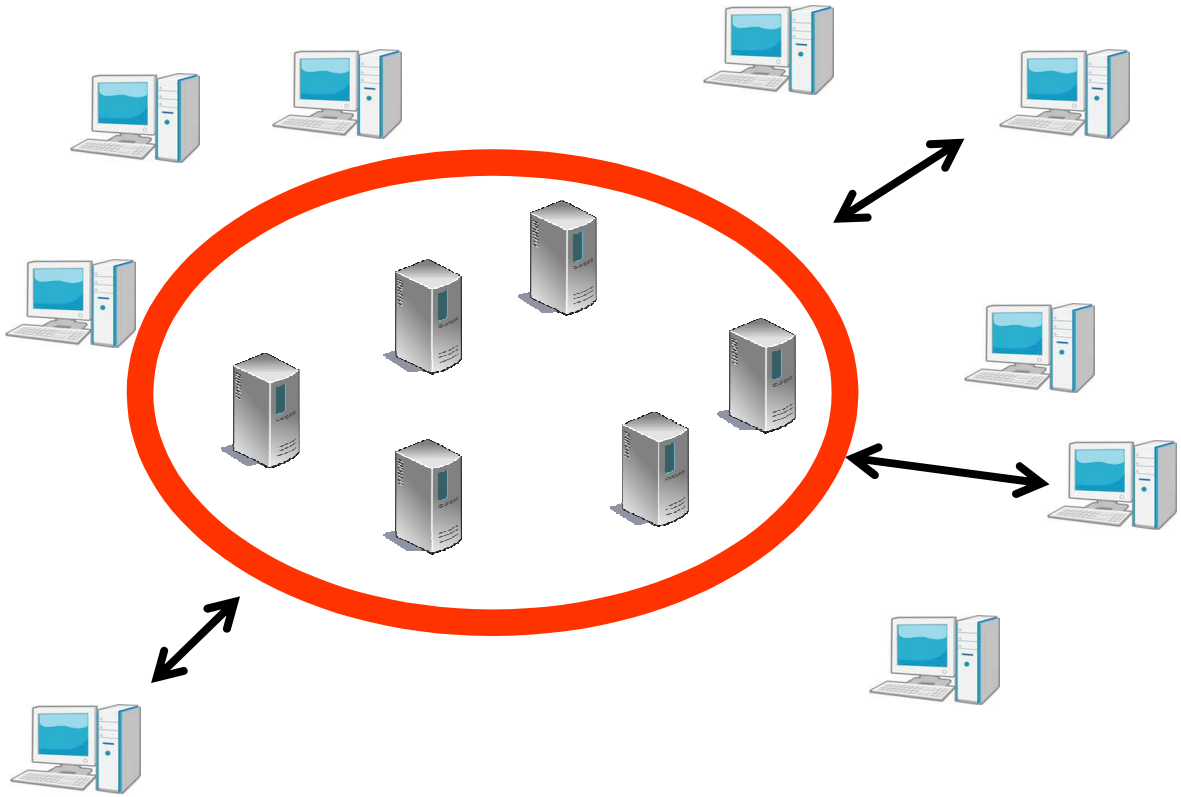
4.3 Strong Consistency

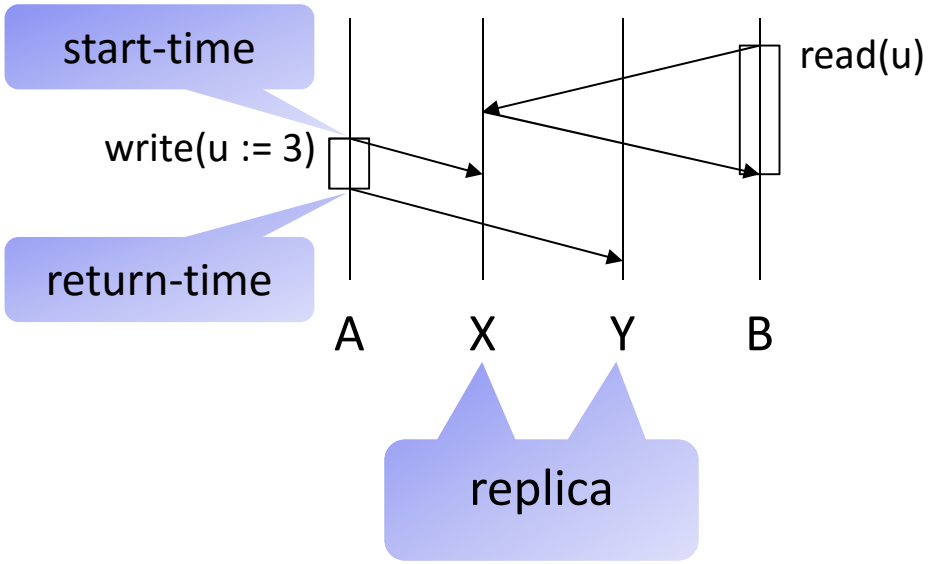
- Interface that describes the system behavior (abstract away implementation details)
- If clients read/write data, they expect the behavior to be the same as for a single storage cell.



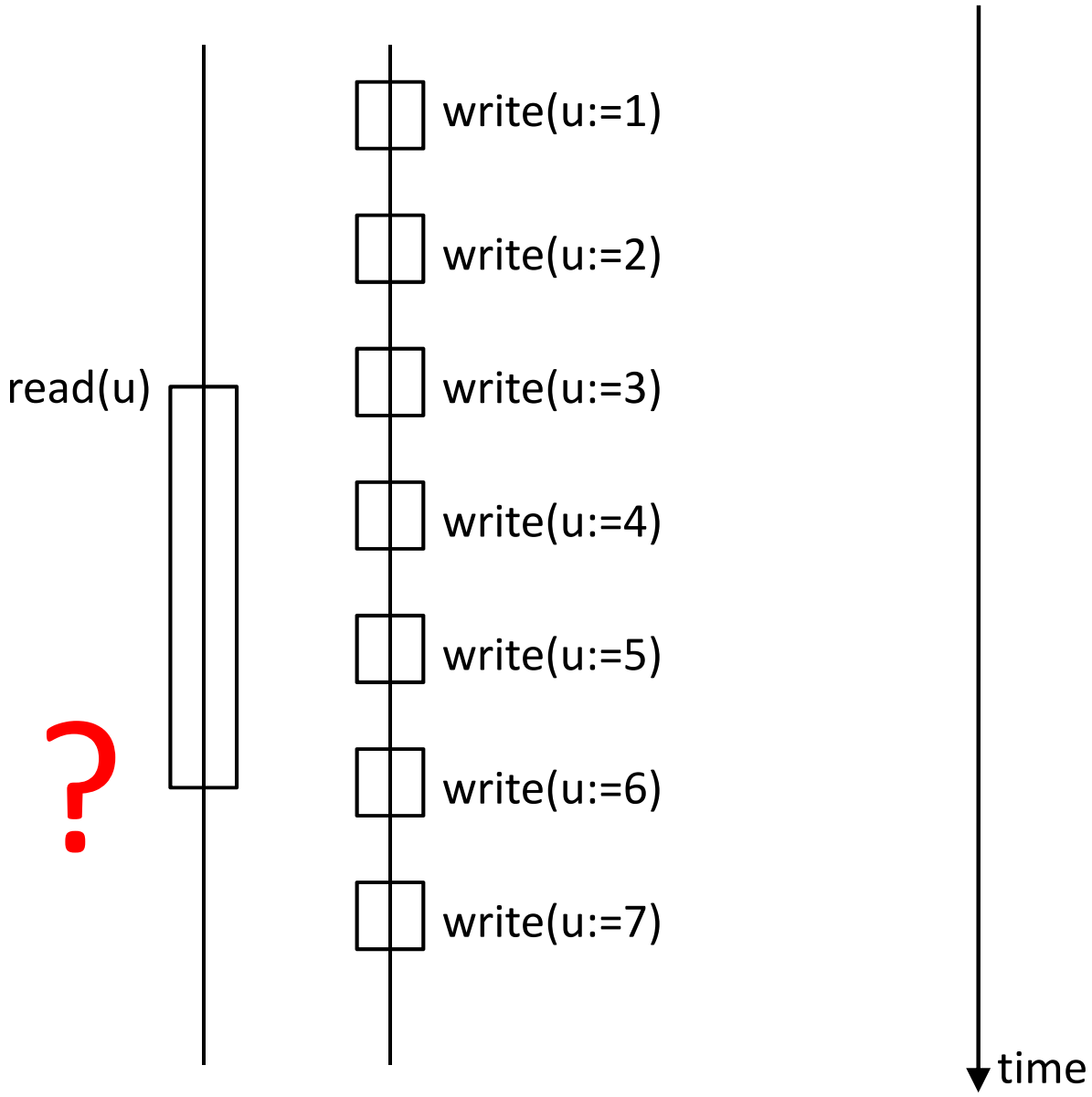
Let‘s Formalize these Ideas

- We have memory that supports 3 types of operations:
  - write(*u* := *v*): write value *v* to the memory location at address *u*
  - read(*u*): Read value stored at address *u* and return it
  - snapshot(): return a map that contains all address-value pairs

- Each operation has a start-time t_S and return-time t_R (time it returns to the invoking client). The duration is given by $t_R - t_S$.



Motivation

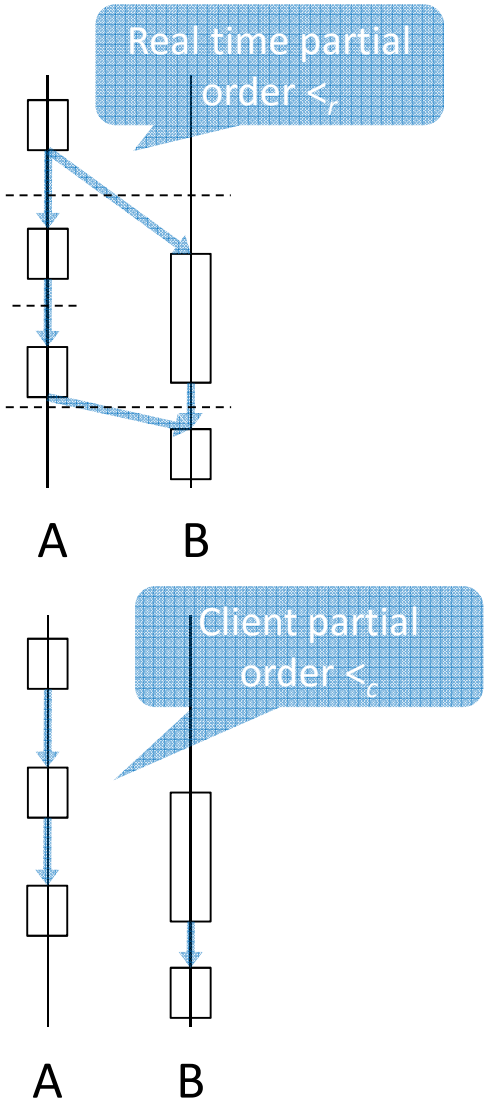

write(u:=1)
write(u:=2)
write(u:=3)
write(u:=4)
write(u:=5)
write(u:=6)
write(u:=7)
read(u)
?
time

Executions

- We look at executions E that define the (partial) order in which processes invoke operations.

- Real-time partial order of an execution $<_r$:
  - $p <_r q$ means that duration of operation p occurs entirely before duration of q (i.e., p returns before the invocation of q in real time).

- Client partial order $<_c$:
  - $p <_c q$ means p and q occur at the same client, and that p returns before q is invoked.



Strong Consistency: Linearizability

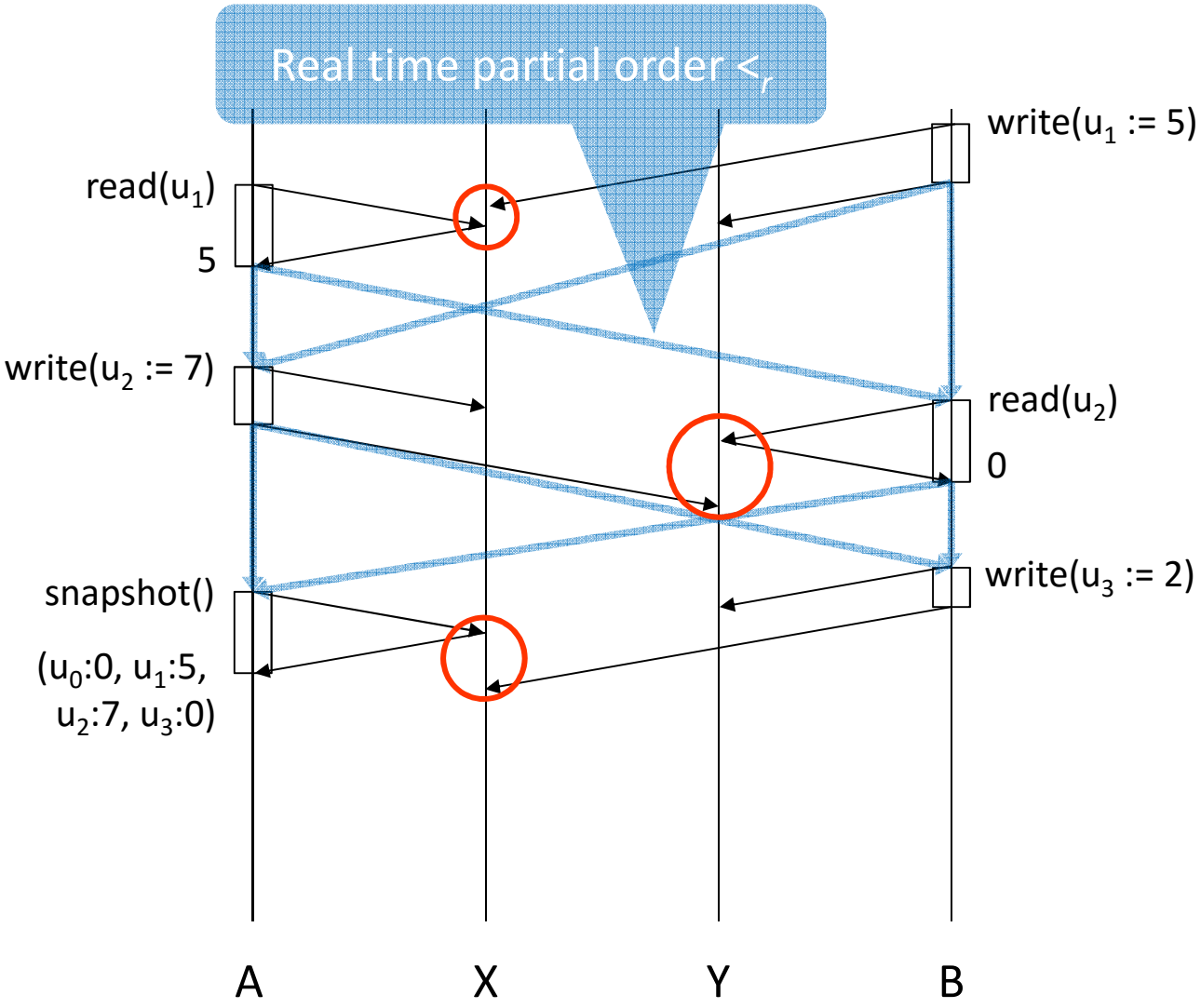
- A replicated system is called linearizable if it behaves exactly as a single-site (unreplicated) system.

Definition

Execution E is linearizable if there exists a sequence H such that:

1) H contains exactly the same operations as E, each paired with the return value received in E
2) The total order of operations in H is compatible with the real-time partial order $<_r$
3) H is a legal history of the data type that is replicated

Example: Linearizable Execution



Valid sequence H:

1.) write(u_1 := 5)
2.) read(u_1) → 5
3.) read(u_2) → 0
4.) write(u_2 := 7)
5.) snapshot() →
(u_0: 0, u_1: 5, u_2:7, u_3:0)
6.) write(u_3 := 2)

For this example, this is the only valid H. In general there might be several sequences H that fullfil all required properties.

Strong Consistency: Sequential Consistency

- Orders at different locations are disregarded if it cannot be determined by any observer within the system.

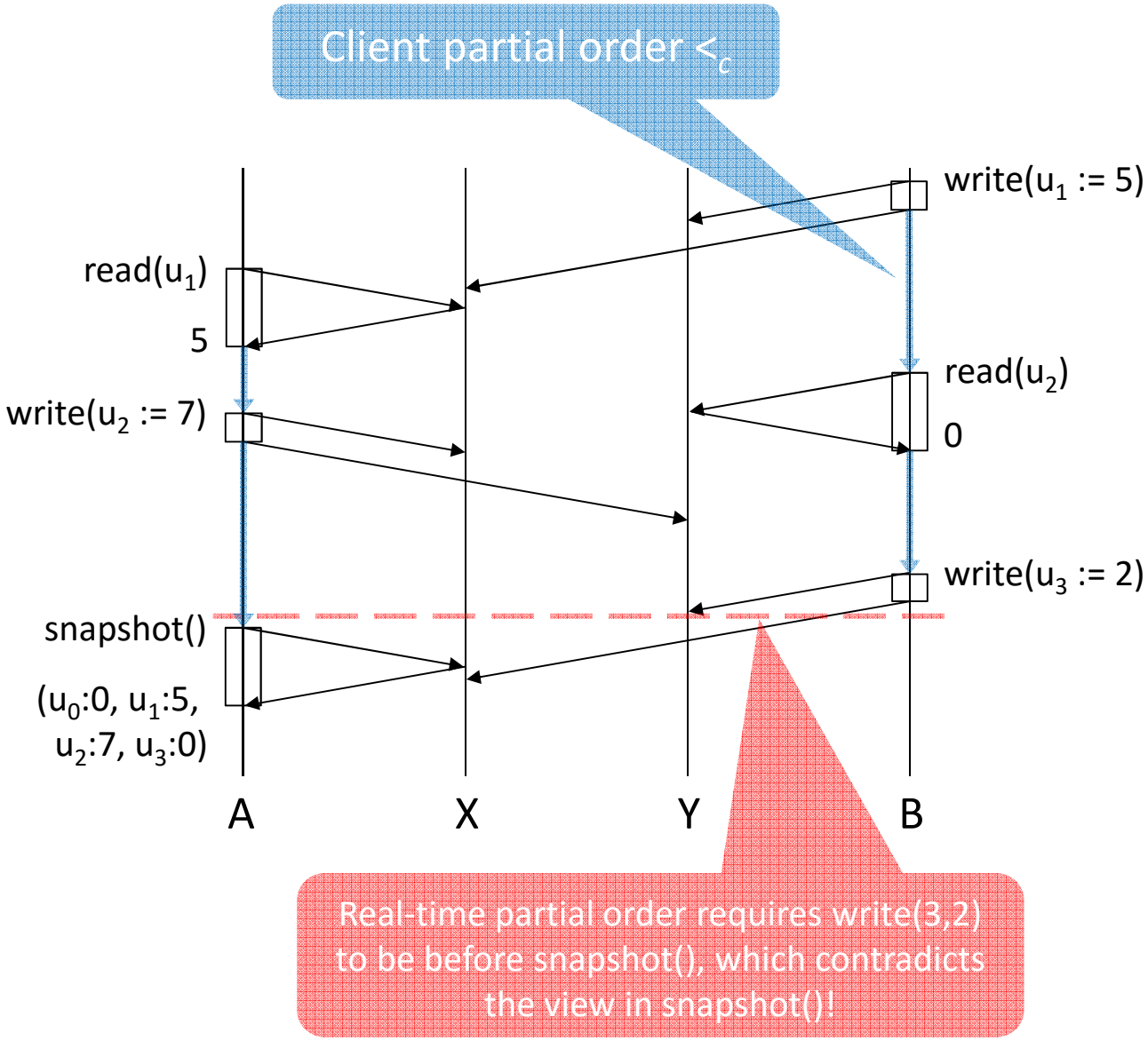
- I.e., a system provides sequential consistency if every node of the system sees the (write) operations on the same memory address in the same order, although the order may be different from the order as defined by real time (as seen by a hypothetical external observer or global clock).

Definition

Execution E is sequentially consistent if there exists a sequence H such that:

1) H contains exactly the same operations as E, each paired with the return value received in E
2) The total order of operations in H is compatible with the client partial order $<_c$
3) H is a legal history of the data type that is replicated

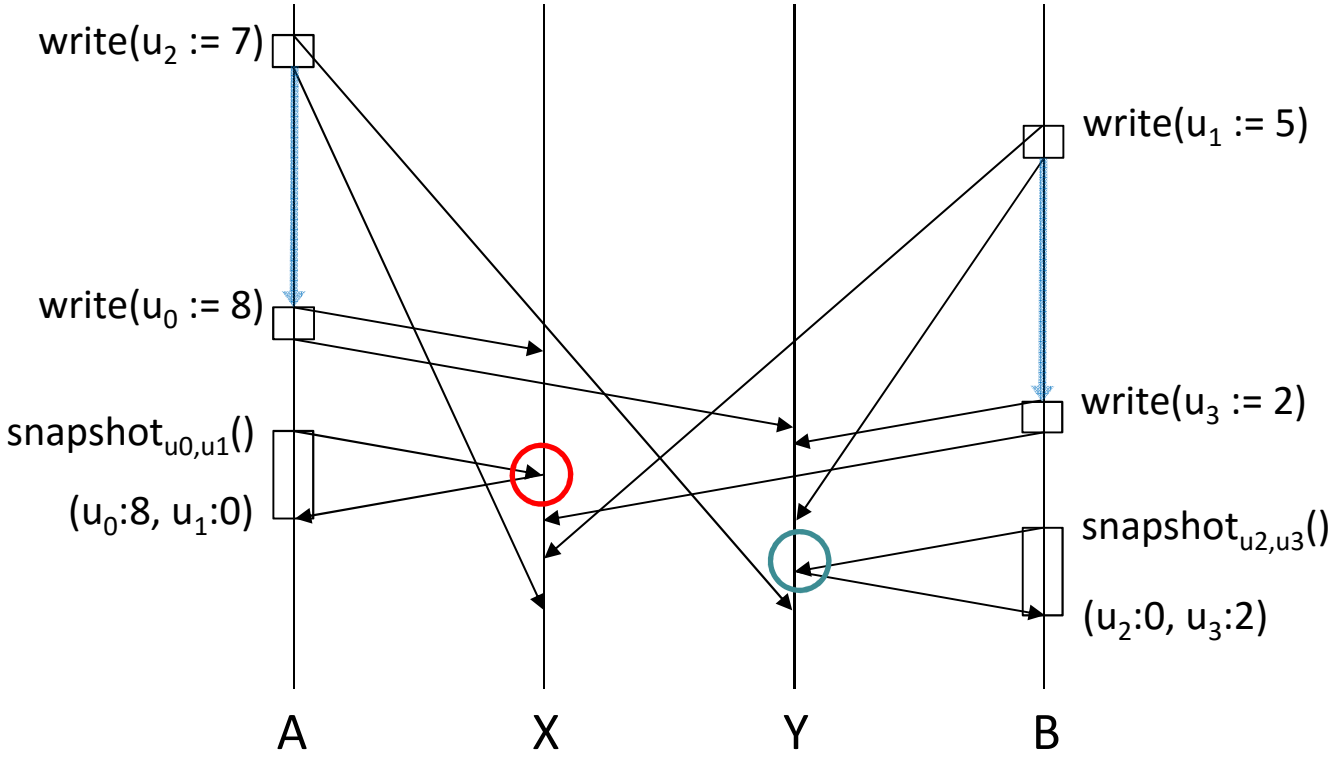
Example: Sequentially Consistent

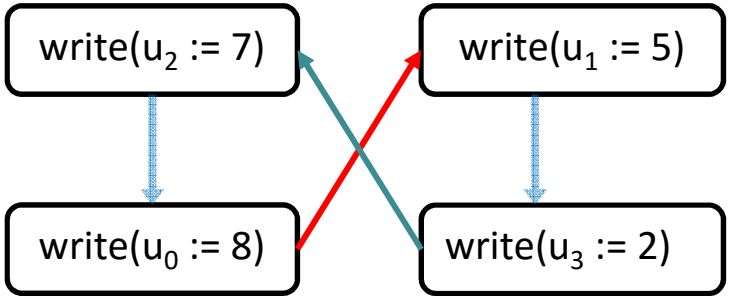


Valid sequence H:

1.) write(u_1 := 5)
2.) read(u_1) → 5
3.) read(u_2) → 0
4.) write(u_2 := 7)
5.) snapshot() →
(u_0:0, u_1:5, u_2:7, u_3:0)
6.) write(u_3 := 2)

Is Every Execution Sequentially Consistent?

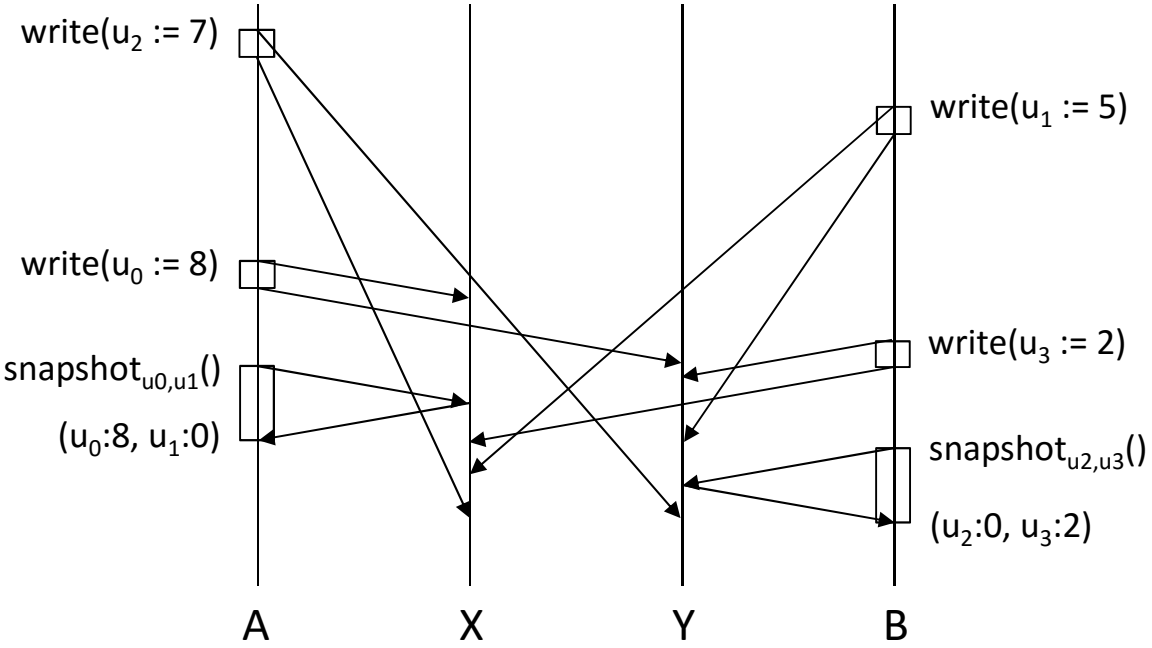




Circular dependencies!

I.e., there is no valid total order and thus above execution is not sequentially consistent

Sequential Consistency does not Compose



- If we only look at data items 0 and 1, operations are sequentially consistent
- If we only look at data items 2 and 3, operation are also sequentially consistent
- But, as we have seen before, the combination is not sequentially consistent

Sequential consistency does not compose!

(this is in contrast to linearizability)


That's all, folks!
Questions & Comments?
Roger Wattenhofer
ETH Zurich – Distributed Computing – www.disco.ethz.ch