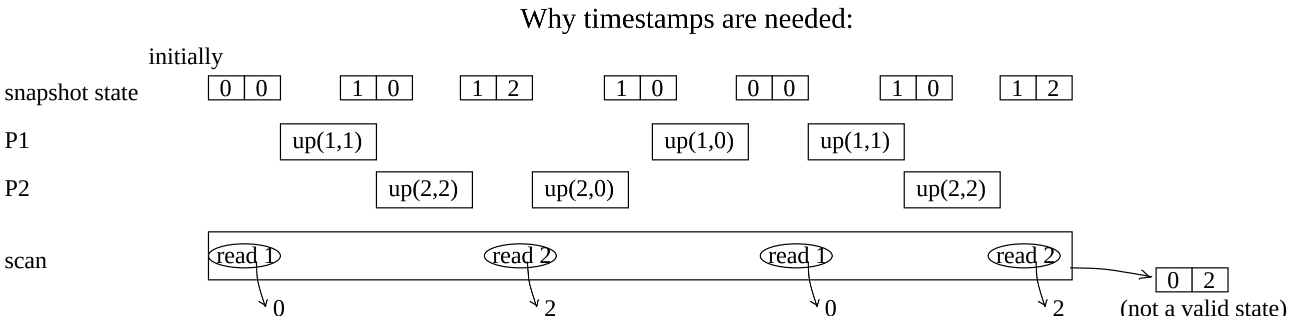
Notes on Linearizability :

To prove implementation linearizable:
For any execution,
construct ordering of the operations that would produce some results;
AND if op_1 completes before op_2 starts, op_1 should come before op_2.
One way to do this:
assign linearization point to each operation within its interval.

Answer to a question from last day: Would the snapshot implementation work without using timestamps? No. Here's an example.



Building better registers

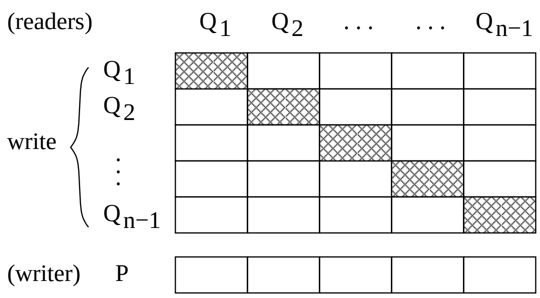
Reference: Amos Israeli and Ming Li; Attiya and Welch, Section 20.2.2.

To implement "powerful" (easy to use) registers from "weak" ones .

(1) Build single-writer, multi-reader (swmr) register
from single-writer, single-reader (swsr) registers.
(We're still in wait-free model: → asynchronous system with halting failures).

swsr **register:** *only one process P can write it,*
and only one process Q allowed to read it.

We will use an array of $(n-1)^2$ swsr registers arranged as shown below.



to write v to the *swmr* register: *P writes v, timestamp (incremented each time you do a write to the swmr register) into registers in bottom row.*

WRONG implementation of read by Q_i :

reads swmr register by looking at its elt in last row,
and return value with latest timestamp.

We need to get the readers to communicate with one another to avoid this kind of situation.

1	2	3	4	5	6	7
1	1	1	0	0	0	0

P starts writing value 1 & dies
Q1 reads 1
Q6 reads 0
execution not linearizable

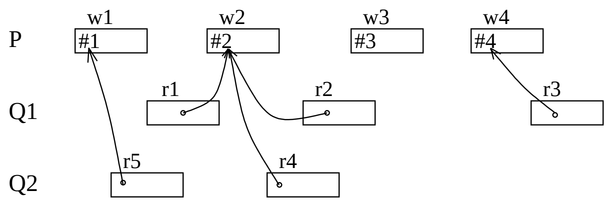
for *proc* Q_i to read *swmr* register :

Q_i reads all swmr registers in column i (including row P);
$(v, t) \leftarrow$ entry with max timestamp;
write (v, t) to all swmr registers of row i;
output v.

Proof Consider any execution:

Linearize it as follows:

order P's writes according to timestamps,
then insert reads into the ordering;
if a read saw max timestamp t,
put it just after write with timestamp t.
(if 2 reads see same max timestamp
put them in order according to their completion times.)



Linearization:

w1 r5 w2 r1 r4 r2 w3 w4 r3

Clearly, all operation return same value as they do in linear order.

Now, check that if op completes before op' begins,
then op precedes op' in linearization.

Case I: op' is a $read$.

- op has already written its entry in its whole row,
  so op' will see it.
- $\rightarrow$ op''s timestamp $\geq$ op's timestamp
- If op is a $read$ & has timestamp $<$ timestamp of op',
  then op definitely precedes op'.
- If op is a write,
  then op precedes op'.
- If op read & has same timestamp as op',
  then op precedes op' because of the way we broke ties between $read$s with same timestamp.

Case II: op' is a write

Then op has timestamp that is strictly smaller than timestamp of op', since P is the only process that increments timestamp.
Therefore, op' comes after op in linearization.