Some issues with linearizability and implementation.

- Randomization and implementation
- Strong linearizability
- (Adversary)

It is not at all an original work! (Neither Carole nor me are authors of any of the papers)

Randomization and adversary

Adversary and randomization.

Ben Or algorithm

Every process p executes the following:

```
0   procedure consensus(v_p)                          {v_p is the initial value of process p}
1       x ← v_p                                       {x is p's current estimate of the decision value}
2       k ← 0
3       while true do
4           k ← k + 1                                 {k is the current phase number}
5           send (R, k, x) to all processes
6           wait for messages of the form (R, k, *) from n − f processes      {"*" can be 0 or 1}
7           if received more than n/2 (R, k, v) with the same v
8           then send (P, k, v) to all processes
9           else send (P, k, ?) to all processes
10          wait for messages of the form (P, k, *) from n − f processes      {"*" can be 0, 1 or ?}
11          if received at least f + 1 (P, k, v) with the same v ≠ ? then decide(v)
12          if at least one (P, k, v) with v ≠ ? then x ← v else x ← 0 or 1 randomly {query r.n.g.}
```

Ben Or

Main points:

- If all proposed values are the same this value is decided
- At the end of the first part of a phase impossible to have 0 and 1. Either
  - O only (ok) or 1 only (ok)
  - (0,?), (1,?), (?)
    - Random choice for processes with ?
    - Then if all random choices in the phase are the same and the same as the value of processes that do not have ? At the end of the phase all proposed values are the same (ok)
- Then termination with proba 1 (at each phase bounded by $\frac{1}{2^n}$)
- Replace local coin with a global coin: at each round for each process get the same random bit. Better?

Strong adversary

(Strong) Adversary?

- Adversary choose when the messages arrive and when processes take steps.
- Adversary knows at any time all the state of the protocol (but not the future)
- It doesn't know the future

Problem… The adversary doesn't control the random choices. If the random choice is say 0, perhaps it could manage the messages in such a way that some processes get 1 at the end of the first part of the phase.

With local coin that is not possible, with global coin it is (with some conditions: $3t > n > 2t$) and Ben Or never terminates!

Aguilera,Toueg:The correctness proof of Ben-Or's randomized consensus algorithm. Distributed Comput. 25(5): 371-381 (2012)

Adversary…

Message passing: choose when the messages arrive and when processes make steps.

Shared memory: choose the schedule of processes. With linearizable objects choose the linearization point.

Linearization

- Given some object (like a queue) define what is its behavior in a concurrent framework (like a concurrent queue)?

First sequential…

- Example queue:
  - operations : *enq*(item x) ; item *deq*()
  - set of states : sequence of items
  - sequential specification :

$$\{state = f\} enq(x) \{state = f.x\} \quad return\ ok$$

$$if(state \neq \emptyset)\{state = a.f\} deq()\{state = f\} \quad return\ a$$

invocation

response

Sequential specification

Another example Register

- Operations: item read, write(item)
- state: item
- sequential specification:

$$\{state = y\} write(x) \{state = x\} \quad return\ ok$$

$$\{state = y\} read() \{state = y\} \quad return\ y$$

Concurrent objects

Sequential specifications defines the correct executions for sequential executions

For concurrent executions? Get a sequential execution that may correspond.

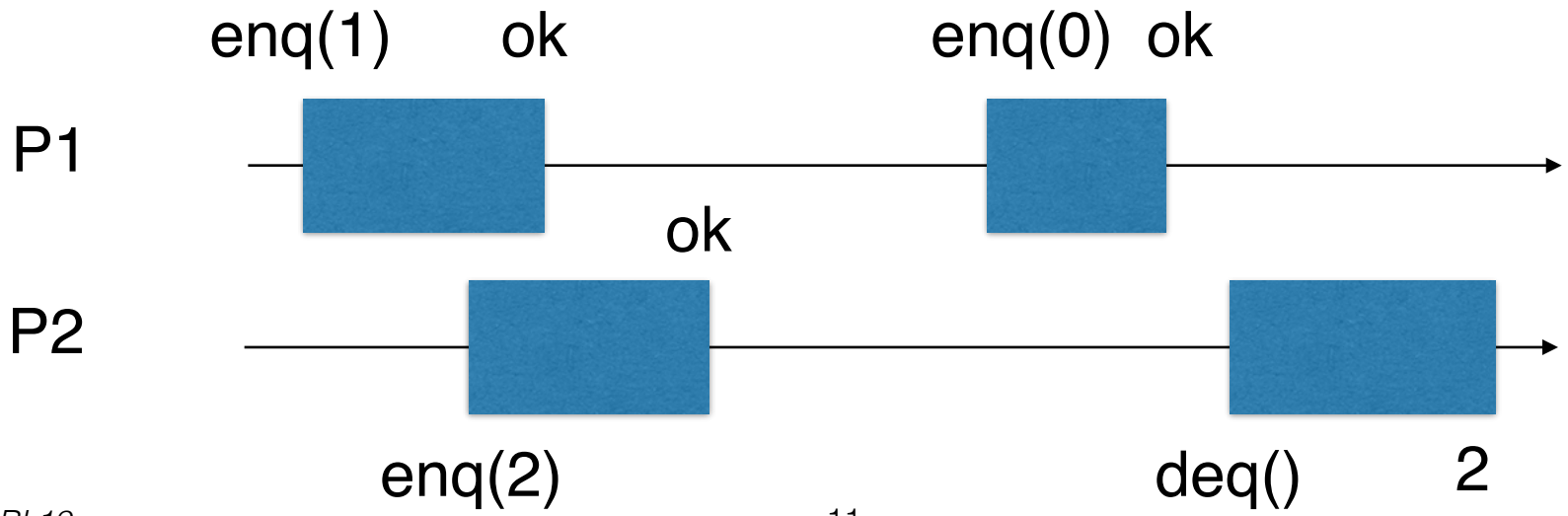
Linearization:

- There is an sequential execution « corresponding » to the execution
  - Choose a linearization point in the interval [invoke, response] -> sequential execution
  - This sequential execution satisfies the sequential specification

Histories

- A *history* is a sequence of invocation and (matching) response

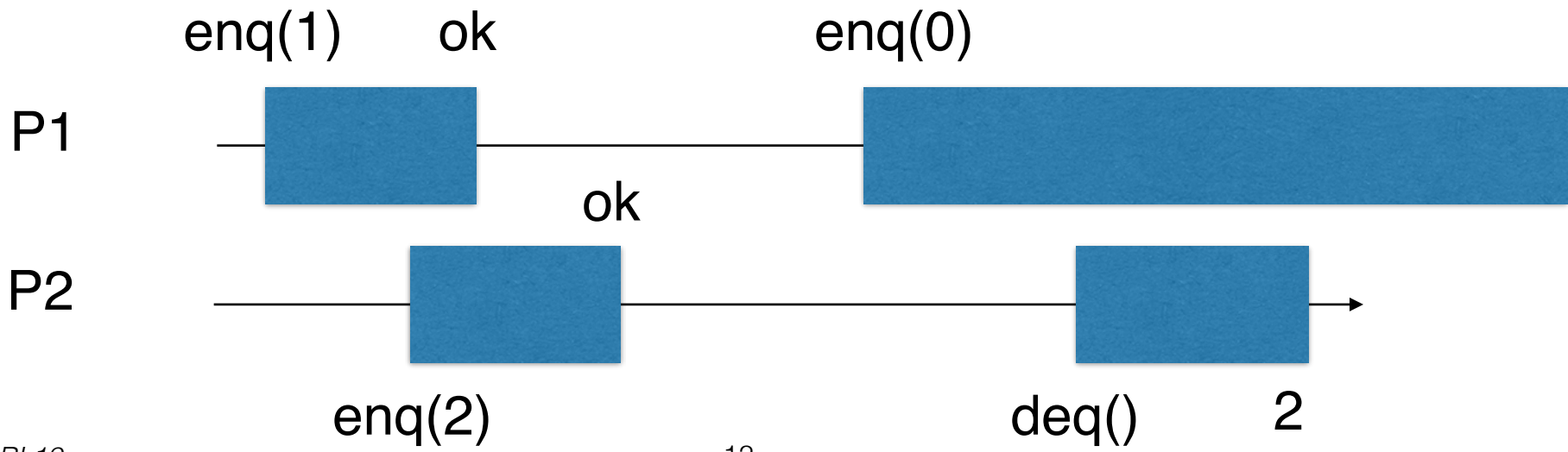
$$enq_1(1)enq_2(2)ok_1ok_2enq_1(0)ok_1deq_2()2_2$$



Histories

- A history is a sequence of invocation and (matching) response

$$enq_1(1)enq_2(2)ok_1ok_2enq_1(0)deq_2()2_2$$



- A history is **sequential** if every invocation is *immediately* followed by a matching response
  - (A sequential history has no concurrent operations)
- A sequential history is **legal** if it satisfies the sequential specification of the shared object

more formally (a little bit…)

- $(H \mid p_i)$(resp. $(H \mid O)$)is the restriction of history to process p_i (resp. Object O)
- Each process performs operations sequentially: invoke - response (possibly (only) the last operation has no answer): *well-formed history*
- An operation op is *pending* if there is no matching response to the invocation. (pending operations in a history correspond to crashes)
- An operation op is *complete* in a history *H* if *H* contains both the invocation and the matching response of op
- A *completion* of *H* is a history *H'* that includes all complete operations of *H* and a subset of incomplete operation with matching responses
- (informally: if some operation has no answer then it is possible to consider either that the operation has not occurred or to complete with a response)

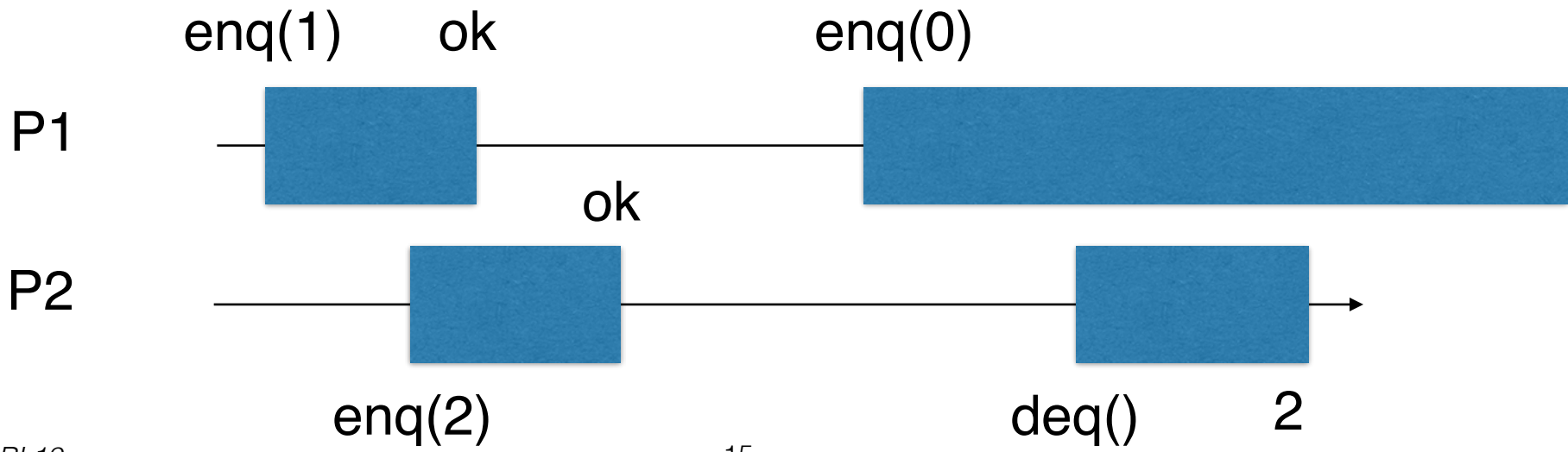
Histories

$$H = enq_1(1)enq_2(2)ok_1ok_2enq_1(0)deq_2()2_2$$

$$H' = enq_1(1)enq_2(2)ok_1ok_2enq_1(0)deq_2()2_2ok_1$$

$$H' = enq_1(1)enq_2(2)ok_1ok_2deq_2()2_2$$

$$H' = enq_1(1)enq_2(2)ok_1ok_2enq_1(0)ok_1deq_2()2_2$$



Histories

- Histories H and G are equivalent for p_i if and only if

$$H|p_i = G|p_i$$

- Histories H and H' are equivalent if for all p_i, H and H' are equivalent

- e.g

$$enq_1(1)enq_2(2)ok_1ok_2enq_1(0)deq_2()2_2$$

$$enq_2(2)ok_2enq_1(1)ok_1deq_2()2_2enq_1(0)$$

Linearizability (Atomicity)

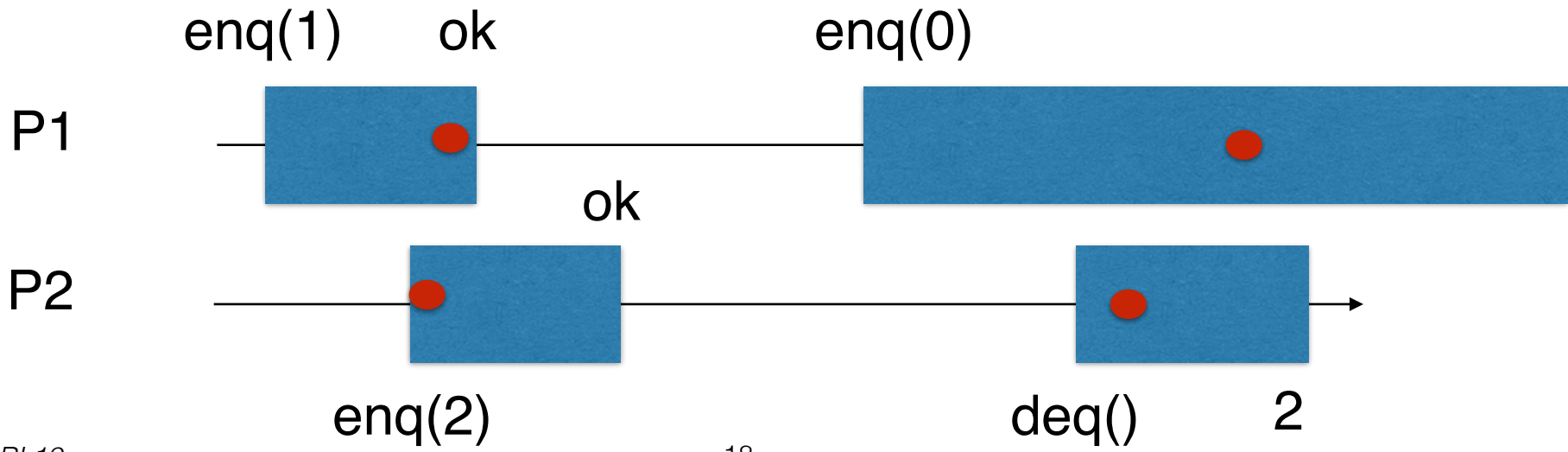
- A history H is **linearizable** if there exists a *sequential legal* history S such that:
  - S preserves the *precedence* relation of H:
    - if op_1 precedes op_2 in H (i.e the response of op1 is before the invocation of op_2 in H) then op_1 precedes op_2 in S
  - S is equivalent to some completion of H
- (there is a linearization point (time) between the time of invocation and time of the matching response such that the corresponding sequential history satisfies the sequential specification)
- (f is a linearization function: f maps each $H \in \mathscr{H}$ to a linearization (=chooses for H one linearization point)

Histories

$$H = enq_1(1)enq_2(2)ok_1ok_2enq_1(0)deq_2()2_2$$

$$S = enq_2(2)ok_2enq_1(1)ok_1deq_2()2_2enq_1(0)ok_1$$

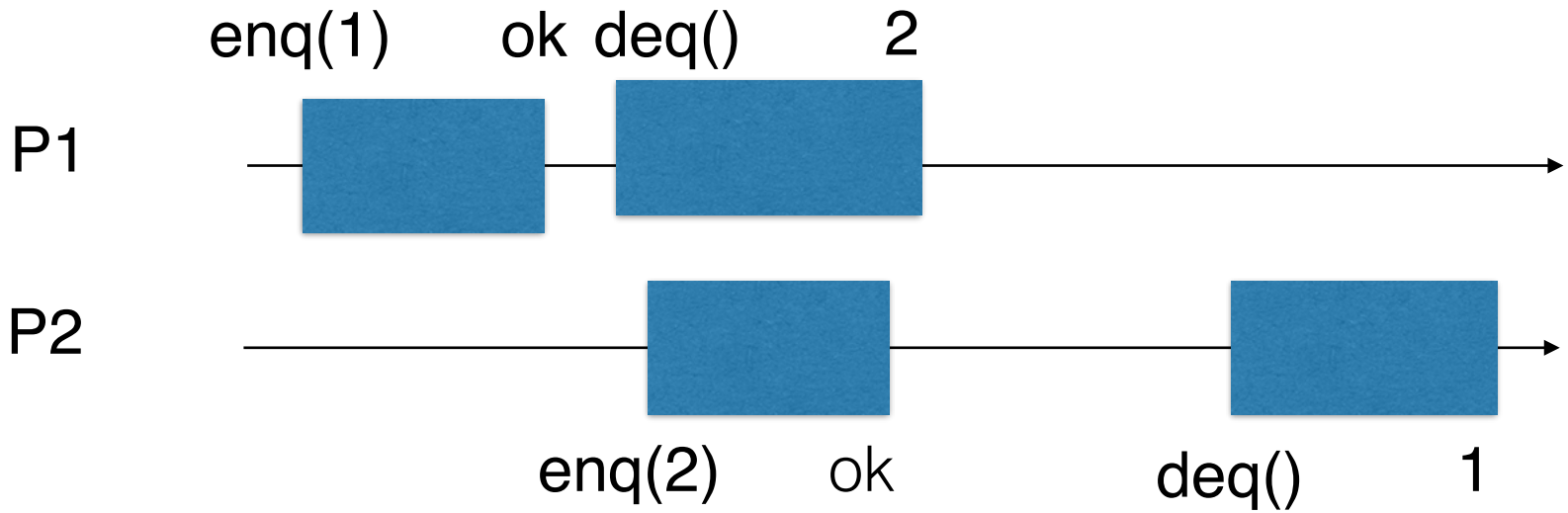
S sequential, legal, equivalent to some completion of H and preserves precedence relation



Histories

$$H = enq_1(1)ok_1deq_1()enq_2(2)ok_2 2_1 deq_2()1_2$$

S ??



Linearizability

Remarks:

Linearization point: for some algorithms, the linearization point is not static.

The linearization point is off-line (some events in the future may force linearization now)

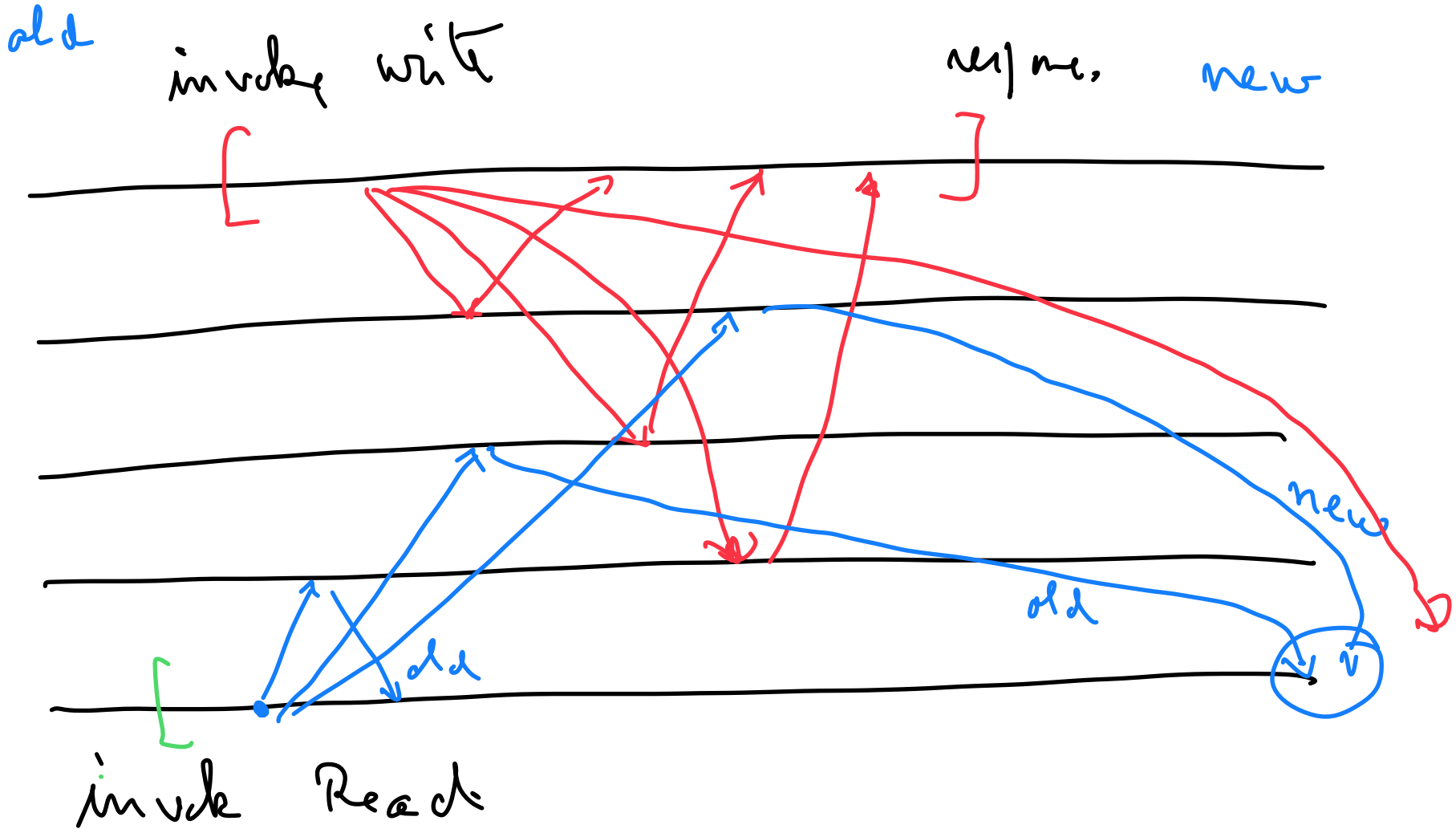
ABD

For the writer
to write(v)
$seq := seq + 1$
send (W, v, seq) to all
wait until receiving $\lfloor n/2 \rfloor + 1$ messages (ACK, seq)

For the reader
to read()
send(R) to all
wait until receiving $\lfloor n/2 \rfloor + 1$ messages (V, v, s) such that $s > seq$
return val
such that (V, val, S) has been received
and S is the max of the sequence number of received V messages

For all processes
when (W, v, s) is received
if $s > seq$ then
$val := v; seq := s$
send (ACK, s)

when(R) is received
send (V, val, seq) to p_r


old
invoke write
release,
new
new
old
old
v
invoke Read

Comments

- The write ends after receptions of a majority of ack: the linearization point is before.
- Depending on the arrival of the 2 last messages the read returns old or new value.
- If the read returns the old value the linearization point of the read is before the linearization point of the write. The order of events is changed after the linearization point of the write

Consequences…

Atomicity and linearizability are not equivalent:

- A program that terminates with atomic registers and does not terminate with linearizable register
- (with strong adversary: the adversary chooses the linearization point)

Algorithm 1 A game for $n \geq 3$ processes

SHARED MWMR REGISTERS R_1, R_2, C

CODE FOR p_i, $i \in \{0, 1\}$:

1: **for** rounds $j = 1, 2, \ldots$ **do**
   // PHASE 1
2: $R_1 \leftarrow [i, j]$
3: **if** $i = 0$ **then**
   // p_0 flips a coin and
   // writes it into C
4: $c \leftarrow$ coin flip
5: $C \leftarrow c$
6: **end if**
   // PHASE 2
7: $R_2 \leftarrow 0$
8: $v \leftarrow R_2$
9: **if** $v < n - 2$ **then**
10: **exit for loop**
11: **end if**
12: **end for**
13: **return**

CODE FOR p_i, $i \in \{2, \ldots, n-1\}$:

14: **for** rounds $j = 1, 2, \ldots$ **do**
   // PHASE 1
15: $R_1 \leftarrow \perp$
16: $C \leftarrow \perp$
17: $u_1 \leftarrow R_1$
18: $u_2 \leftarrow R_1$
19: $c \leftarrow C$
20: **if** $u_1 = \perp$ **or** $u_2 = \perp$ **or** $c = \perp$ **then**
21: **exit for loop**
22: **end if**
23: **if** $u_1 \neq [c, j]$ **or** $u_2 \neq [1 - c, j]$ **then**
24: **exit for loop**
25: **end if**
   // PHASE 2
26: $R_2 \leftarrow 0$
27: $v \leftarrow R_2$
28: $v \leftarrow v + 1$
29: $R_2 \leftarrow v$
30: **end for**
31: **return**

From Vassos Hadzilacos, Xing Hu, Sam Toueg: On Register Linearizability and Termination. PODC 2021

terminates for atomic registers and not for linearizability

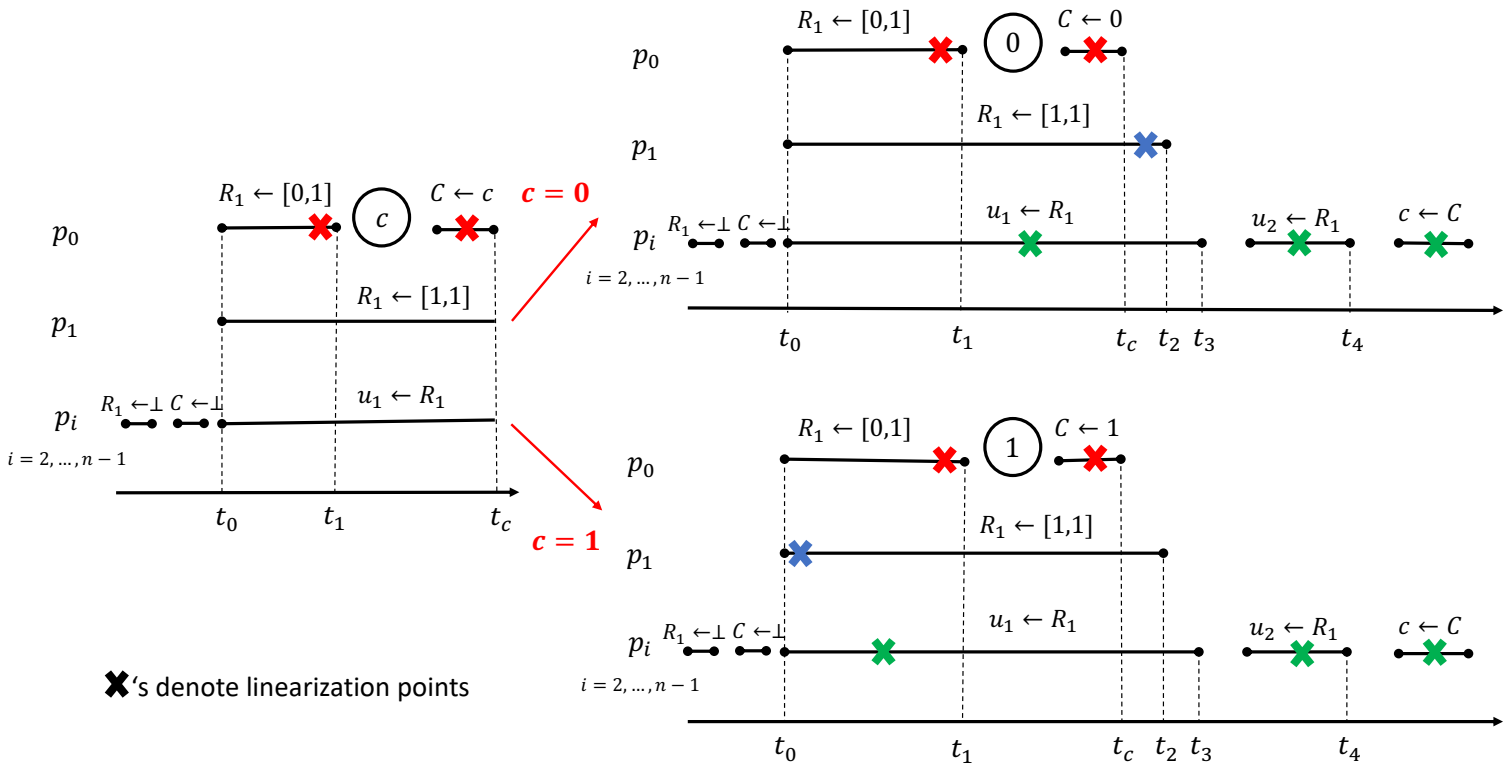


Figure 1: Phase 1 in round $j = 1$ of an infinite execution

From Vassos Hadzilacos, Xing Hu, Sam Toueg: On Register Linearizability and Termination. PODC 2021

Strong linearization

With linearizability the order of all operations is made off-line given the entire execution.

With strong linearizability the order of all operations has to be fixed irrevocably on-line without the knowledge of the rest of the execution.

Strong linearization

Let $\mathscr{H}$ the set of histories: a linearization function f maps each $H \in \mathscr{H}$ to a sequential $f(H)$ (Linearization)

A function f is a strong linearization function for $\mathscr{H}$ is a linearization function such that for all histories $G, H \in \mathscr{H}$ if G is a prefix of H, $f(G)$ is a prefix of $f(H)$

But…

Some (most?) problems do not have strong linearizable implementations from SWMR-registers.

- MWMR registers from SWMR registers…
- Max-registers
- Counters
- Snapshot
- …

Some Works:

- Vassos Hadzilacos, Xing Hu, Sam Toueg: On Register Linearizability and Termination. PODC 2021: 521-531

  Marcos K. Aguilera, Sam Toueg: The correctness proof of Ben-Or's randomized consensus algorithm. Distributed Comput. 25(5): 371-381 (2012)

  Hagit Attiya, Constantin Enea, Jennifer L. Welch: Impossibility of Strongly-Linearizable Message-Passing Objects via Simulation by Single-Writer Registers. DISC 2021: 7:1-7:18

- Wojciech M. Golab, Lisa Higham, Philipp Woelfel: Linearizable implementations do not suffice for randomized distributed computation. STOC 2011: 373-382

- Maryam Helmi, Lisa Higham, Philipp Woelfel: strongly linearizable implementations: possibilities and impossibilities. PODC 2012: 385-394