1. Read/write register

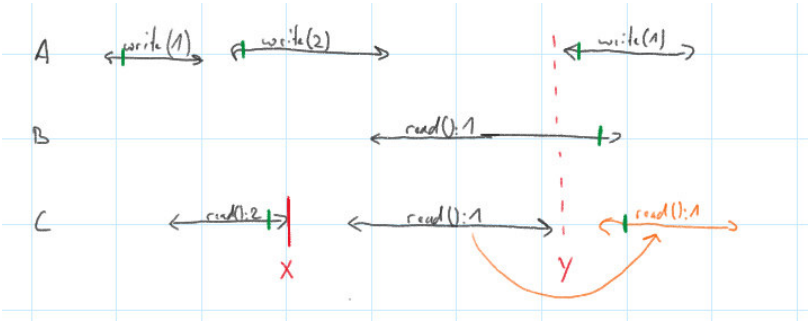
a. Concurrent threads A,B,C, register r

This history is not linearizable, but it is sequentially consistent. It is not linearizable because at point X, 2 must be written but the C r.read():1 perceedes the second A r.write(1).

However, it is sequentially consistent with the following equivalent history:

A: r.write(1)
A: r:void
A: r.write(2)
A: r:void
C: r.read()
C: r:2
A: r.write(1)
A: r:void
C: r.read()
C: r:1
B: r.read()
B: r:1

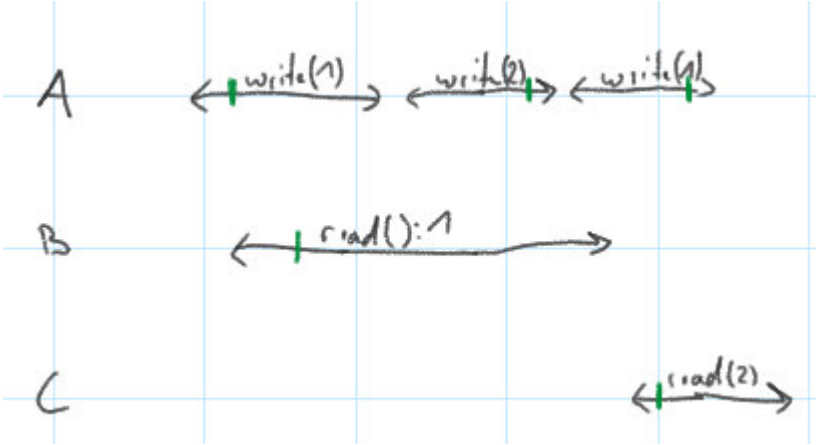
The following illustration shows the invocations. In orange, the movement of the C read(): 1 is shown which makes the history sequentially consistent.



b. Concurrent threads A , B , C , register r.

This is history is linearizable. The following history is equivalent. The figure shows in green, where the method “takes effect”

A: r.write(1)
A: r:void
B: r.read()
B: r:1
A: r.write(2)
A: r:void
C: r.read()
C: r:2
A: r.write(1)
A: r:void

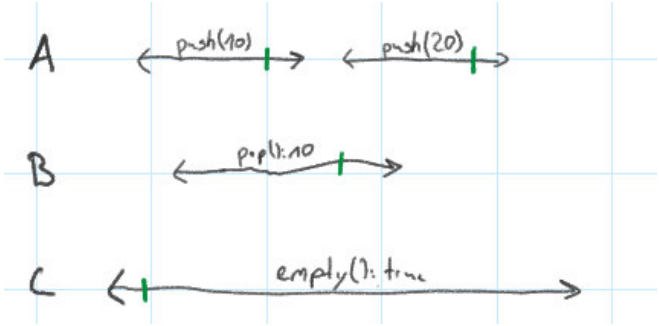


2. Stack

a. Concurrent threads A , B , C , stack s

This history is linearizable, the equivalent history looks the following:

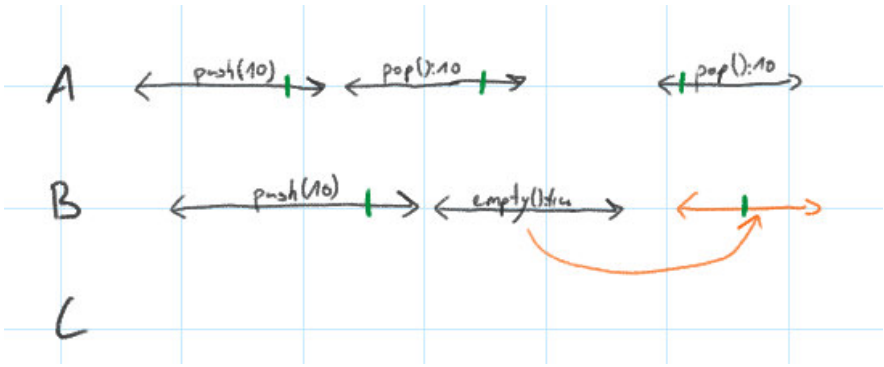
C: s.empty()
C: s:true
A: s.push(10)
A: s:void
B: s.pop()
B: s:10
A: s.push(20)
A: s:void



b. Concurrent threads A , B , C , stack s

This history is not linearizable because the B empty(): true precedes the last A pop():10. The history is sequentially consistent. The orange drawing show how the empty operation can be moved to make the history sequentially consistent.

A: s.push(10)
A: s:void
B: s.push(10)
B: s:void
A: s.pop()
A: s:10
A: s.pop()
A: s:10
B: s.empty()
B: s:true



3. Queue

a. Concurrent threads A , B , C , queue q.

This history is neither linearizable nor sequentially consistent under the following assumptions:

- dequeue also removes the retrieved element from the queue
- the queue has been initially empty
- the dequeue operation is thread safe i.e. it is considered as critical section and accordingly protected from concurrent access

Then, it is not possible because y is enqueued only once but dequeued twice. If for example the third assumption does not hold, then A and C could dequeue at exactly the same time and both get y as a result. If the first assumption does not hold, then we could get y as many times as we like by invoking dequeue. If the second assumption does not hold, then we could define, that initially the queue had one entry, which was y.