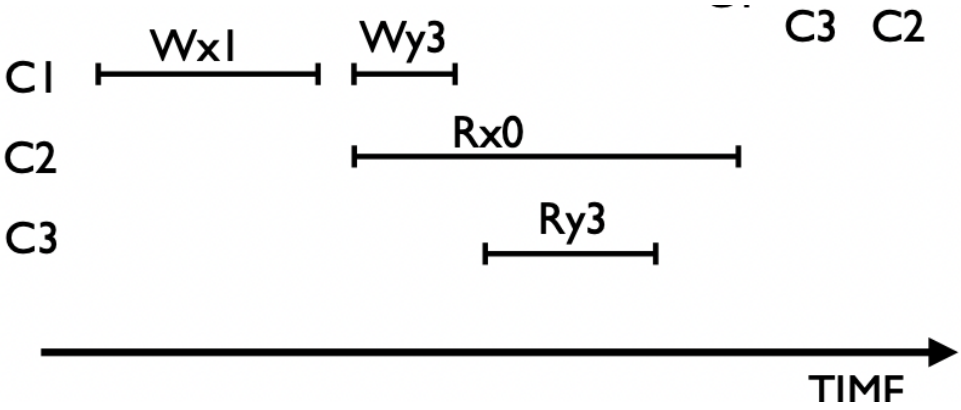
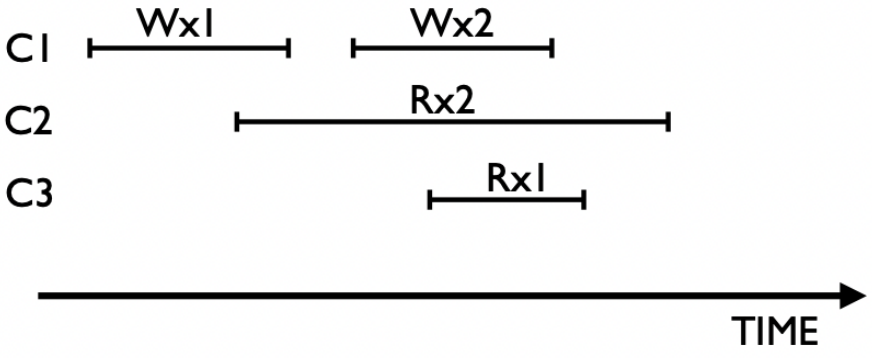
CAS CS 350
Lec 26

Summary of Consistency Models

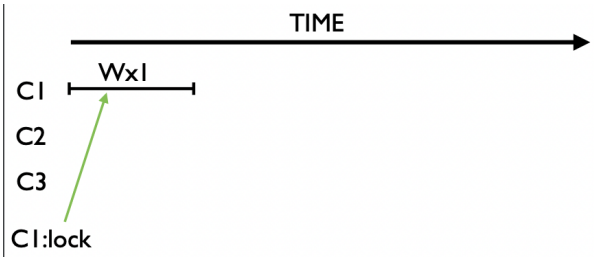
1. Linearizability
    a. Read/Write operations are executed in the order defined by real time
    b. Every read will see the value of the latest completed Write
    c. The raft service supports linearizability
2. Example
    a. 

    b. Three clients interact with a black-box system that stores objects x, y (initially x = y = 0)
    c. The system maintains multiple replicas for fault-tolerance and we don't know the consistency guarantees provided
    d. WxN → client sets value of x to N
    RxN → client reads x and observes value N
    e. The execution above is not linearizable because C2 should have observed x as 1
    f. 

    g. The execution above is linearizable because C3 observes x as 1 while C2 read overlaps with both operations by C1 so it may see the result of any of those
    Or (x = 0)

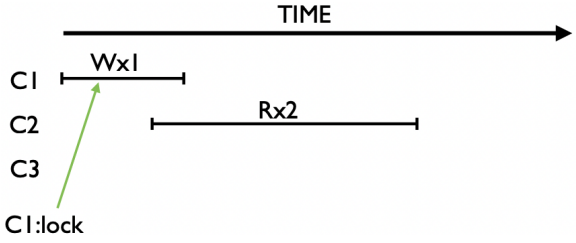
h. Further overview

i.



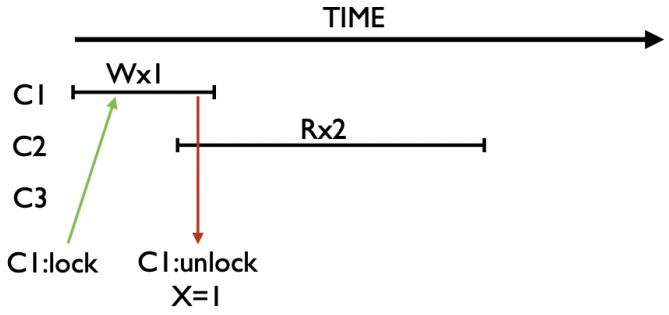
ii. C1 gets locked (might have been blocked waiting for it)

iii.



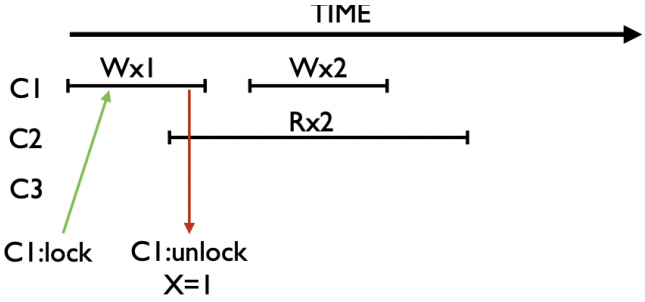
iv. C2 starts read but gets delayed in the network (or block on lock)

v.



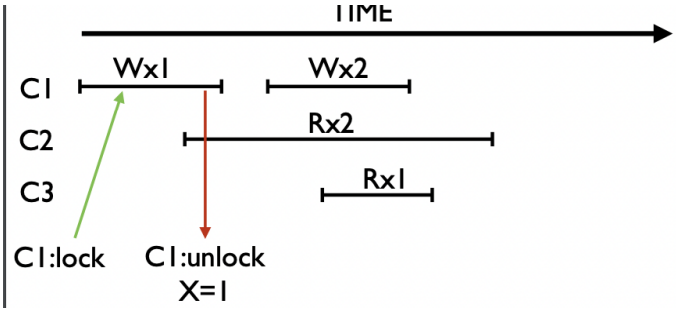
vi. C1 release lock - think X = 1 instantaneously happens

vii.

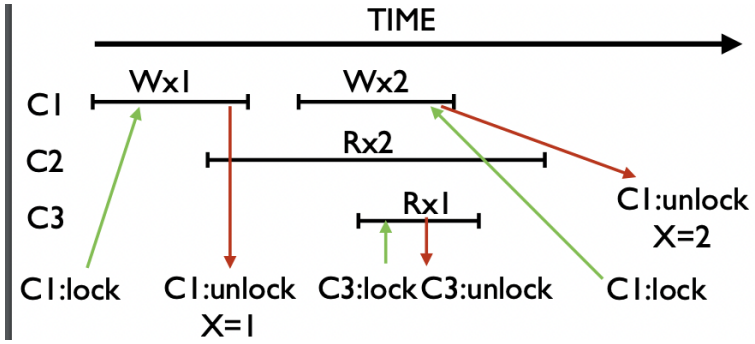


viii. C2 still in flight has not acquired lock and C1 starts another write

ix.



x. C3 starts an op while C2 and C1 are in flight, 3 concurrent operations

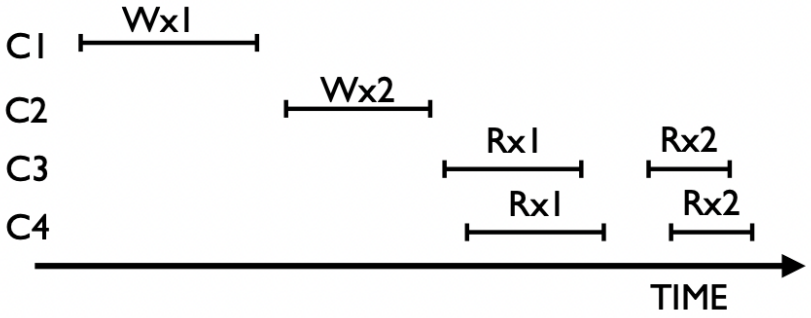


xi.

xii. For whatever reason, C3 gets lock first then C1 then C2
Remember: locks are acquired in random order!

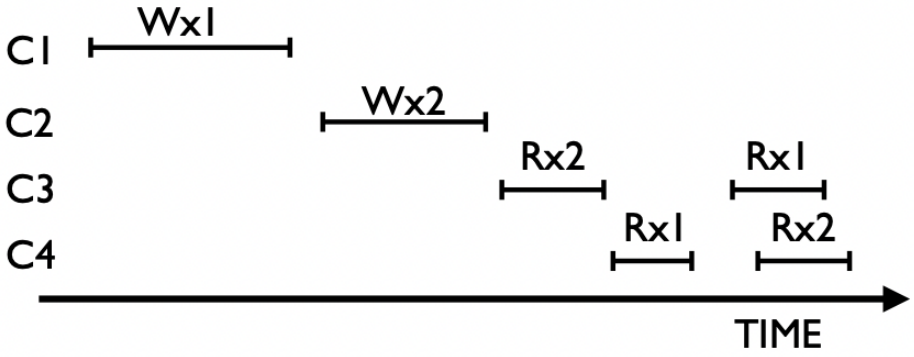
xiii. This would lead to a linear consistent order of
C1: Wx1
C3: Rx1
C1: Wx2
C2: Rx2

3. Sequential consistency
   a. Read/Write operations are executed in some order that respects the order of operations per client
   b. Weaker than linearizability
   c. PNUTS provides sequential consistency per record (called "Timeline consistency")
      i. All clients will see operations on a record in the same order (but operations across records may appear in any order)



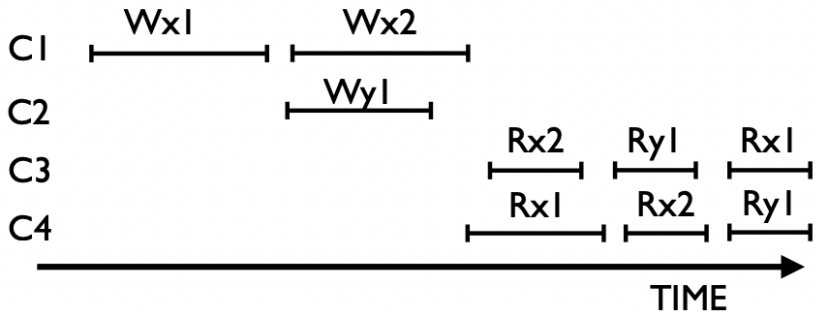
d.

   e. The execution above is sequentially consistent because operations are executed in some order defined by the system that respects the order of operations per client
   f. Possible order
      i. C1: Wx1
      ii. C3: Rx1
      iii. C4: Rx1
      iv. C2: Wx2
      v. C3: Rx2
      vi. C4: Rx2



g.

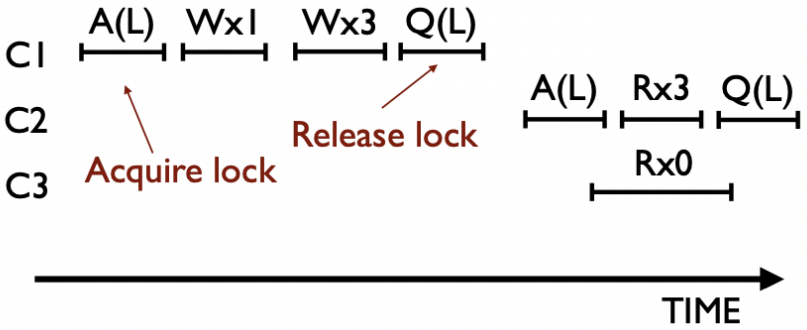
h. The execution above is not sequentially consistent because operations across clients can be reordered but all clients must see the same order



i.

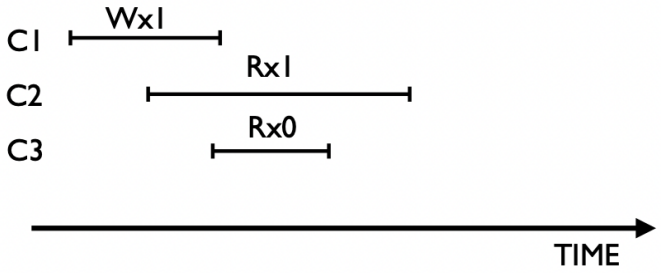
j. The execution above is not sequentially consistent because the global order must respect the order of operations per client
(C3 observes x = 2 before x =1 but C1 sets x = 1 before x = 2)

4. (Lazy) Release Consistency
   a. Relies on explicit use of locks to propagate updates
      i. RC pushes updates upon lock release
      ii. LRC pulls updates when acquiring locks
   b. We've seen it in Distributed Shared Memory (DSM) systems (E.g. TreadMarks)

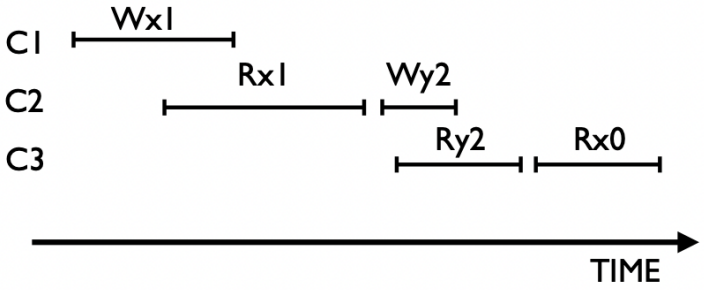


   c.
   d. If program uses lock properly, the execution above is sequentially consistent
   e. Propagation of updates is guaranteed only when the program uses locks
   f. C2 correctly receive the latest value of x
   g. C3 does not use locks and thus observes a stale value

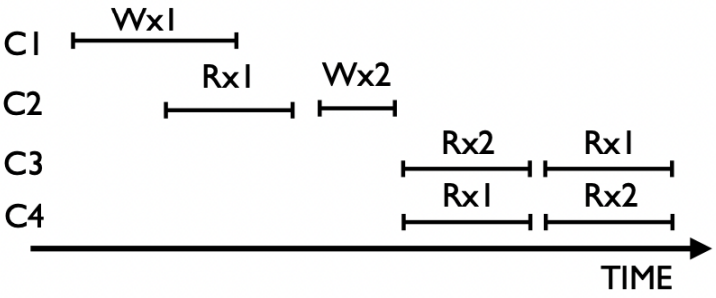
5. Causal Consistency
    a. Read/Write operations that may be causally related are executed in some global order
    b. Causality is defined by the Happened-Before relationship
    c. Weaker than sequential consistency
    d. We've seen it in Distributed Shared Memory (DSM) systems (E.g. TreadMarks)



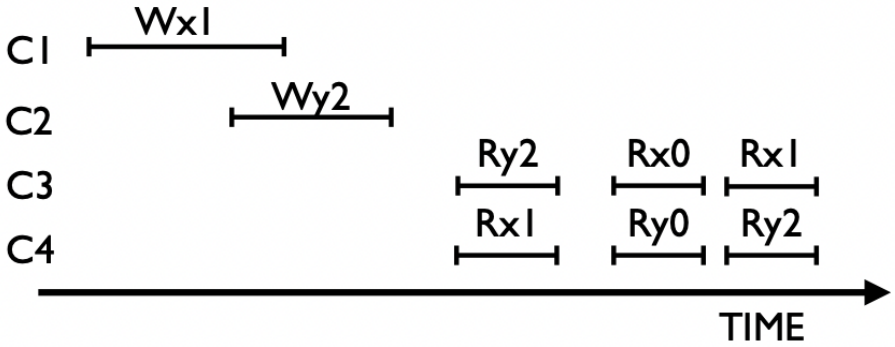
    e.
    f. The execution above is causally consistent between operations - client C2 and C3 may see different values for x



    g.
    h. The execution above is not causally consistent because C3 observed y =2 while C2 set y = 2 after observing x =1
    i. Rx1 may have triggered Wy2 by C2 (causality)
    j. C3 observes y = 2 so it must also observe x = 1



    k.
    l. The execution above is not causally consistent because clients must observe all causally related updates in the same order
        i. C2: Rx1 and C2: Wx2 may be causally related
        ii. The value x = 1 that is read by C2 has been set by C1
        iii. C1: Wx1 and C2: Wx2 may be causally related (by transitivity)

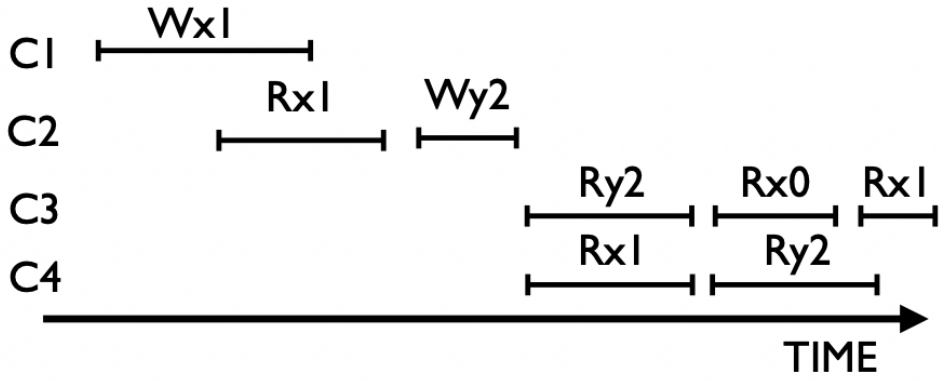


m.

n. The execution above is causally consistent because operations Wx1 and Wy2 are not causally related

o. (Note: this execution is not sequentially consistent)

6. Eventual Consistency
    a. If no new writes, all reads will see the same value after some time
        i. The amount of time required to converge is system specific
        ii. May introduce conflicts that must be resolved
    b. Weaker than causal consistency
    c. We've seen it in PNUTS and Dynamo
        i. Updates on different records may be observed by two PNUTS clients in different orders
        ii. Dynamo supports eventual consistency and uses vector clocks for conflict resolution



d.

e. The execution above is eventually consistent because C3 and C4 may see updates on x and y in a different order

f. (Note: this execution is not causally consistent)