Distributed Systems

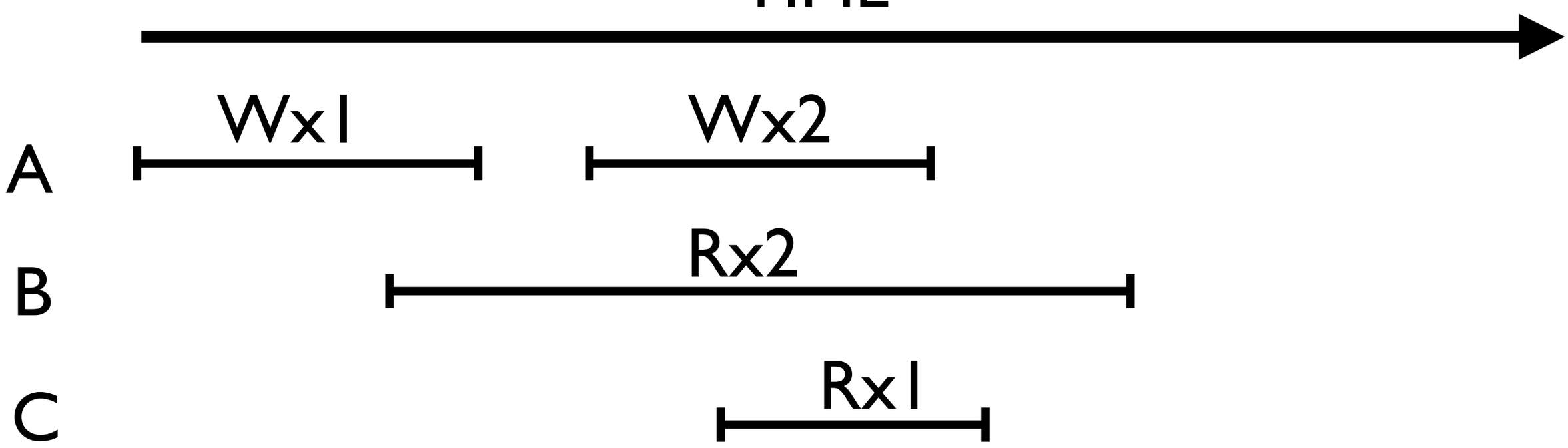
Spring Semester 2020

Lecture 17: Existential Consistency

John Liagouris
liagos@bu.edu

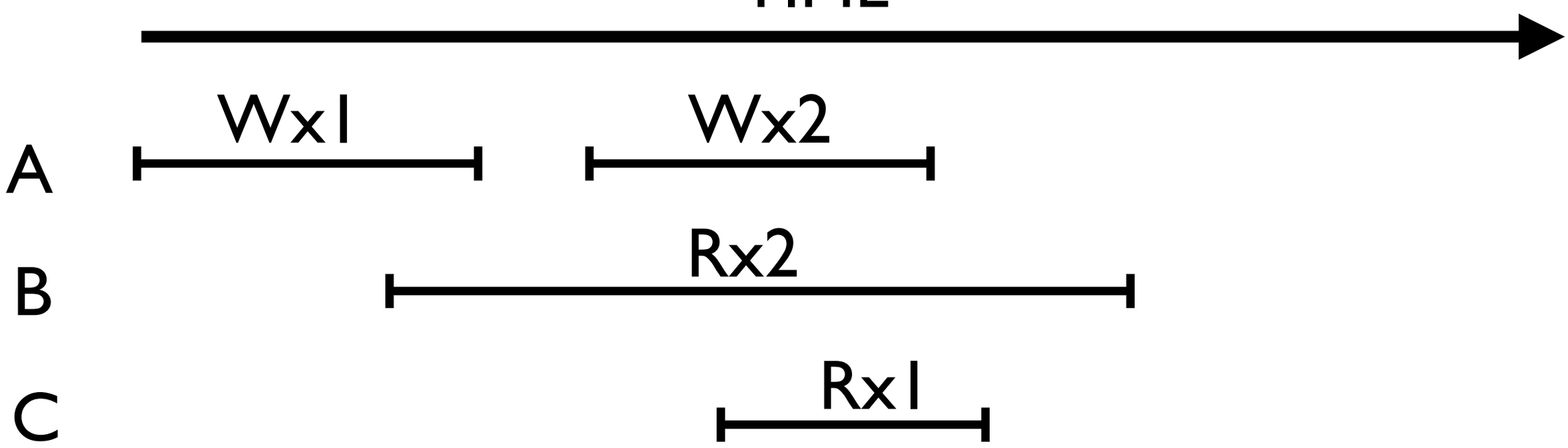
Linearizability

There exists a total order of all operations, that matches real-time (for non-overlapping ops), and in which each read sees most recent preceding write.



Linearizability

There exists a total order of all operations, that matches real-time (for non-overlapping ops), and in which each read sees most recent preceding write.



Linearizable order: Wx1 Rx1 Wx2 Rx2

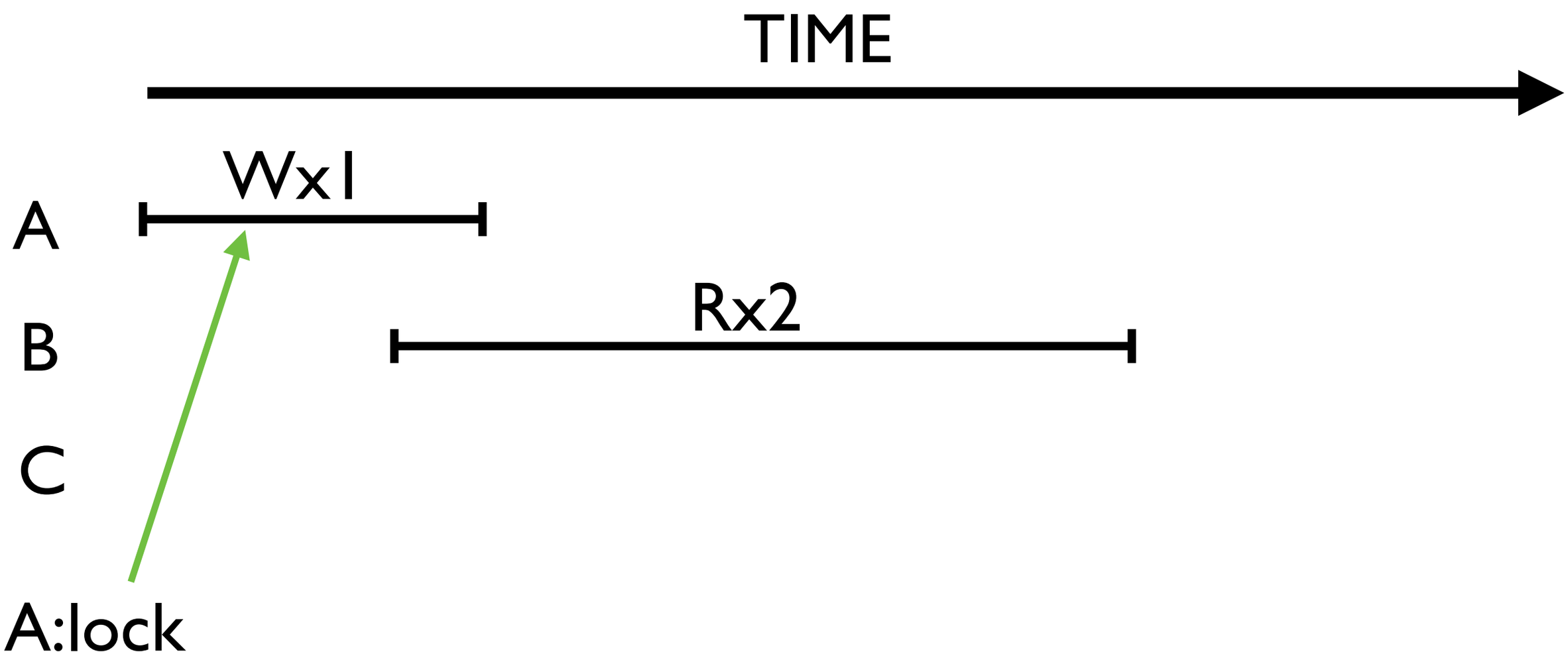
obeys real-time rule — Wx1->Wx2 and Wx1->Rx1

obeys value rule

Linearizability

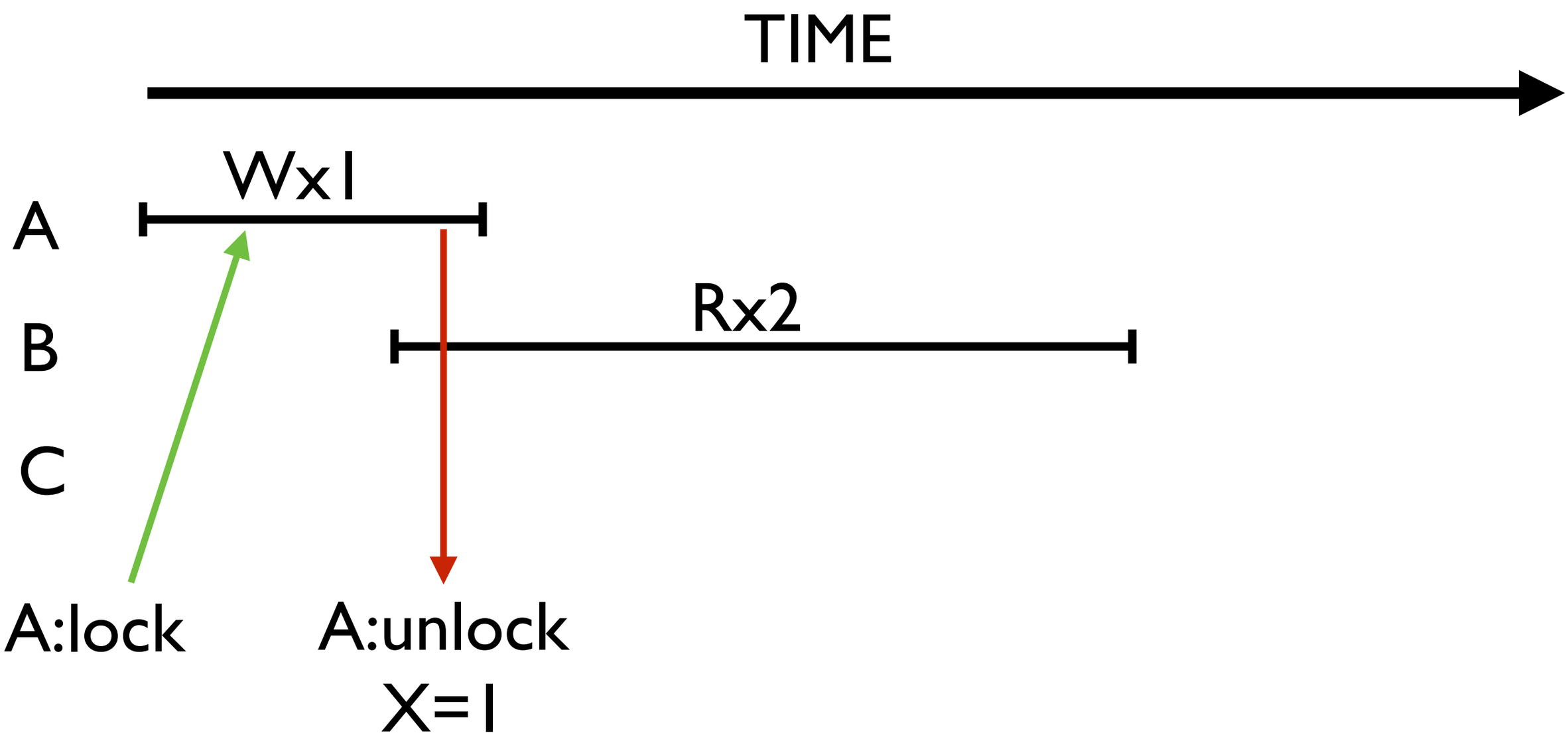
A get's lock (might have been blocked waiting for it)

Linearizability



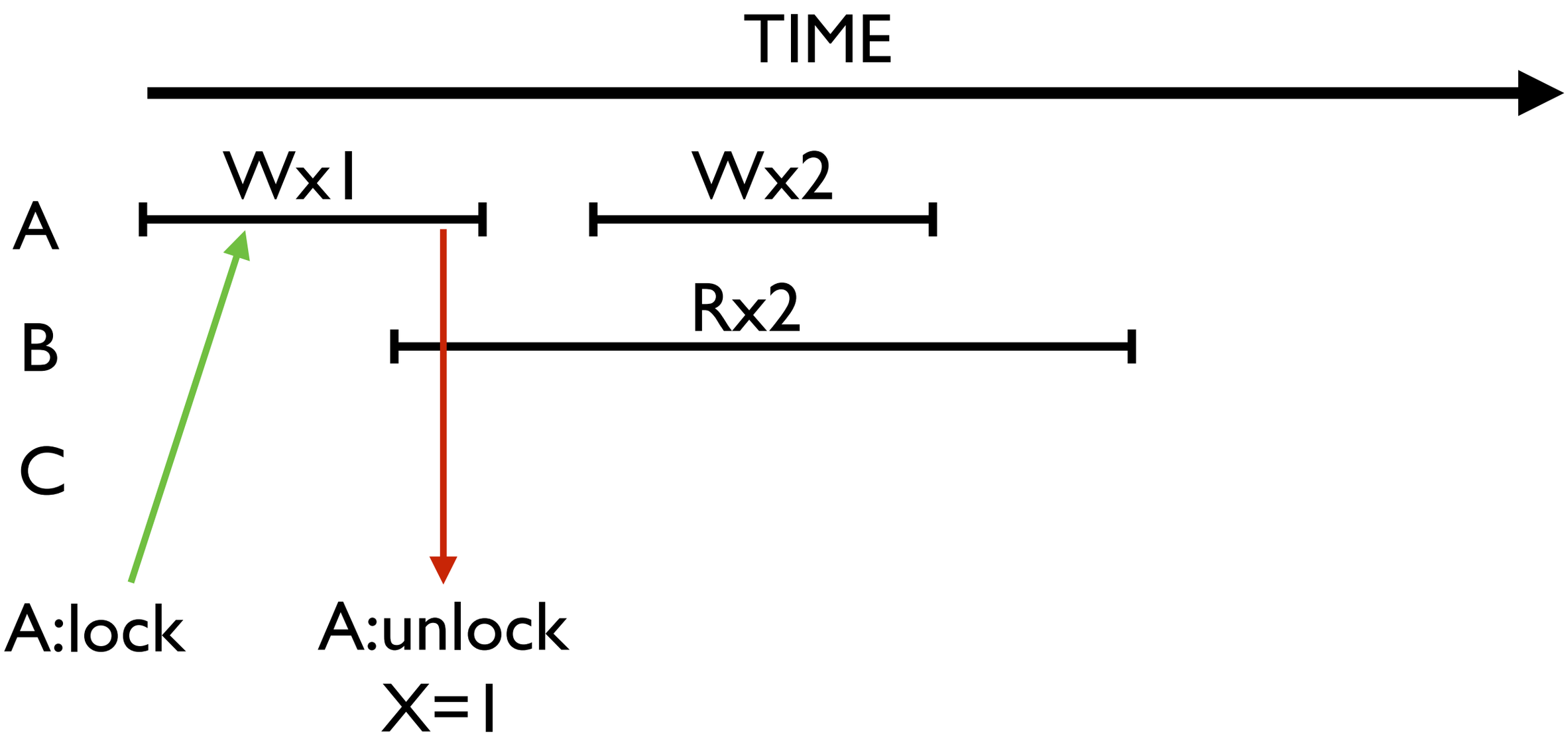
B starts read but gets delayed in the network
(or blocks on lock)

Linearizability



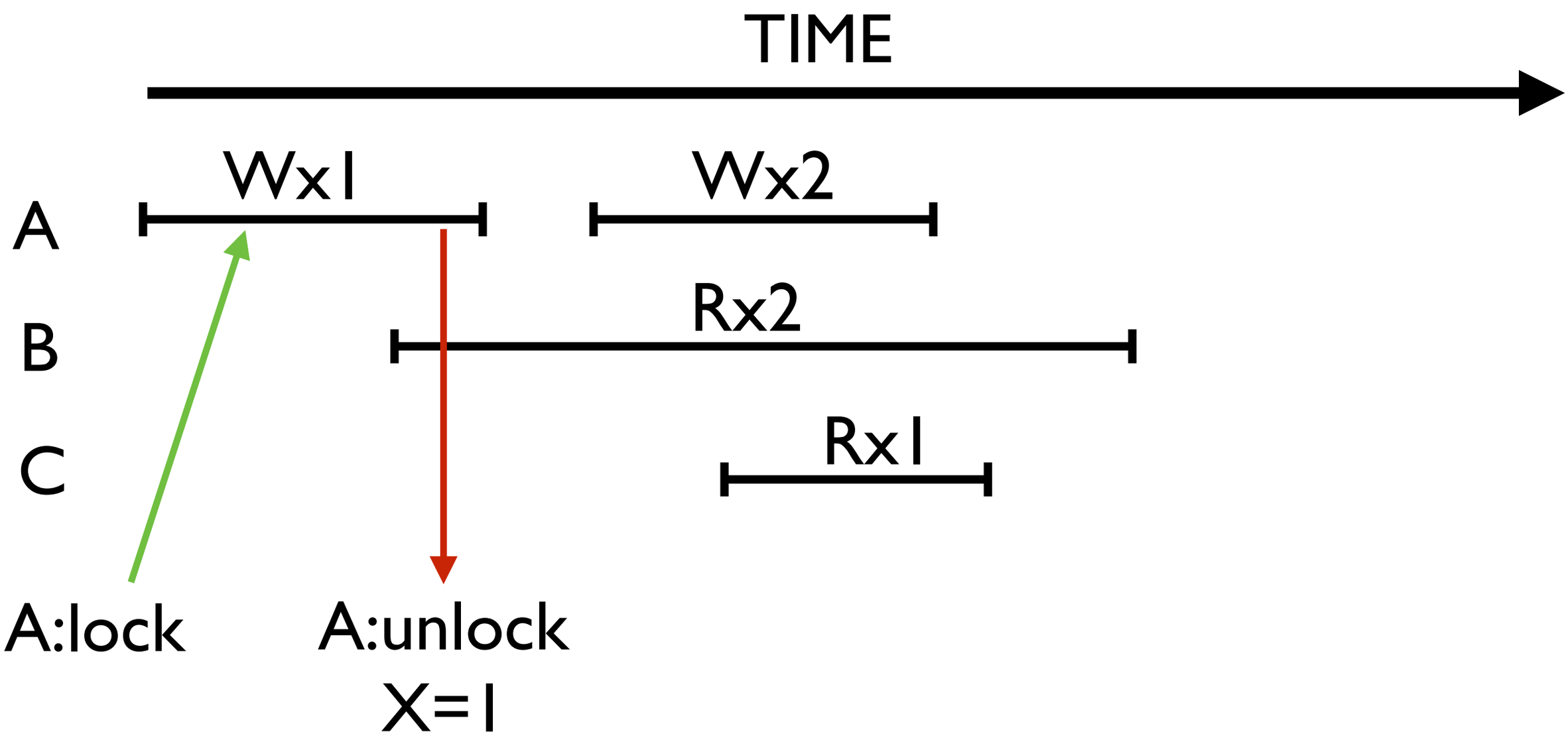
A release lock — think X=1 instantaneously happens

Linearizability



B still in flight has not acquired lock
A starts another write

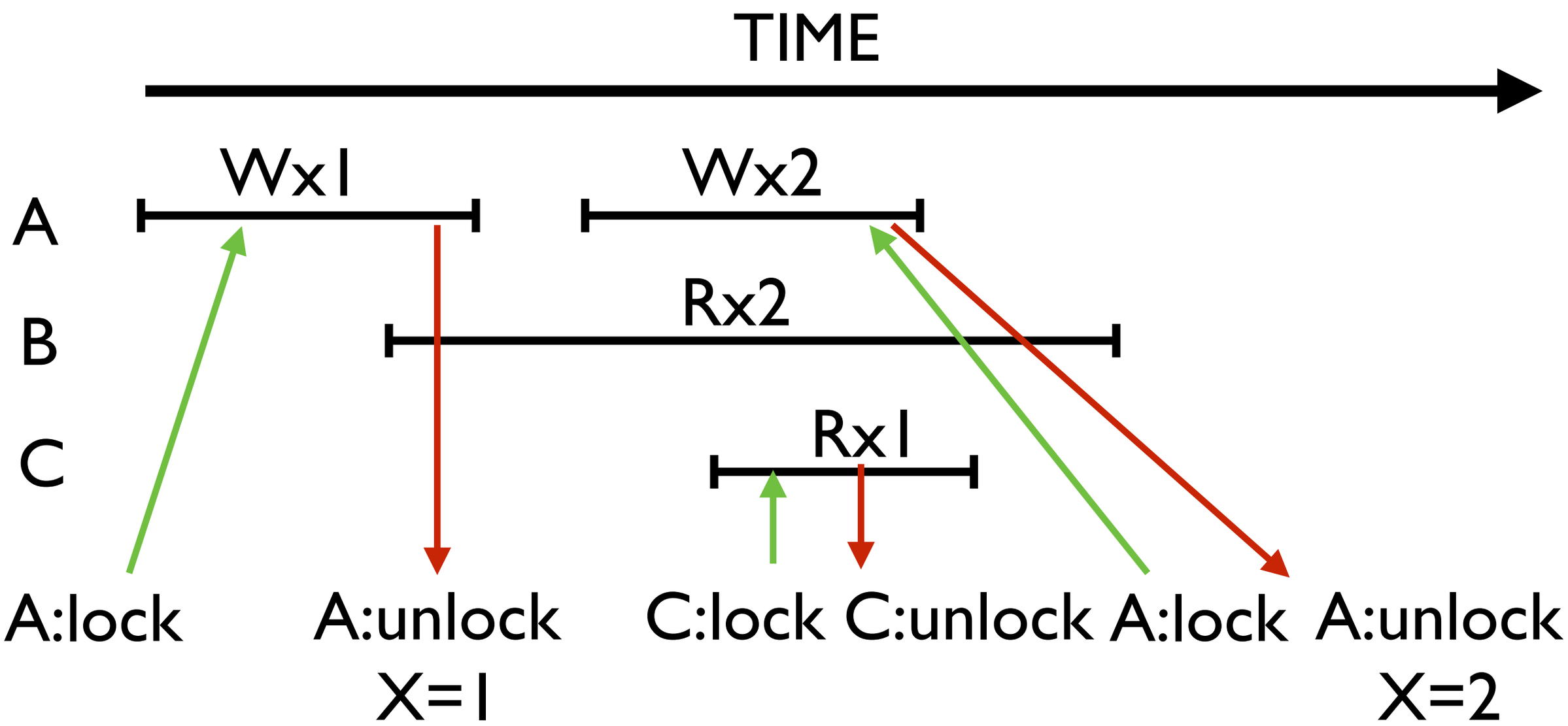
Linearizability



C start's an op while B and A are inflight

3 Concurrent ops

Linearizability



For what ever reason's C gets lock first then A then B —
This would lead to a Linear consistent order of
A:Wx1,C:Rx1,A:Wx2, B:Rx2

Checking Linearizablility

- Tricky — if writes and reads overlap
  - several orderings could be valid must check the results of all overlapping operations and see if they form one of the valid orders
- but strictly subsequent (in "Real Time") operations must see side effects of prior completed operations
- They would like to check "passively"

Linearizability is a "Local Model"

- Can spot violations by looking at traces of individual keys — no need to consider multi-key traces
- Simply evaluating all ops on a key tells you if the linearizable consistency was met

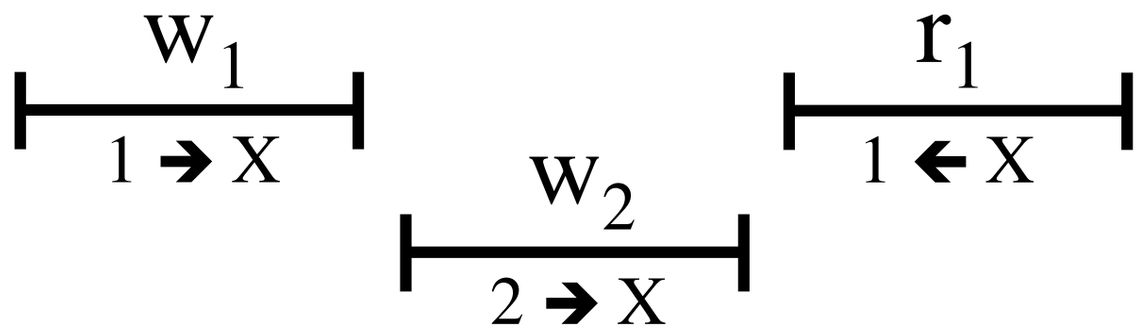
So how do they do it

- Randomly select a small subset of keys
- Trace all operations (reads and writes) on the key
- Analyze the trace offline for violations

The Trace

- **key**, **r/w**, **value** hash, **start** time, **end** time, …

Each read can be matched to its write and the operations can be ordered based on start and end times — overlaps/concurrency can be detected

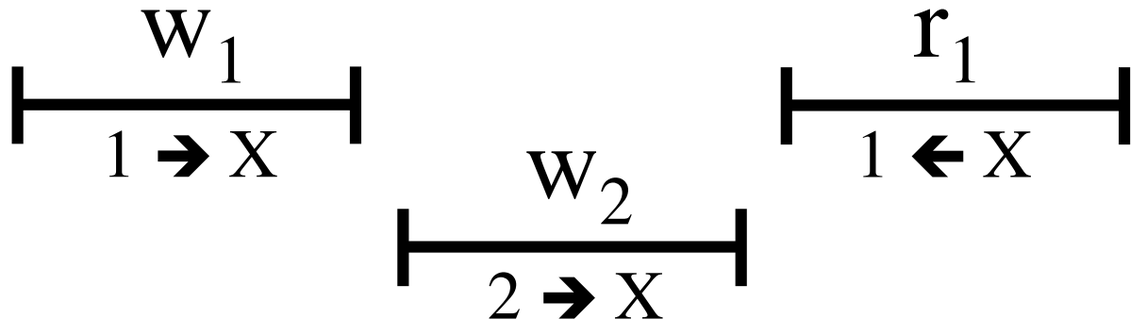


(a) Execution.

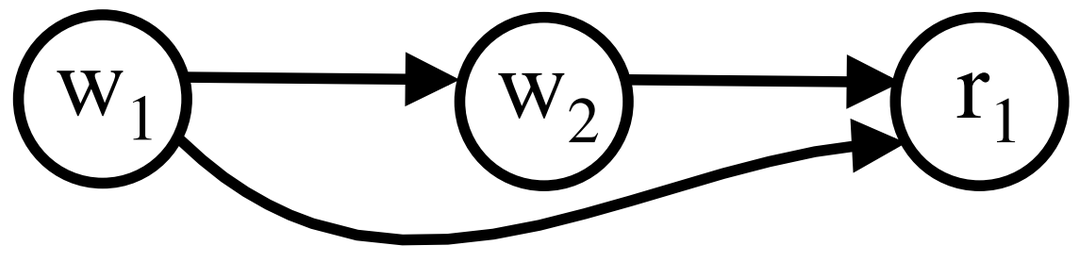
W1:X=1, strictly precedes W2: X=2, which strictly precedes read R1, that sees the value of W1

Graph construction

- node for each read and write
- edges for time ordering
  if op2.start > op1.end
  then edge(op1,op2)
- edges for value ordering
  if op2.read == op1.write
  then edge(op1,op2)



(a) Execution.

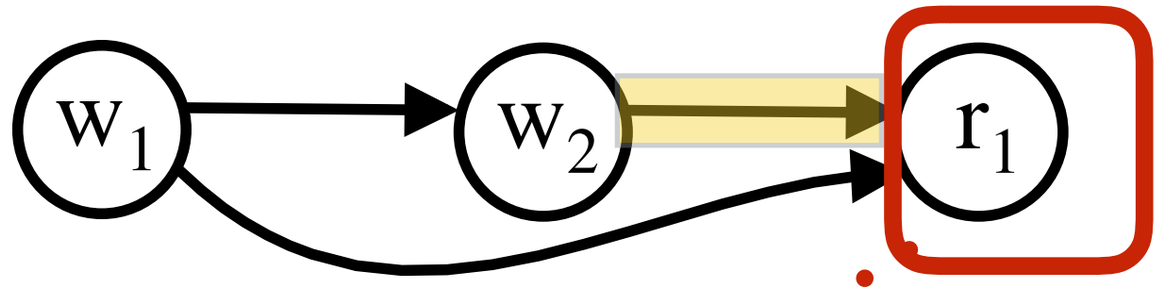


(b) Graph before merge.

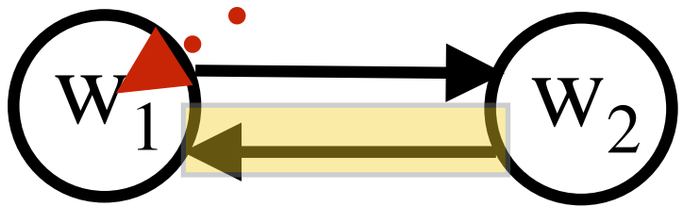
"Linearizability: requires that there exits a total order that is **legal**, and agrees with the real-time ordering of ops. A total order is legal if it a read is ordered after the write it observed with no other writes between"

Graph construction

- Merge: put read nodes into their write nodes.
- If read's go behind a more recent write we will get a cycle
  - NO loop linearizable
  - Loop no legal total order



(b) Graph before merge.

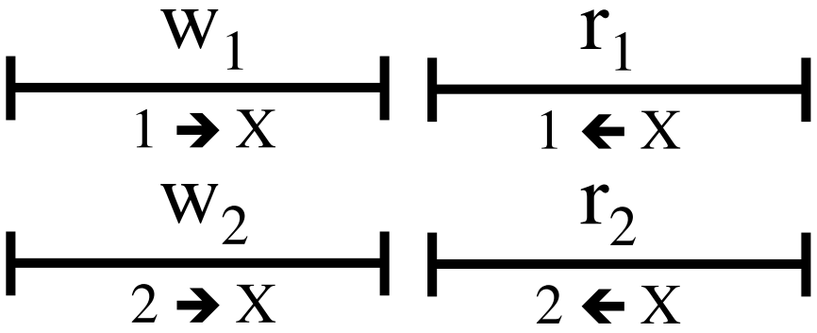


(c) Graph after merge.

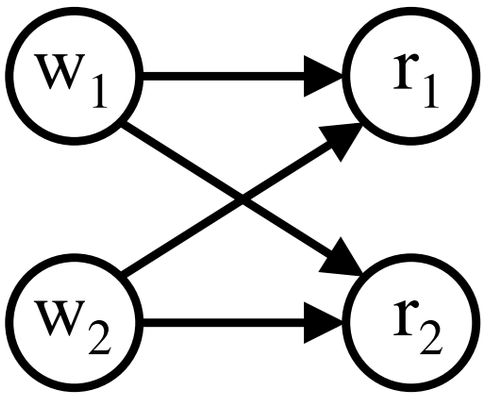
STALE READ ANOMALLY

"Linearizability: requires that there exits a total oder that is **legal**, and agrees with the real-time ordering of ops. A total order is legal if it a read is ordered after the write it observed with **no other writes between**"

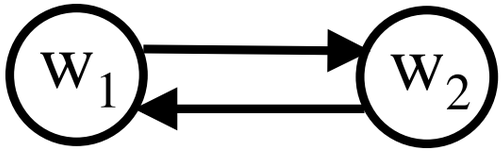
Total Order Anomaly



(a) Execution



(b) Graph before merge.



(c) Graph after merge.

- Linearizability requires that there is a single consistent total order for all operations on an object
- All observed side effects of concurrent operations must agree on a single order
- eg. either X=1,X=2 or X=2,X=1 is ok but not both!

Will their technique make mistakes?

Will their technique make mistakes?

Yes if clocks are wrong.

Wx1 Wx2 Rx1

Will their technique make mistakes?

Their approach does not check non-sampled keys

Why can't things like causal consistency not be checked?

this is legal under causal consistency:

```
C0: Wx1
C1:   Rx1
C2:     Rx0
```

this is illegal:

```
C0: Wx1
C1:   Rx1 Wy2
C2:         Ry2 Rx0
```

Online Consistency Checking Practical Consistency

- "Principled" approach too expensive for realtime use — yet would really like to know how this system is doing and detect when things are going wrong
- So instead take a much simpler approach — get see if caches are consistent ;-)

Results

	Anomalous Reads	Percentage Of Filtered (241M)	Percentage Of Overall (937M)
Linearizable	3,628	0.00151%	0.00039%
Stale Read	3,399	0.00141%	0.00036%
Total Order	229	0.00010%	0.00002%
Per-object Seq	607	0.00025%	0.00006%
Per-User	378	0.00016%	0.00004%
Read-after-Write			
Global	3,399	0.00141%	0.00036%
Per-Region	1,558	0.00065%	0.00017%
Per-Cluster	519	0.00022%	0.00006%

Wow only 0.004% of reads show linearizability anomalies

Results

	Anomalous Reads	Percentage Of Filtered (241M)	Percentage Of Overall (937M)
Linearizable	3,628	0.00151%	0.00039%
Stale Read	3,399	0.00141%	0.00036%
Total Order	229	0.00010%	0.00002%
Per-object Seq	607	0.00025%	0.00006%
Per-User	378	0.00016%	0.00004%
Read-after-Write			
Global	3,399	0.00141%	0.00036%
Per-Region	1,558	0.00065%	0.00017%
Per-Cluster	519	0.00022%	0.00006%

Wow only 0.004% of reads show linearizability anomalies

- Few writes (1 in 450)
- Locality (consistency better within writer's cache)
- Fast invalidation messages

What do the results mean?

Conclusion

- low-cost checking technique
- low-consistency design can provide remarkably high consistency!
  - i.e. they rarely display stale data to the user — sheds doubt on need for costly consistent designs
- consistency not always a correctness question — freshness of info shown to users
- FB specific results — but maybe representative of many web-sites