Sequential Consistency & Linearizability

CS4405 – Analysis of Concurrent and Distributed Programs

Burcu Kulahcioglu Ozkan

TU Delft

Correctness for concurrent objects

Problem setting (distributed databases):

- Database consists of objects that can be read/written by transactions
  - An object is defined by the set of operations on it and by the behavior of the object when these operations are invoked sequentially (sequential specifications*)
- Clients interact with databases using transactions
- We capture what happens in a database by a history
- A history is partial order over the operations of a set of transactions

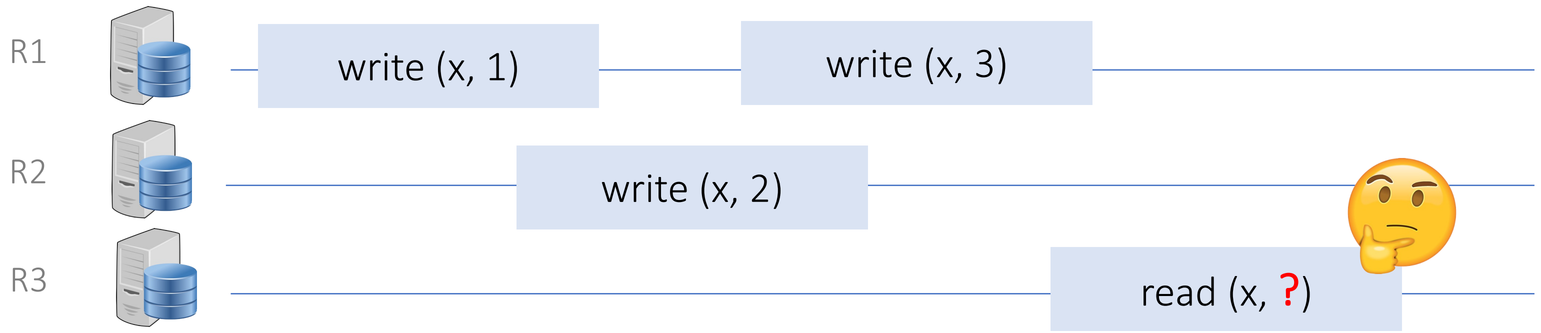
- In this lecture: Each transaction consists of a single operation
- In the next lecture, transactions with atomic visibility of a set of operations

Correctness for concurrent objects

Concurrent accesses to multiple copies of the object

Consistency condition defines which concurrent operation executions are considered correct

Consistency model is a contract between programmer and replicated system, i.e., it specifies the consistency between replicas and what can be observed as possible results of operations.



Strong consistency models

- Sequential consistency (SC) (or serializability for transactional accesses) used in 1970s in the database context

- Linearizability: Stronger consistency condition (SC + real time requirements)

Next lecture: Weak consistency models

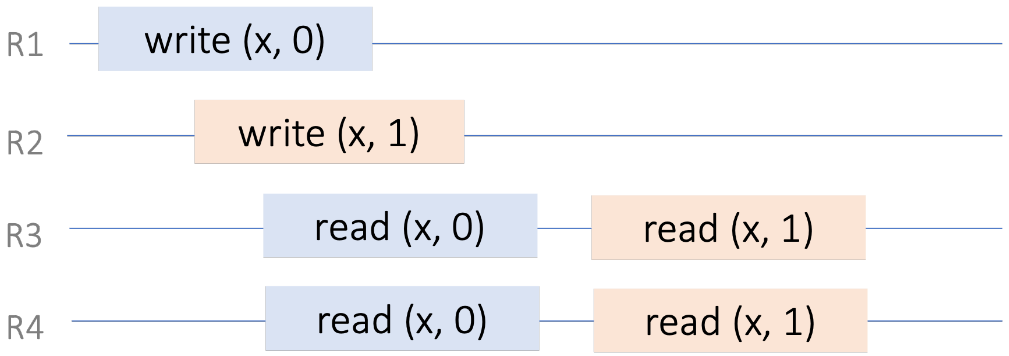
How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs

LESLIE LAMPORT

IEEE, 1979

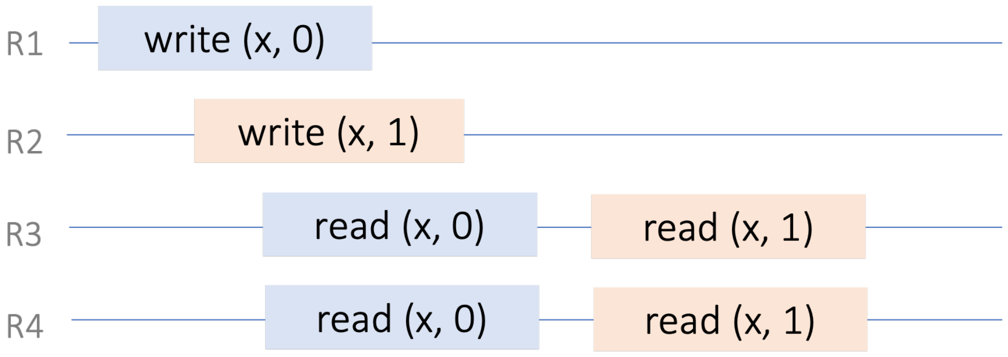
Sequential Consistency (SC)

- Operations appear to take place in some total order:
  - All servers execute all ops in some identical sequential order
  - The order preserves each client's own local ordering

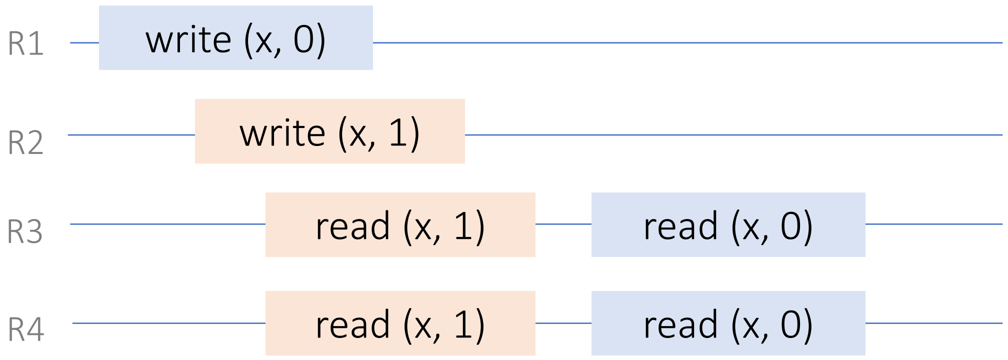


Sequential Consistency

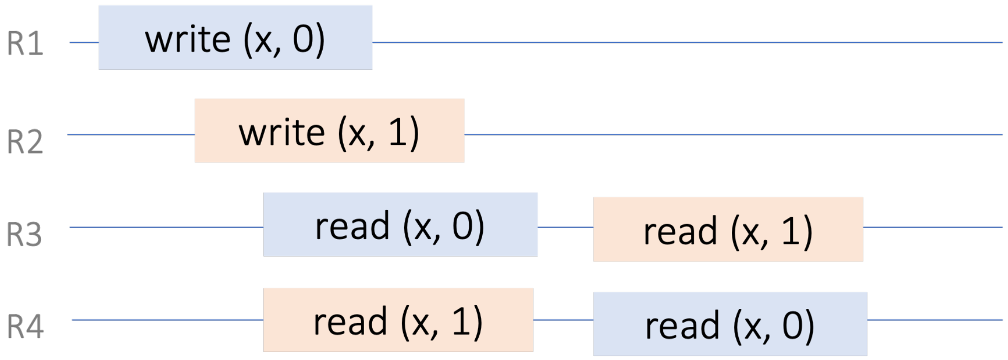
Sequentially consistent



Also sequentially consistent



Not sequentially consistent



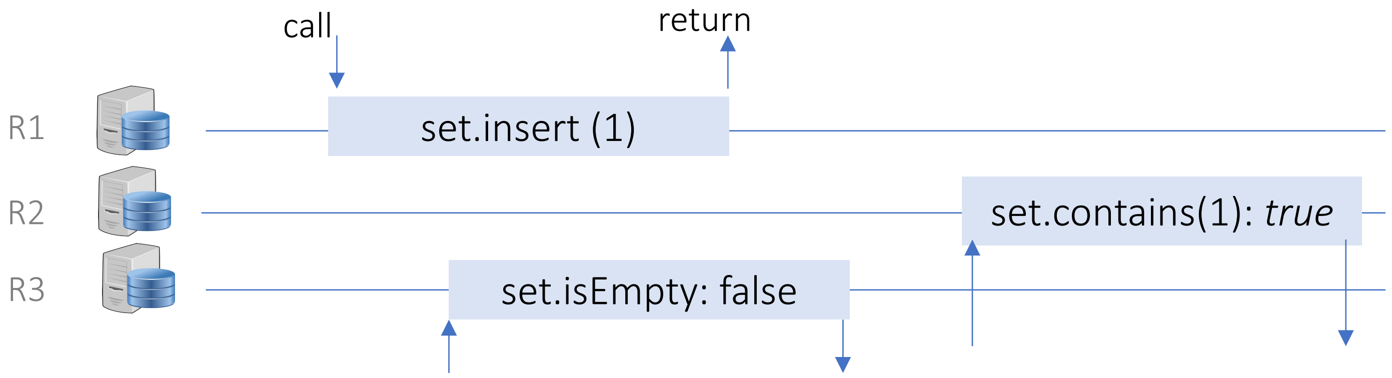
Linearizability

Linearizability: A Correctness Condition for Concurrent Objects

MAURICE P. HERLIHY and JEANNETTE M. WING
Carnegie Mellon University

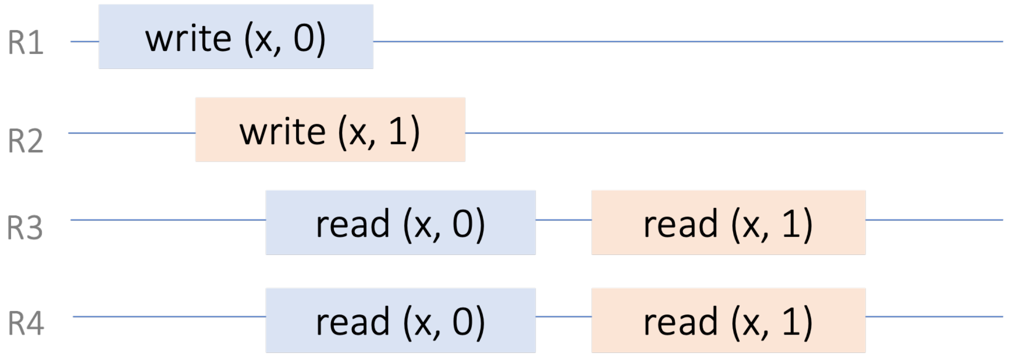
TOPLAS, 1990

- SC + preserve real-time order
  - All servers execute all ops in some identical sequential order
  - The order preserves each client's own local ordering
  - The order preserves real-time guarantee
- Operation calls appear to take effect one at a time, each operation takes effect between the point at which it is called and when it returns
  - As soon as writes complete successfully, the result is immediately replicated to all nodes atomically and is made available to reads

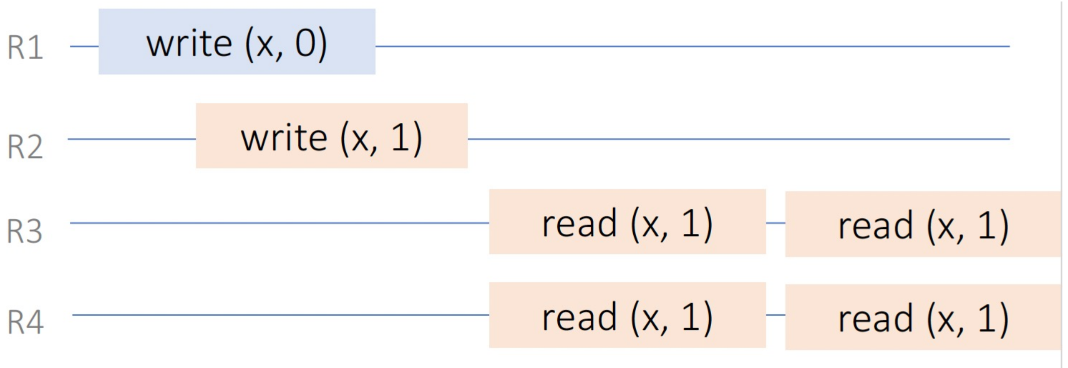


Linearizability

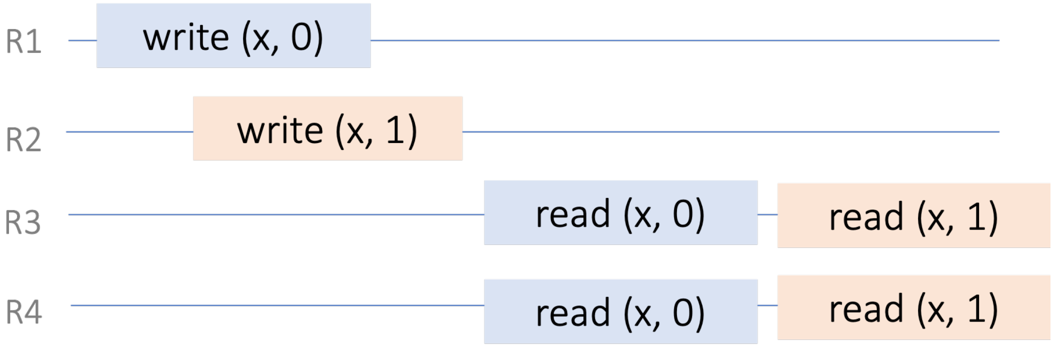
Linearizable



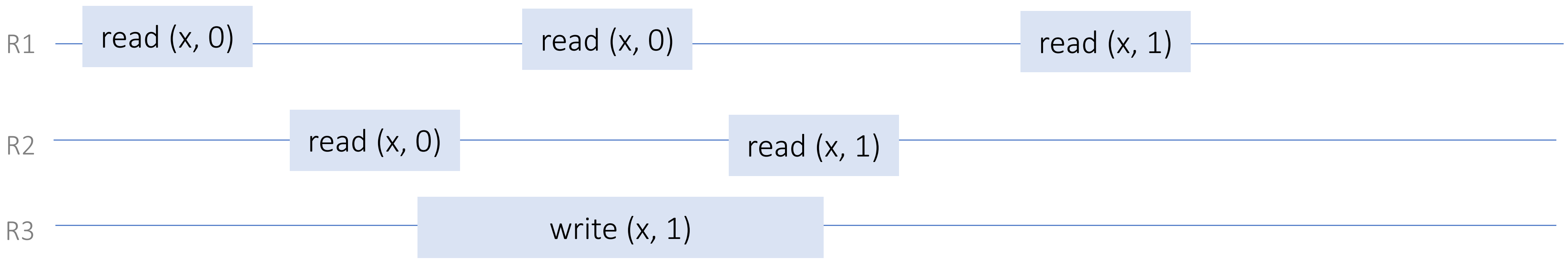
Also linearizable



Not linearizable



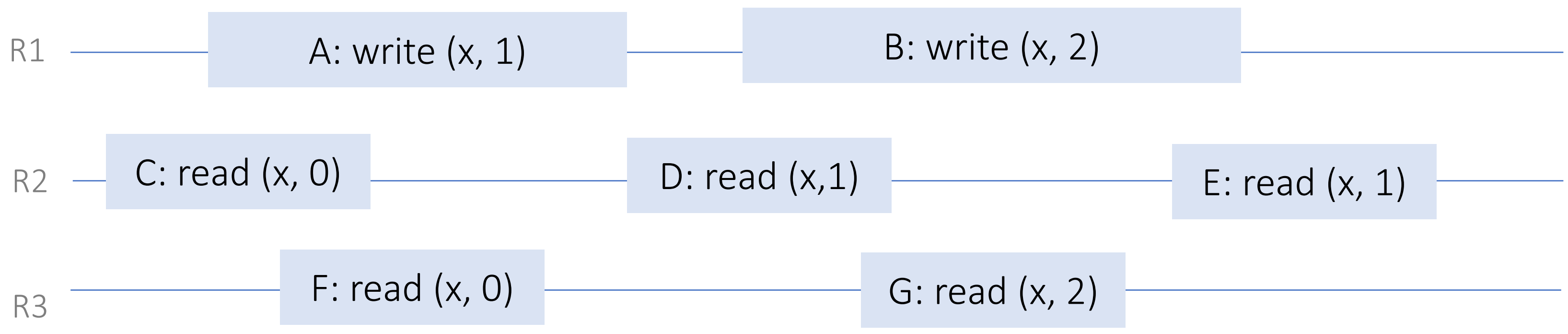
Linearizability: Example



Q: Is the execution above linearizable?

A: Yes, there is a *total order* of operations *that conform to their real time order* and *satisfy the specification* of the data variable x.

Linearizability: Example



Q: Is the execution above linearizable?

A: No, in a linearizable system, it is not possible the read operation E to read the value **1**.

Testing for Linearizability

Term Project

A concurrent datatype C is linearizable wrt S if every history h of C is linearizable wrt S.

Test setting:

- Distributed database X claims to provide linearizable consistency
- Testing database X:
  - Run some client transactions on the database
  - (Optional) Insert network or process faults during the execution
  - Collect execution histories of the test executions (with call and return events)
- Checking the linearizability of histories:
  - Check each history whether it satisfies linearizability or it violates linearizability
  - E.g., using Knossos linearizability checker

How to check linearizability of histories?

- Checking linearizability is an NP-complete problem (Gibbons & Korach, 1997)
  - Inherently difficult to implement a checker
- Automated model checking techniques apply exhaustive state space exploration
  - Beware of state space explosion problem
  - The problem is to find ways of pruning a huge search space: in the worst case, its size is $O(N!)$ where N is the length of the run of a concurrent system.

How to check linearizability of histories?

- Wing & Gong's linearizability algorithm

Input: A complete history h and a sequential specification object S with an undo operation.
Returns: true iff h is linearizable.

```
def isLinearizable(h, S) : Boolean = {
  if (h is empty) return true
  else{
    for each minimal operation op {
      // try linearizing op first
      let res be the result of op in h
      run op on S
      if (S gives result res && isLinearizable(h – op, S)) return true
      else undo op on S
    }
    return false
  }
}
```

an op is minimal if no return is before the call of op, i.e., it could be linearized first

Lowe's algorithm applies optimizations on top of WG algorithm

Testing and Verifying Concurrent Objects. J.M. Wing, C. Gong, C. J. Parallel Distrib. Comput. 1993

Testing for Linearizability. G. Lowe. Concurrency and Computation: Practice and Experience. 2017

How to check linearizability of histories?

- Knossos linearizability checking tool (used in the Jepsen test system)
  - An implementation of the Wing and Gong linearizability checker, plus Lowe's extensions
  - Used for checking linearizability of many in-production distributed systems
  - Bugs found in several distributed key/value stores

1 Knossos tablet Co 907, listing sheep, goats, pigs and cattle

https://github.com/jepsen-io/knossos

Some references for further reading

- Testing and verifying concurrent objects.
  J.M. Wing, C. Gong, C. J. Parallel Distrib. Comput. 1993
- Experience with Model Checking Linearizability.
  M. T. Vechev, E. Yahav, G. Yorsh. SPIN 2009
- Line-up: a complete and automatic linearizability checker.
  S. Burckhardt, C. Dern, M. Musuvathi, R. Tan. PLDI 2010.
- Model Checking of Linearizability of Concurrent List Implementations.
  P. Cerný, A. Radhakrishna, D. Zufferey, S. Chaudhuri, R. Alur. CAV 2010.
- Automatically Proving Linearizability.
  V. Vafeiadis. CAV 2010
- Faster Linearizability Checking via P-Compositionality.
  A. Horn, D. Kroening. FORTE 2015.
- Testing for linearizability
  G. Lowe. Concurrency and Computation: Practice and Experience. 2017