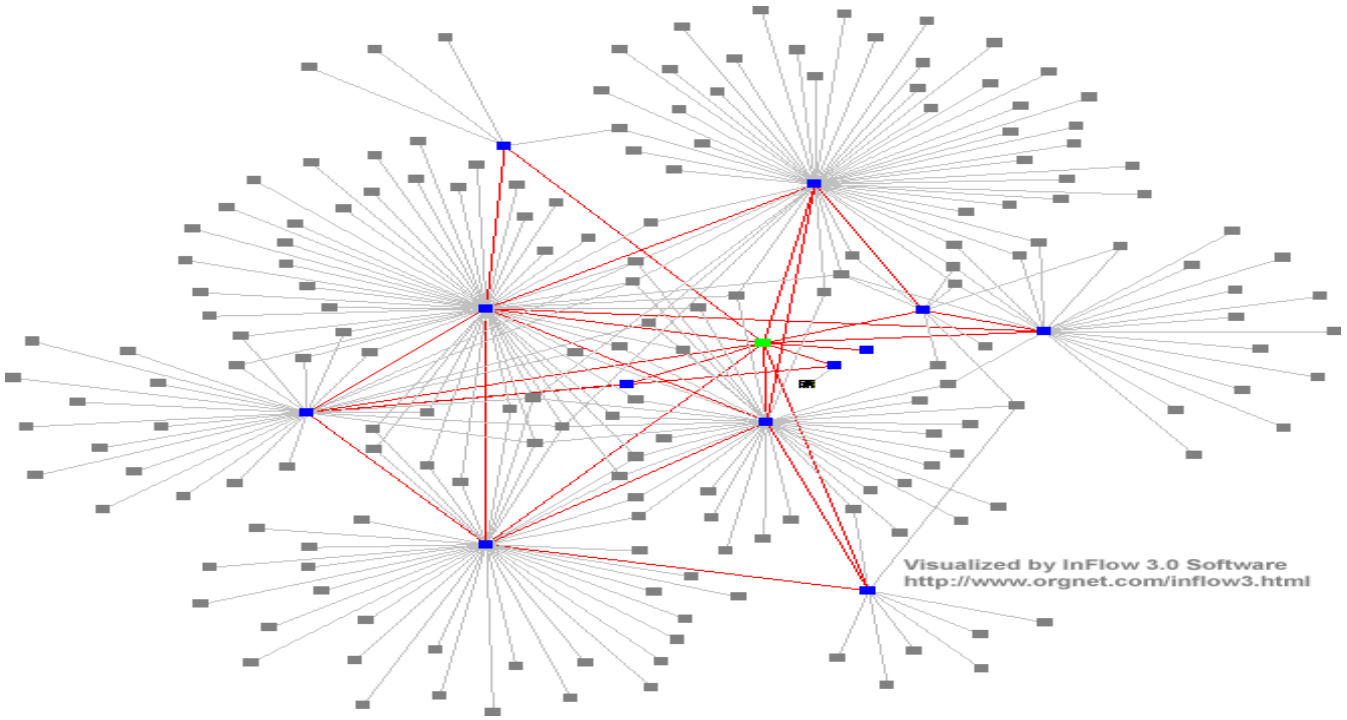
DCS Transaction Issue Part-2

Linearizablity and Consistency



Dr. Sunny Jeong. **spjeong@uic.edu.cn**

Special thanks to Dr. HJ Lee

Outline

- Linearizability
- Consistency
- Consistency protocols
- Parallel vs. concurrent Computing

Concurrency and Linearizability

- Unlike parallelism, not always running faster.

- Useful for:
  - *Managing replicas*
  - *App responsiveness*
    - Example: Respond to GUI events in one thread while another thread is performing an expensive computation
  - *Processor utilization* (mask I/O latency)
    - If 1 thread is stuck working, others have something else to do
  - *Failure isolation*
    - Convenient structure if want to interleave multiple tasks and do not want an exception in one to stop the other

1. **Linearizability** (strict Consistency)

- When multiple nodes are concurrently accessing replicas, how can we define consistency?

- > **Concept of Linearizability**
  - Every operation takes effect **atomically**
  - All operations behave as if executed **on a single copy**
  - Every operation returns an **“up-to-date” value** (Storing consistency“)

Linearizability (Strongest)

Definition:
Operations appear to occur in the exact real-time order they were invoked.

Conditions:
Each operation appears instantaneous (atomic).All nodes observe operations in the same real-time order.

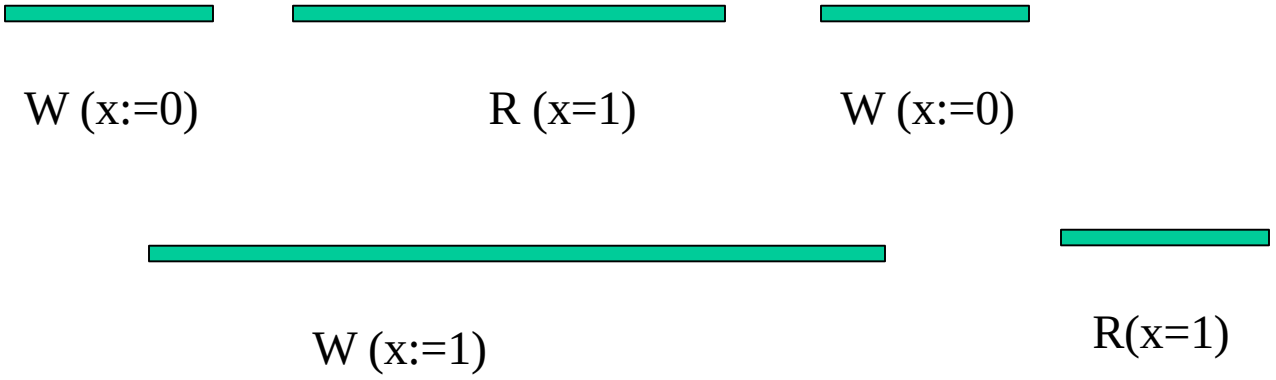
Example:
If Alice's write happens before Bob's read in real time, Bob must see Alice's write.

Performance:
Slower but provides the most intuitive and correct behavior.

Linearizability (strict Consistency)

Linearizability is a correctness criterion for *concurrent object* (Herlihy & Wing, ACM TOPLAS 1990). It provides the illusion that each operation on the object takes effect in zero time, and the result is "equivalent to" some legal sequential computation.

(Definition of consistency for concurrent systems)

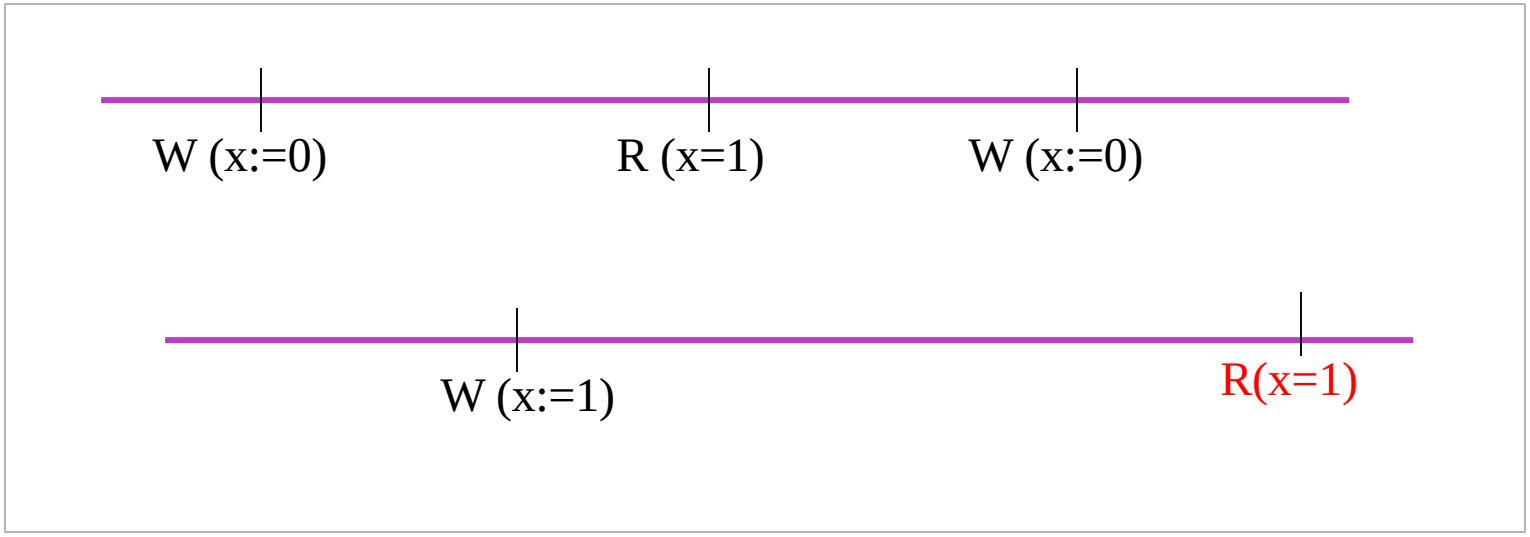


Is this case a linearizability?

Linearizability (strict Consistency)

A trace is *consistent*, when every read returns the *latest* value written into the shared
variable preceding that read operation. A trace is **linearizable**, when (1) it is consistent

, and (2) the **temporal ordering** among the reads and writes is respected.



Is this case a linearizability? -> violated

2. Sequential Consistency (Moderate)

Definition: Operations appear in the same total order for all processes, but not necessarily in real-time order.

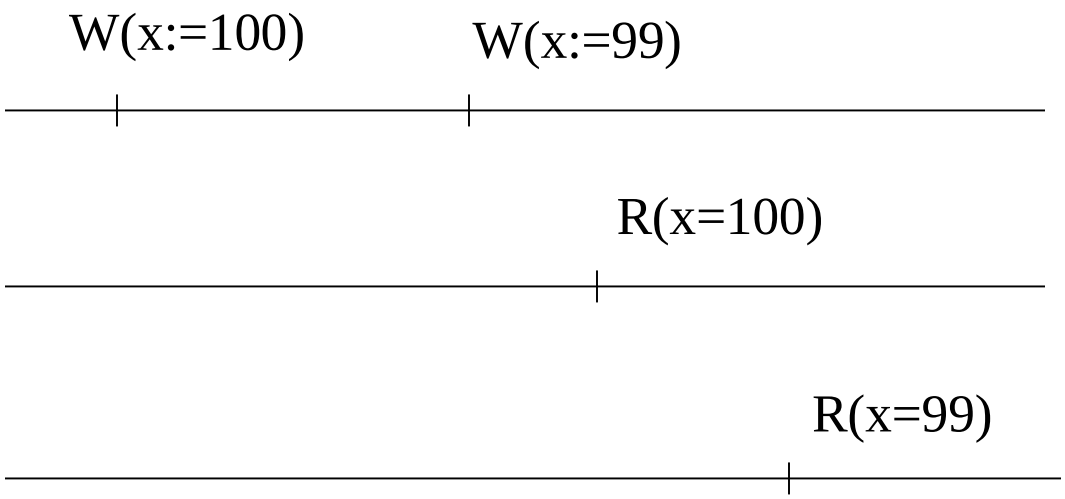
Conditions: All processes see the same interleaving of operations. Real-time constraints are not enforced.

Example: Bob might see his own write before Alice's even if Alice's came first in real-time.

Performance: More scalable than linearizability but less intuitive.

Sequential consistency

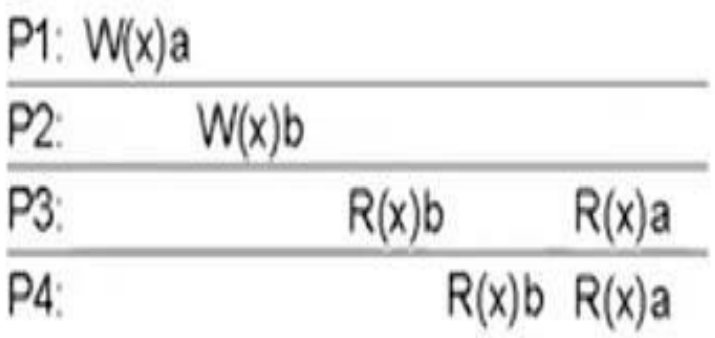
A consistent trace allows some interleaving of local temporal event orders across different replicas



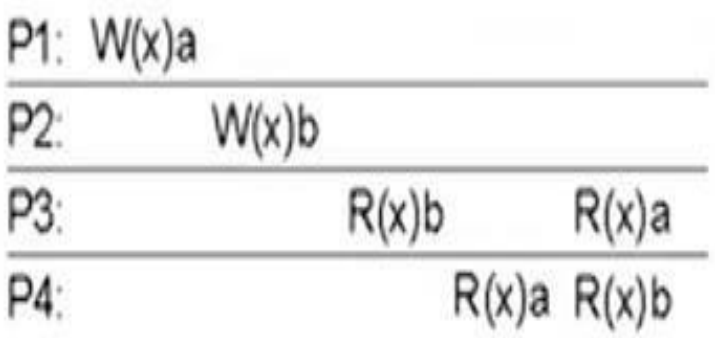
The result of any execution is the same as if all processor operations were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified **by its program** (Time does not play any role)

Sequential consistency

Which one is sequential consistency satisfied here?



(a)



(b)

3. Causal Consistency (Weakest)

Definition:
Causally related operations must be seen in the same order by all processes, but unrelated operations can be reordered.

Conditions:
Only operations with cause-effect relationships must be ordered.

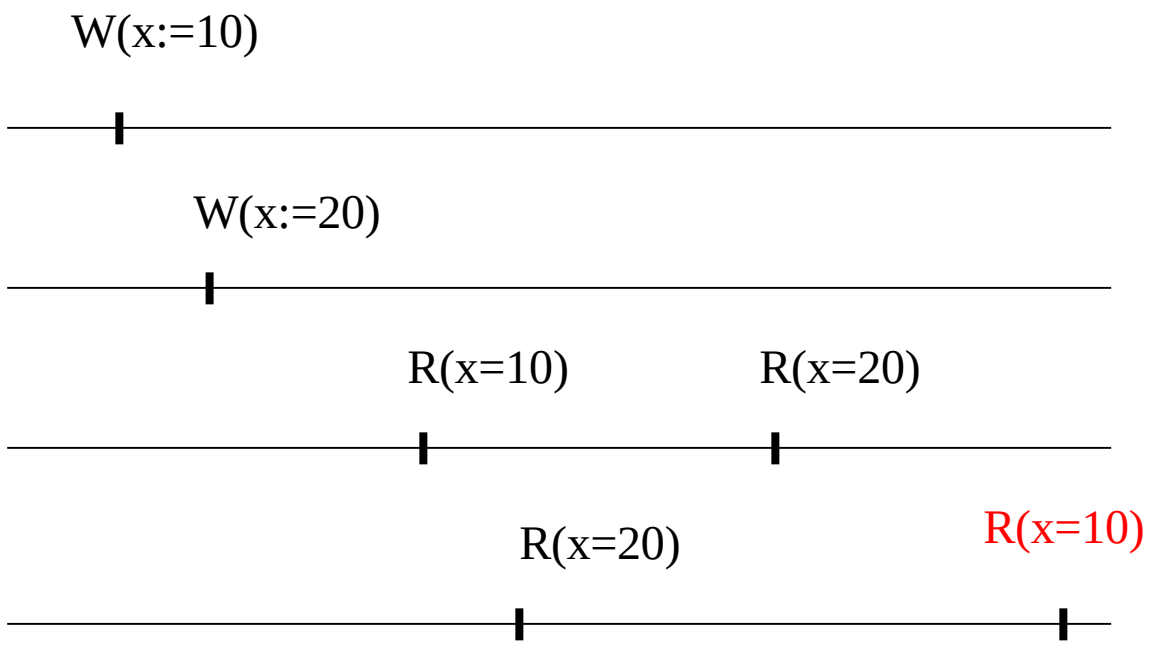
Example:
If Alice comments and Bob replies to her comment, Bob's reply must follow Alice’s comment. But two independent replies might appear in different orders to different users.

Performance:
High performance, good for social media or collaborative apps

Causal consistency

All writes that are *causally related* must be seen by every process in the same order.

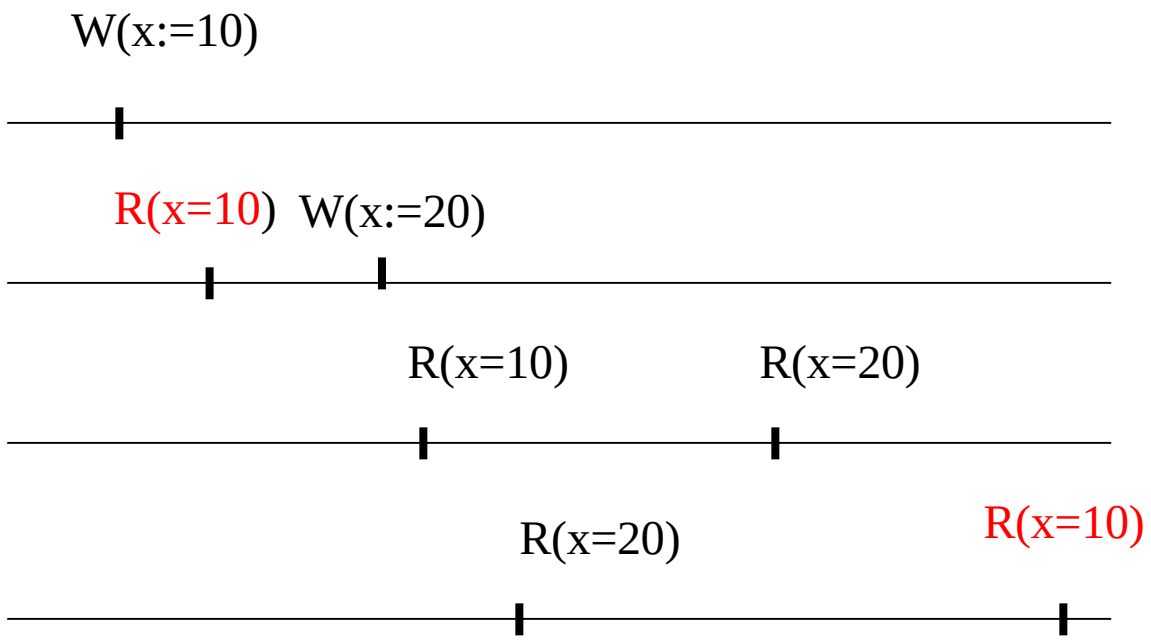
a=1+a, b=a*1, c=2+i



Causal consistency

All writes that are *causally related* must be seen by every process in the same order.

What if there is R(x=10) on the P2



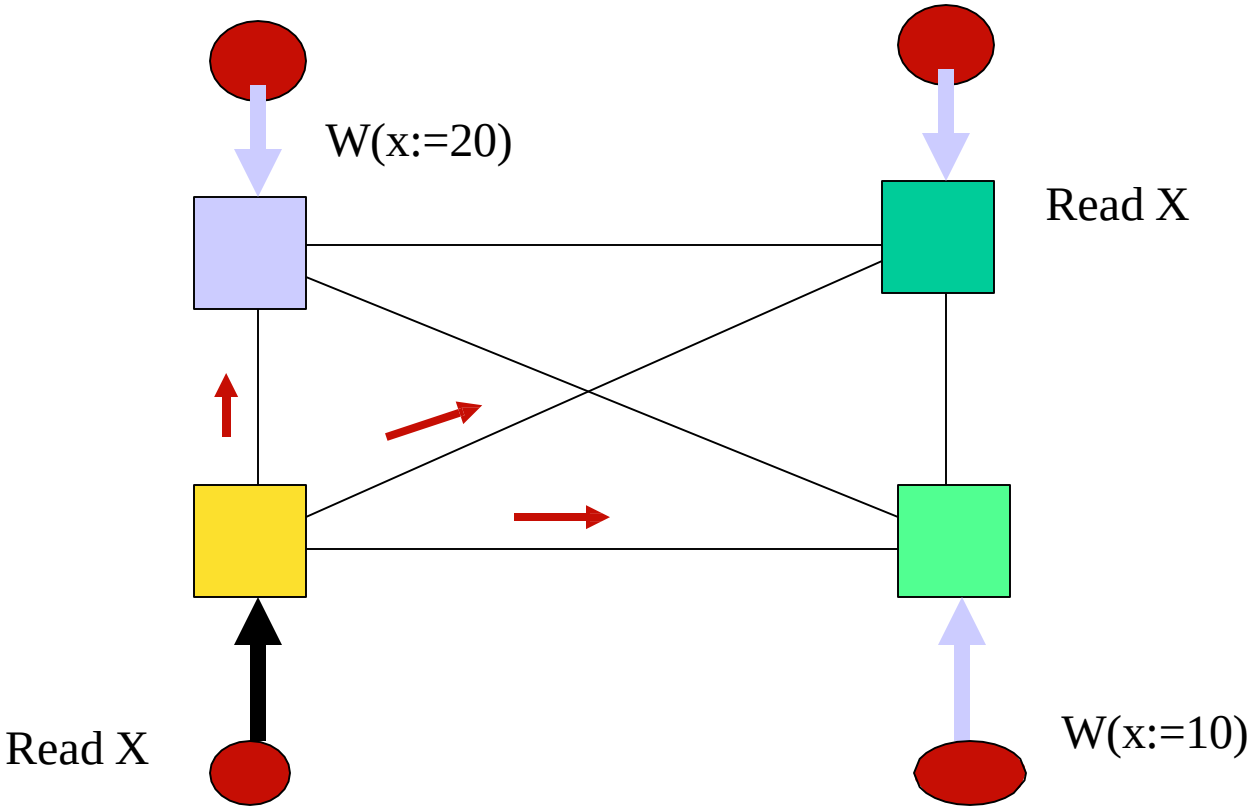
Comparison Table

Model	Order Guarantee	Real-Time Order	Performance
Linearizability	Total order + real-time	Yes	Slow
Sequential Consistency	Total order	No	Moderate
Causal Consistency	Partial (causal) order	No	Fast

Implementing consistency models

Why are there so many consistency models? The cost (measured by message complexity) of implementation decreases as the models become “weaker”.

Implementing linearizability



Needs total order multicast of all reads and writes

Implementing linearizability

- The total order broadcast forces every process to accept and handle all reads and writes in the **same temporal order**.

- The peers update their copies in response to a write, but only send acknowledgements for reads. After this, the local copy is returned.

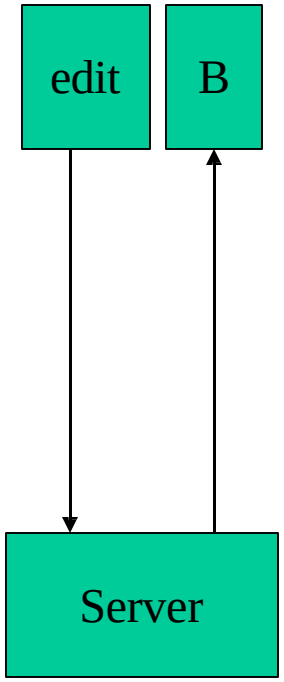
Use total order broadcast all writes only,
but immediately return local copies for reads.

Client centric consistency model

- **Each process must be able to see its own updates**. Consider updating a webpage. If the editor and the browser are not integrated, the editor may send an updated HTML page to the server, but the browser could still return a stale version when the page is viewed.

- To implement this consistency model, the editor must invalidate the cached copy, forcing the browser to fetch the recently uploaded version from the server.

- Each write operation following a read should take effect on the previously read copy, or a more recent version of it.



Protocols for Consistency

(1) Primary-Based Protocol

-Write only once.

-Except for one primary, all are Backup (copy).

-When reading, the client reads data from the closest backup copy.

-If there is a problem with writing, write the values in order and to the primary finally.

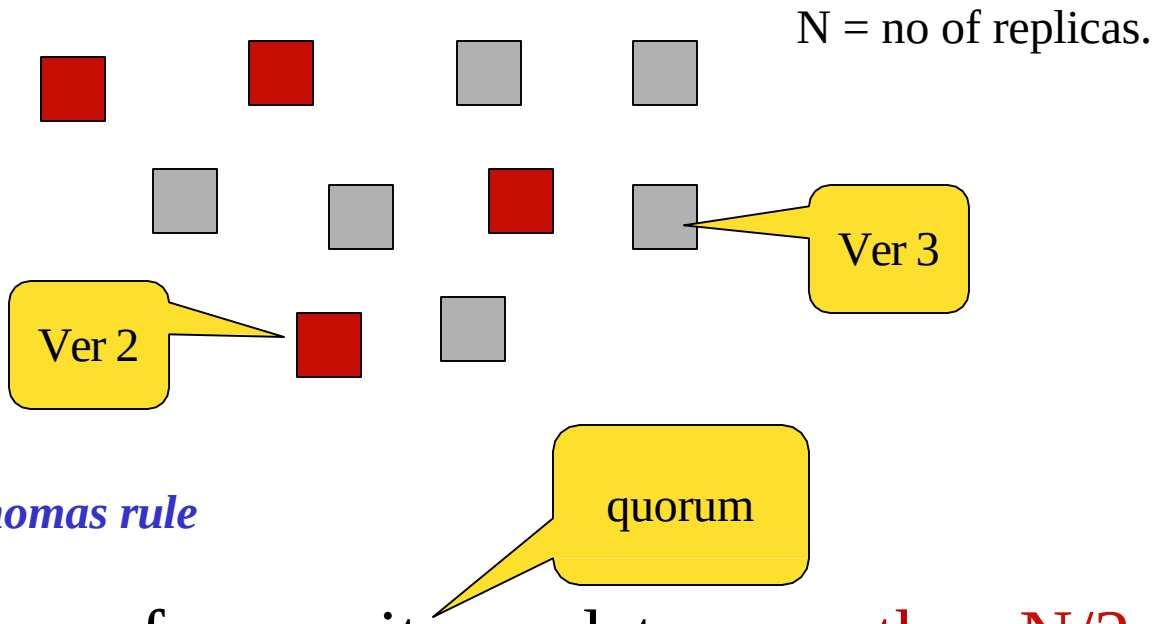
Takes long waiting(block) time to update all the replicas

Protocols for Consistency

(2) Quorum Based Protocol

A **quorum system** engages only a **designated minimum number of the replicas** for every read or write operation – this number is called the **read or write quorum**. When the quorum is not met, the operation (read or write) is postponed.

Quorum-based protocols



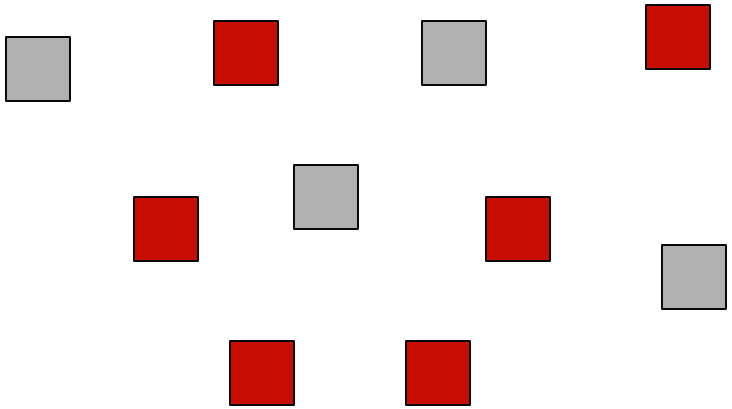
Thomas rule

To perform write, update more than N/2 of them, and tag it with new version number.

To perform read, access more than N/2 replicas with identical values or version numbers.

Otherwise, abandon the read

How it works

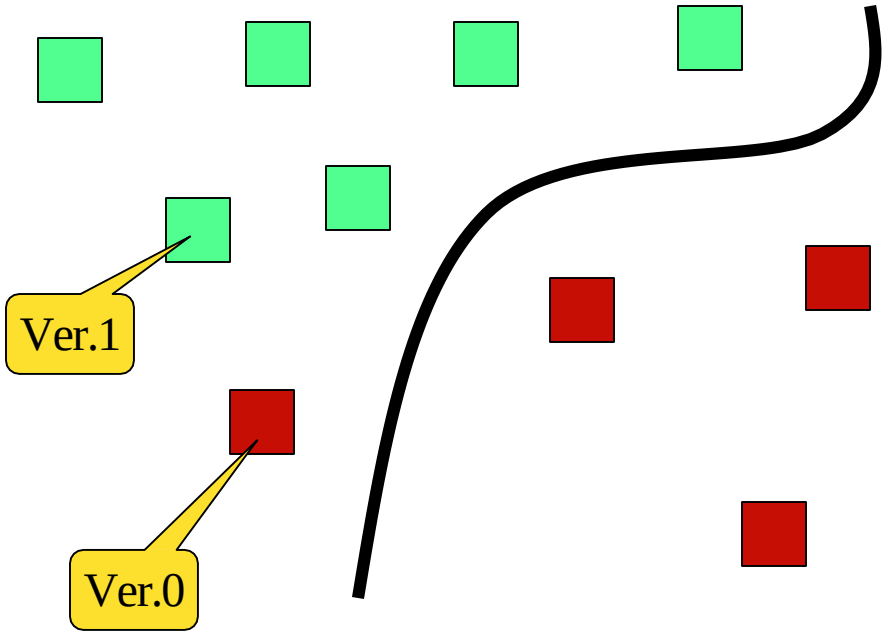


N = no of replicas.

1. Send a write request containing the state and new version number to all the replicas and waits to receive acknowledgements from a write quorum. At that point the write operation is complete and the proxy can return to the user code.

2. Send a read request for the version number to all the replicas, and wait for replies from a read quorum. Then it takes the biggest version number.

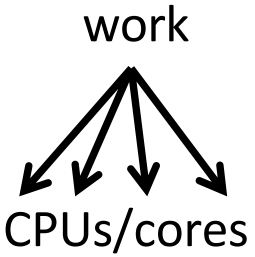
Quorum-based protocols



After a partition, only the larger segment runs the consensus protocol. The smaller segment contains stale data, until the network is repaired.

Parallel vs. concurrent Computing

- **parallel:** Using multiple processing resources (CPUs, cores) at once to solve a problem faster.
  - Example: A sorting algorithm that has several threads each sort part of the array.



- **concurrent:** Multiple execution flows (e.g. threads) accessing a shared resource at the same time.
  - Example: Many threads trying to make changes to the same data structure (a global list, map, etc.).