

**Bule Hora University**  
**Faculty of Engineering and Technology**  
**Department of Information technology**

Year: III

Semester: II

Subject: Introduction to Distributed System  
Code: ITec3102

Course

**Lecture Notes Unit V - Client Centric Consistency Models**

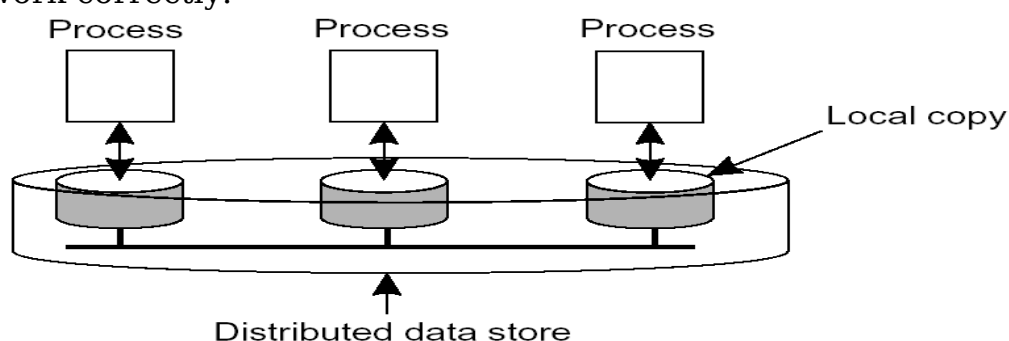
- 1.1 Introduction to Consistency Model**
- 1.2 Eventual Consistency**
- 1.3 Monotonic Reads**
- 1.4 Monotonic Writes**
- 1.5 Read Your Writes**
- 1.6 Writes follow Reads**

**1.1. Introduction to Consistency Model**

- ✓ An important issue in distributed systems is the replication of data. Data are generally replicated to enhance **reliability** or improve **performance**. One of the major problems is keeping replicas consistent. Informally, this means that when one copy is updated we need to ensure that the other copies are updated as well; otherwise the replicas will no longer be the same.
- ✓ If objects (or data) are shared, we need to do something about concurrent accesses to guarantee state consistency.
- ✓ To keep replicas consistent, we generally need to ensure that all **conflicting** operations are done in the same order everywhere
- ✓ Conflicting operations: From the world of transactions:
  - Read-write conflict: a read operation and a write operation act concurrently
  - Write-write conflicts: two concurrent write operations

**Consistency Model**

- ✓ A consistency model is essentially a contract between processes and the data store (distributed collection of storages accessible to clients). It says that if processes agree to obey certain rules, the store promises to work correctly.

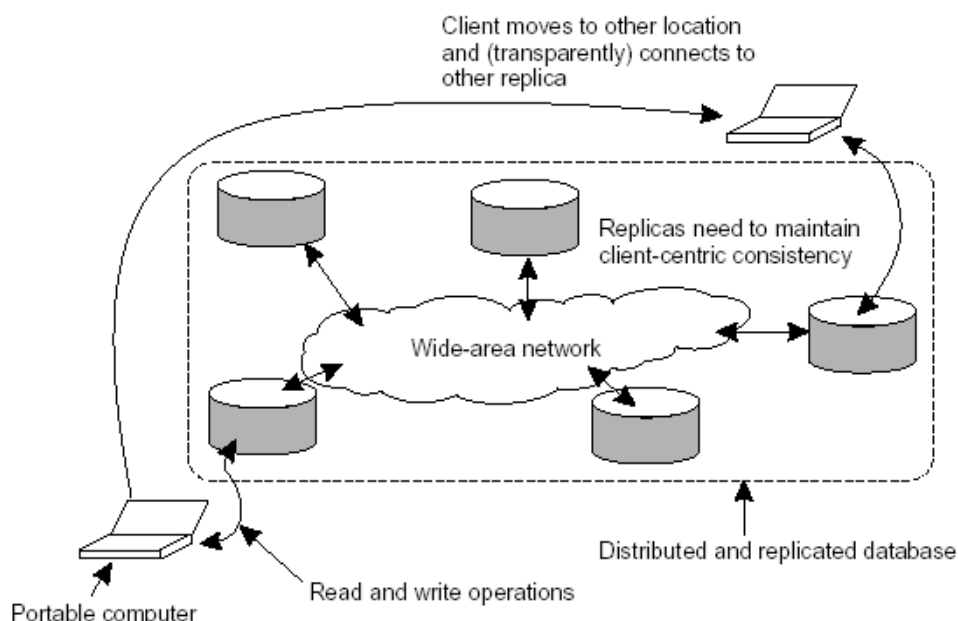


## Client Centric Consistency Model

- Client-centric consistency models are generally used for applications that lack simultaneous updates – i.e., most operations involve reading data. Only few processes involve in updates.
- It concentrates on what specific *clients* want, instead of what should be maintained by servers.
- The following are the client centric consistency models
  - Eventual Consistency
  - Monotonic Reads
  - Monotonic Writes
  - Read Your Writes
  - Writes follow Reads

### 1.2. Eventual Consistency

- ✓ In many cases concurrency appears only in restricted form. In many applications most processes only read data. In some cases if for a long time no update takes place all replicas gradually and eventually become consistent. It is called eventual consistency.
- ✓ Clients access distributed data store using, generally, the local copy. Updates are eventually propagated to other copies.
- ✓ Eventual consistency essentially requires only that updates are guaranteed to propagate to all replicas. However, problems arise when different replicas are accessed over a short period of time. For example, for the mobile user example, eventual consistent data stores will not work properly



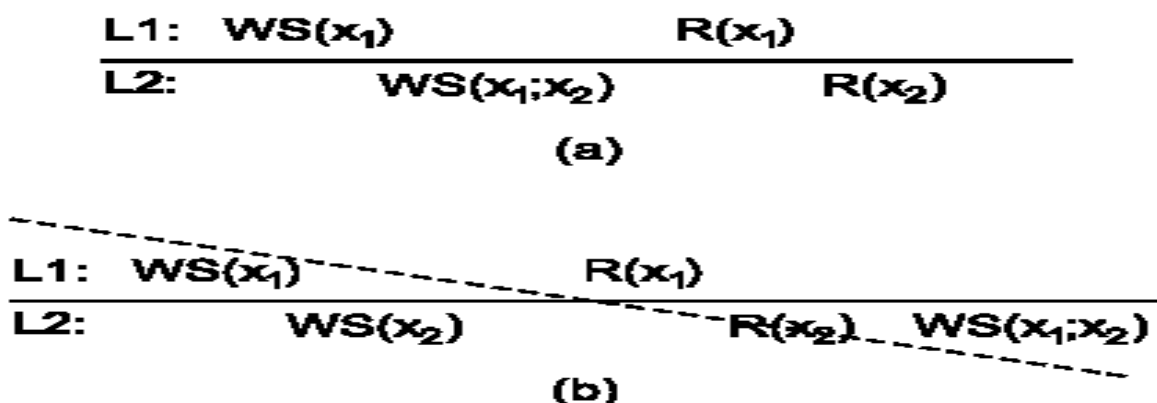
A mobile user accessing different replicas of a distributed database has problems with eventual consistency.

### 1.3. Monotonic Reads

✓ If a process reads the value of a data item  $x$ , any successive read operation on  $x$  by that process will always return that same value or a more recent value. In other words, monotonic-read consistency guarantees that if a process has seen a value of  $x$  at time  $t$ , it will never see an older version of  $x$  at a later time.

✓ As an example where monotonic reads are useful, consider a distributed email database. In such a database, each user's mailbox may be distributed and replicated across multiple machines. Mail can be inserted in a mailbox at any location. However, updates are propagated in a lazy (i.e., on demand) fashion. Only when a copy needs certain data for consistency are those data propagated to that copy. Suppose a user reads his mail in San Francisco. Assume that only reading mail does not affect the mailbox, that is, messages are not removed, stored in subdirectories, or even tagged as having already been read, and so on. When the user later flies to New York and opens his mailbox again, **monotonic-read consistency** guarantees that the messages that were in the mailbox in San Francisco will also be in the mailbox when it is opened in New York.

**Example:** The read operations are performed by a single process  $P$  at two different local copies ( $L1$  &  $L2$ ) of the same data store



**Notation:**  $WS(x_i[t])$  is the set of write operations (at  $L_i$ ) that lead to version  $x_i$  of  $x$  (at time  $t$ ):  $WS(x_i[t_1]; x_j[t_2])$  indicates that it is known that  $WS(x_i[t_1])$  is part of  $WS(x_j[t_2])$

**Note:** Parameter  $t$  is omitted from figures

To guarantee monotonic-read consistency, all operations in  $WS(x_1)$  should have been propagated to  $L2$  before the second read operation takes place. In other words, we need to know for sure that  $WS(x_1)$  is part of  $WS(x_2)$  which is expressed as  $WS(x_1; x_2)$ .

- (a) A monotonic-read consistent data store.
- (b) A data store that does not provide monotonic reads

#### 1.4. Monotonic Writes

- ✓ A data store is said to be **monotonic-write consistent** if the following condition holds:  
*A write operation by a process on a data item  $x$  is completed before any successive write operation on  $x$  by the same process.*
- ✓ That is, a write operation on a copy of data item  $x$  is performed only if that copy has been brought up to date by means of any preceding write operations, which may have taken place on other copies of  $x$ .

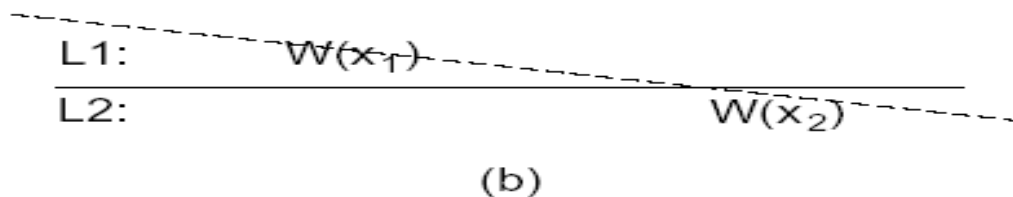
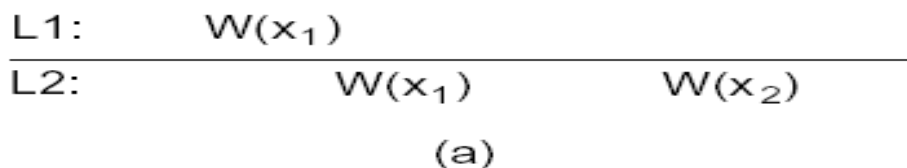


**Figure (a) - a data store that is monotonic-write consistent**

- P performs a write operation on  $x$  at L1,  $W(x_1)$ . Later, P performs a write operation on  $x$  at L2,  $W(x_2)$
- $W(x_2)$  requires that  $W(x_1)$  is updated on L2 before it.

**Figure (b) - a data store that is not monotonic-write consistent**

- $W(x_1)$  has not been propagated to L2

#### 1.5. Read Your Writes

- ✓ A data store is said to be read-your-writes consistent if the following condition holds:

The effect of a write operation by a process on data item  $x$ , will always be seen by a successive read operation on  $x$  by the same process.

- ✓ That is, a write operation is always completed before a successive read operation by the same process, no matter where that read operation takes place.

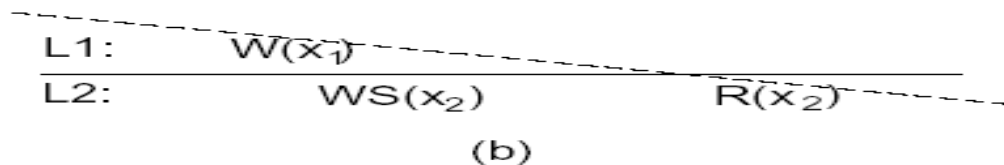
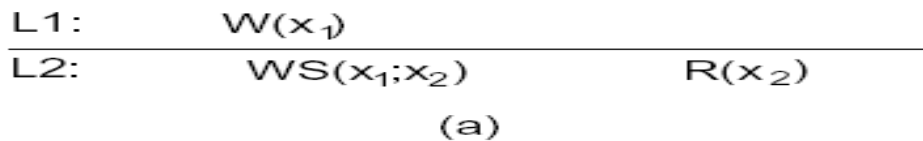


**Figure (a) - a data store that is read-your-writes consistent**

- $P$  performs a write operation on  $x$  at  $L1$ ,  $W(x_1)$ . Later,  $P$  performs a read operation on  $x$  at  $L2$ ,  $R(x_2)$ .
- $WS(x_1;x_2)$  states that  $W(x_1)$  is part of  $WS(x_2)$ .

**Figure (b) - a data store that is not read-your-writes consistent**

- $W(x_1)$  is left out of  $WS(x_2)$ . That is, the effects of the previous write operation by process  $P$  have not been propagated to  $L2$ .

**Example:** Updating your Web page and guaranteeing that your Web browser shows the newest version instead of its cached copy.

## 1.6. Writes follow Reads

- ✓ A data store is said to be writes-follow-reads consistent if the following condition holds:  
A write operation by a process on a data item  $x$ , following a previous read operation on  $x$  by the same process, is guaranteed to take place on the same or a more recent value of  $x$  that was read.
- ✓ That is, any successive write operation by a process on a data item  $x$  will be performed on a copy of  $x$  that is up to date with the value most recently read by that process.

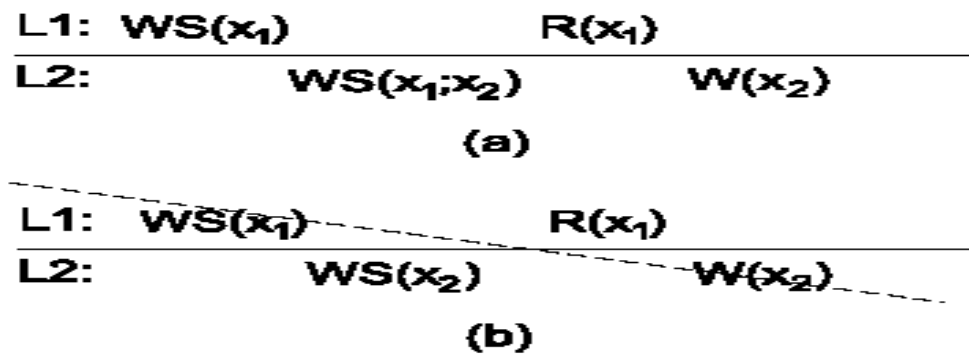


**Figure (a) - a data store that is writes-follow-reads consistent**

- P performs a read operation on  $x$  at L1,  $R(x_1)$ .
- The write operations that led to  $R(x_1)$ , also appear in the write set at L2, where P later performs  $W(x_2)$ .

**Figure (b) - a data store that is not writes-follow-reads consistent**

- The write operations that led to  $R(x_1)$ , did not appear in the write set at L2, before P later performs  $W(x_2)$ .

**Example:** Writes-follow-reads consistency can be used to guarantee that users of a network newsgroup see a posting of a reaction to an article only after they have seen the original article.