

# Réplication et cohérence de données (Data replication and consistency)

**Claudia-Lavinia Ignat**

**Directrice de Recherche Inria**

**[claudia.ignat@inria.fr](mailto:claudia.ignat@inria.fr)**

**B-036 LORIA**

# Organisation

- Lecture (16h)
  - Last 4h dedicated to presentations of assigned papers (groups of 2 students)
- Exercises (8h)
  - A project (groups of 2 students)
- Web site: <https://members.loria.fr/CIgnat/>

# Evaluation

- Paper presentation
  - Groups of 2 students
  - 20 min presentation + 10 min questions
- Project
  - Groups of 2 students
- Exam
  - No documents

# Course overview

- Introduction to replication
- Consistency models (\*)
- Consistency protocols (\*)
- Pessimistic replication vs. optimistic replication
- Optimistic replication approaches

(\*) Andrew S. Tanenbaum, Maarten Van Steen, "Distributed Systems: Principles and Paradigms", 2002

# Replication is everywhere

- Facebook
  - 3 billion active users
- X (Twitter)
  - 586 million monthly active users
  - 500 million tweets / day
  - 2.1 billion search queries /day
- Google
  - >2.5 million servers (in July 2016)
  - >8.5 billion search requests/day
  - Synchronous data replication (Google Drive, Gmail, Google Sites, Calendar)
- Amazon
  - >310 million active customer accounts
  - ~15,000\$ revenue/second



# Introduction

- Distributed system
  - Sites (processing unit + storage device)
  - Communication links (bidirectional communication between 2 sites)
  - Messages (no guarantee to be delivered within a maximum delay)
- Failures of sites and links
  - An error leads to a faulty state
  - Errors: human mistakes or physical damage
  - Site failures: stopping (crash of a critical subsystem), malicious actions (byzantine failures)
  - Link failures: messages are no longer transmitted or excessively delayed, unidirectional transmission, byzantine communication failure
  - One consequence: network partition

# Reliability (Fiabilité)

- Property of tolerating component failures for the longest time
- A system is perfectly reliable if it never fails
- A system is reliable if it fails rarely and almost always recovers from component failures and design faults s.t. its activity is resumed without perceptible interruption
- A system is reliable to the extent that it is able to successfully complete a service request once it accepts it

# Availability (Disponibilité)

- Accessibility of system services to users
- A system is highly available if the fraction of its down-time is very small (failures are rare or it can restart very quickly after a failure)
- A system is highly available if denial of service request is rare



# Reliability vs. Availability

- Reliability: duration of time a system is expected to remain in continuous operation
- Availability: fraction of time instants where the system is operational
- A reliable system is not necessarily highly available
- A highly available system is not necessarily reliable
- Dependability = Reliability x Availability

# Reasons for replication

- Reliability and availability
  - System continues to work if one replica crashes
  - Better protection against corrupted data
- Performance
  - Scaling in numbers - no overloading of a server (replicated web servers)
  - Scaling with the size of a geographical area – reduced communication latency (web cache)
- Challenge: how to maintain consistency between replicated data ?

# CAP Theorem<sup>1</sup>

- Properties of shared-data systems
  - Data consistency
  - System availability
  - Tolerance to network partition
- Only two out of three properties can be achieved at a given time

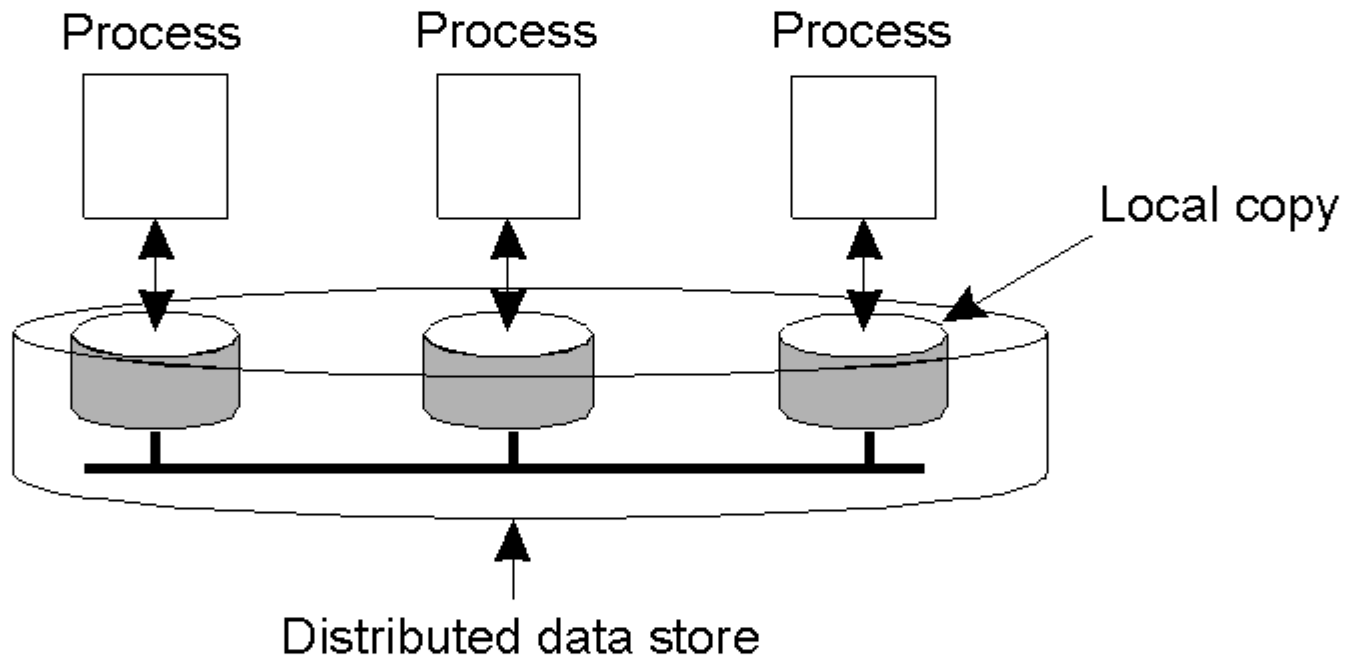
1. Brewer, E.A. Towards robust distributed systems. In Proc. of PODC, 2000.

# CAP Theorem

- A system without network partition, can achieve consistency + availability
  - Client and storage system are part of the same environment
- In a large-scale distributed systems, network partitions exist
  - Consistency and availability cannot be both achieved
    - Relax consistency, maintain availability
    - Maintain consistency, tolerate unavailability under certain conditions

# Data-Centric Consistency Models

- The general organization of a logical data store, physically distributed and replicated across multiple processes.



# Data-Centric Consistency Models

- Write operation when it changes data, otherwise read operation
- Consistency model=contract between processes and data store
  - If processes agree to obey certain rules, data store promises to work correctly
- A process that performs a read expects a return value that shows results of last write
- Too strict criteria as lack of global clock, need for other consistency models
- Each model restricts the values that a read can return

# Strict Consistency

- *Any read on a data item  $x$  returns a value corresponding to the result of the most recent write on  $x$*
- Assumes existence of absolute global time
- Two main issues:
  - Definition of “most recent event”
  - Instantaneous execution of operations
- Example:
  - $X$  a data item stored at machine B
  - Machine A reads  $X$  at time  $T_1$  and message sent to B to read  $X$
  - At  $T_2$  a process on B writes  $X$
  - Read should return old value of  $X$  regardless of where machines are and how close  $T_2$  and  $T_1$  is

# Strict Consistency

- “most recent event” needs perfectly synchronised clocks (0 delay between any 2 sites)
- An operation that requires a remote access cannot be executed instantaneously (a local operation launched after a remote operation can be terminated before)
- Notation:
  - $W_i(x)a$  – process  $P_i$  writes  $x$  with value  $a$
  - $R_i(x)b$  – process  $P_i$  reads  $x$  with value  $b$



# Strict Consistency

Behavior of two processes, operating on the same data item.

P1:  $W(x)a$   
-----  
P2:  $R(x)a$   
(a)

P1:  $W(x)a$   
-----  
P2:  $R(x)NIL$   $R(x)a$   
(b)

A strictly consistent store

A store that is not strictly consistent.

- Strict consistency is an ideal model
- Need for relaxed consistency models

# Sequential Consistency (1)

- Sequential consistency (defined by Lamport)
  - *The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program*
  - Any valid interleaving of operations is possible, but all processes see the same interleaving
  - There is no reference to the “most recent” write
  - A process sees writes from all processes but only its own reads

# Sequential Consistency (2)

P1:	W(x)a		
<hr/>			
P2:	W(x)b		
<hr/>			
P3:		R(x)b	R(x)a
<hr/>			
P4:		R(x)b	R(x)a

(a)

P1:	W(x)a		
<hr/>			
P2:	W(x)b		
<hr/>			
P3:		R(x)b	R(x)a
<hr/>			
P4:		R(x)a	R(x)b

(b)

- a) A sequentially consistent data store.
- b) A data store that is not sequentially consistent.

# Linearizability (1)

- Linearizability weaker than strict consistency but stronger than sequential consistency
  - $ts_{OP}(x)$  timestamp assigned to operation OP performed on data item x
  - *The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program. In addition if  $ts_{OP_1}(x) < ts_{OP_2}(y)$ , then operation  $OP_1(x)$  should precede  $OP_2(y)$  in this sequence*
  - A linearizable data store is also sequentially consistent

# Example Sequential Consistency (1)

Process P1

Process P2

Process P3

---

**x = 1;**  
**print ( y, z);**

**y = 1;**  
**print (x, z);**

**z = 1;**  
**print (x, y);**

# Example Sequential Consistency (2)

- Four valid execution sequences

**x = 1;**

**print (y, z);**

**y = 1;**

**print (x, z);**

**z = 1;**

**print (x, y);**

**Prints: 001011**

**Signature:**

**001011**

**x = 1;**

**y = 1;**

**print (x,z);**

**print(y, z);**

**z = 1;**

**print (x, y);**

**Prints: 101011**

**Signature:**

**101011**

**y = 1;**

**z = 1;**

**print (x, y);**

**print (x, z);**

**x = 1;**

**print (y, z);**

**Prints: 010111**

**Signature:**

**110101**

**y = 1;**

**x = 1;**

**z = 1;**

**print (x, z);**

**print (y, z);**

**print (x, y);**

**Prints: 111111**

**Signature:**

**111111**

- 000000 not permitted
- 001001 not allowed

# Formal Expression Sequential Consistency (1)

- Each process  $P_i$  has an associated execution  $E_i$  of read and write performed on data store  $S$

P1:	W(x)a		
<hr/>			
P2:	W(x)b		
<hr/>			
P3:		R(x)b	R(x)a
<hr/>			
P4:		R(x)b	R(x)a

- $E_1: W_1(x)a$
- $E_2: W_2(x)b$
- $E_3: R_3(x)b, R_3(x)a$
- $E_4: R_4(x)b, R_4(x)a$
- Merge  $E_i$  into  $H$  s.t. each operation in  $E_i$  appears in  $H$  once

# Formal Expression Sequential Consistency (2)

- Legal values for H must obey the constraints
  - Program order must be maintained: if  $OP_1$  before  $OP_2$  in  $E_i$ , then  $OP_1$  before  $OP_2$  in H
  - Data coherence must be respected:  $R(x)$  must return the value most recently written to x
- $H=W_2(x)b, R_3(x)b, R_4(x)b, W_1(x)a, R_3(x)a, R_4(x)a$



# Linearizability and Sequential Consistency

contrainte de temps :  $W(x)a$  précède  $W(x)b$

$p_1$	$W(x)a$		
$p_2$		$W(x)b$	
$p_3$		$R(x)a$	$R(x)b$
$p_4$		$R(x)a$	$R(x)b$

Notation :

$R_i(x)a$  : par  $p_i$ , lecture de  $x$ , résultat  $a$

$W_i(x)a$  : écriture de  $x$ , valeur  $a$

$S = W(x)a R_3(x)a R_4(x)a W(x)b R_3(x)b R_4(x)b$

linéarisable

$p_1$	$W(x)a$		
$p_2$	$W(x)b$		
$p_3$		$R(x)a$	$R(x)b$
$p_4$		$R(x)a$	$R(x)b$

$p_1$	$W(x)a$		
$p_2$	$W(x)b$		
$p_3$		$R(x)b$	$R(x)a$
$p_4$		$R(x)a$	$R(x)b$

$S = W(x)a R_3(x)a R_4(x)a W(x)b R_4(x)b R_3(x)b$

séquentiel, non linéarisable

non séquentiel

# Sequential Consistency

- It is costly to be realised
  - $t$  the minimal transfer time of a message between two sites
  - $r$  the reading period
  - $w$  the writing period
  - $r+w \geq t$
  - Gain on reading results in a lose of writing time

# Causal Consistency (1)

- If event B is caused or influenced by an earlier event A, causality requires that everyone sees A and then B
- Concurrent operations = operations not causally related
- Examples:
  - If  $E_1$ : p writes x, then  $E_2$ : q reads x, then  $E_1 \rightarrow E_2$
  - If  $E_1$ : p reads x, then  $E_2$ : p writes y, then  $E_1 \rightarrow E_2$  (value of y depends on x)
  - If  $E_1$ : p writes x, then  $E_2$ : q writes y (independently), then  $E_1 \parallel E_2$

# Causal Consistency (2)

- *Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.*

# Causal Consistency (3)

P1:	W(x)a		W(x)c		
P2:		R(x)a	W(x)b		
P3:		R(x)a		R(x)c	R(x)b
P4:		R(x)a		R(x)b	R(x)c

- Allowed with a causally-consistent store
- Not allowed with sequentially or strictly consistent store.

# Causal Consistency (4)

P1:	W(x)a			
P2:		R(x)a	W(x)b	
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

(a)

$W_2(x)b$  depends on  $W_1(x)a$

P1:	W(x)a			
P2:			W(x)b	
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

(b)

- a) A violation of a causally-consistent store.
- b) A correct sequence of events in a causally-consistent store.

# Causal Consistency (5)

- Implementation needs keeping track of which processes have seen which writes
- Construction of a dependency graph
- One solution based on vector timestamps

# FIFO Consistency (1)

- *Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes.*



# FIFO Consistency (2)

P1: W(x)a

---

P2:            R(x)a    W(x)b    W(x)c

---

P3:    R(x)b    R(x)a    R(x)c

---

P4:    R(x)a    R(x)b    R(x)c

- valid FIFO consistency

# FIFO Consistency (3)

```
x = 1;  
print (y, z);  
y = 1;  
print(x, z);  
z = 1;  
print (x, y);
```

Prints: 00

(a)

```
x = 1;  
y = 1;  
print(x, z);  
print (y, z);  
z = 1;  
print (x, y);
```

Prints: 10

(b)

```
y = 1;  
print (x, z);  
z = 1;  
print (x, y);  
x = 1;  
print (y, z);
```

Prints: 01

(c)

- Result impossible to obtain with sequential consistency

# FIFO Consistency (4)

Process P1

x = 1;

if (y == 0) kill (P2);

Process P2

y = 1;

if (x == 0) kill (P1);

- With FIFO consistency both processes  $P_1$  and  $P_2$  can be killed

# Summary of Consistency Models

Consistency	Description
<b>Strict</b>	<b>Absolute time ordering of all shared accesses matters.</b>
<b>Linearizability</b>	<b>All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp</b>
<b>Sequential</b>	<b>All processes see all shared accesses in the same order. Accesses are not ordered in time</b>
<b>Causal</b>	<b>All processes see causally-related shared accesses in the same order.</b>
<b>FIFO</b>	<b>All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order</b>

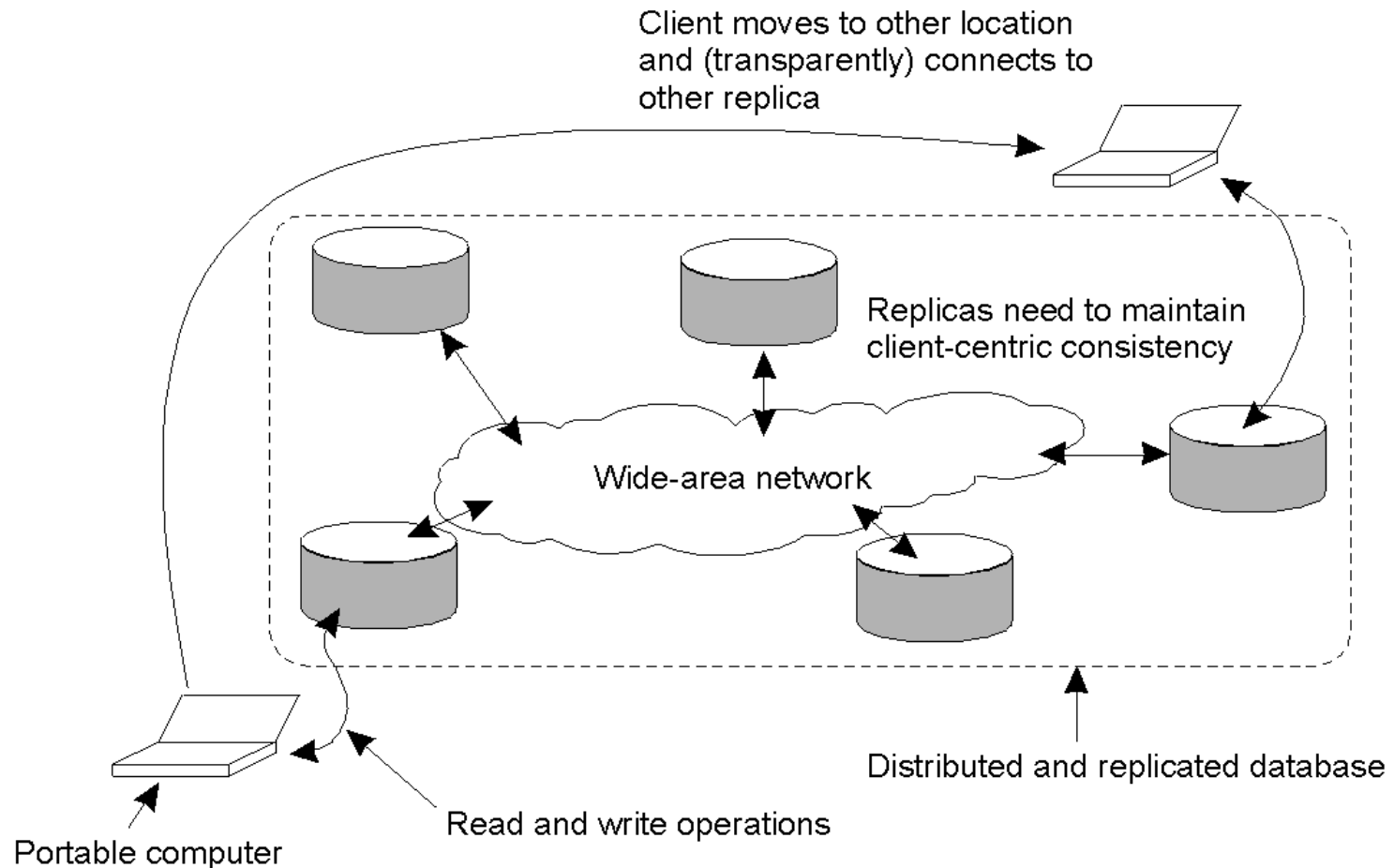
# Data Centric vs. Client Centric Consistency Models

- Data-centric consistency model
  - Provides a systemwide consistent view on a data store
  - Assumption: concurrent processes may simultaneously update the data store
- Client-centric consistency model
  - Assumption: lack of simultaneous updates or when updates happen they can easily be resolved
  - Most operations involve reading data
  - Data stores offer a very weak consistency model called eventual consistency

# Eventual Consistency (1)

- Concurrency appears only in a restricted form
  - Database systems
    - most processes hardly perform updates, but only read data
  - DNS
    - domains assigned to naming authorities that are allowed to update their part of the name space
    - no write-write conflicts, only read-write conflicts
    - Acceptable to propagate an update in a lazy fashion
  - Web pages
    - Usually updated by a single authority
    - No write-write conflicts
    - Design of local caches for efficiency
- If no updates take place for a long time, all replicas become consistent

# Eventual Consistency (2)



# Client centric consistency

- Provides guarantees for a single client concerning consistency of accesses to a data store by that client
- A client connects to different replicas during a period of time and the differences should be made transparent
- Whenever a client connects to a new replica, that replica is brought up to date with the data that was manipulated by that client before and can reside at other replica sites
- Notations
  - $x_i[t]$  version of  $x$  at local copy  $L_i$  at time  $t$
  - $WS(x_i[t])$  series of write operations at  $L_i$  that took place since initialization
  - $WS(x_i[t_1];x_j[t_2])$  if operations in  $WS(x_i[t_1])$  have been performed at the local copy  $L_j$  at time  $t_2$



# Monotonic Reads (1)

- *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*
- Example:
  - A distributed email database
  - Each user's mailbox may be distributed and replicated across multiple machines
  - Mail can be inserted at any location
  - Updates propagated in a lazy manner
  - Emails read (no remove, etc.) at location X are present when they are read later at location Y

# Monotonic Reads (2)

L1:	WS(x <sub>1</sub> )	R(x <sub>1</sub> )
<hr/>		
L2:	WS(x <sub>1</sub> ;x <sub>2</sub> )	R(x <sub>2</sub> )

(a)

L1:	WS(x <sub>1</sub> )	R(x <sub>1</sub> )
<hr/>		
L2:	WS(x <sub>2</sub> )	R(x <sub>2</sub> ) WS(x <sub>1</sub> ;x <sub>2</sub> )

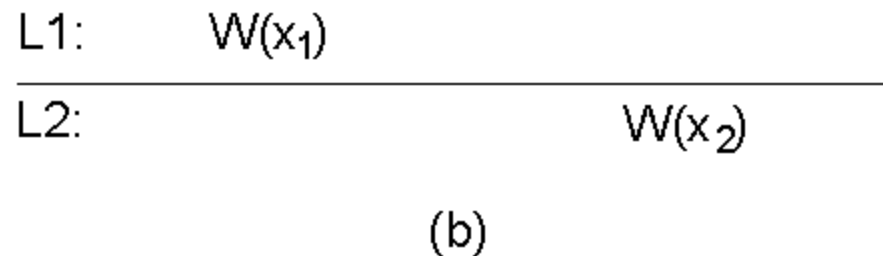
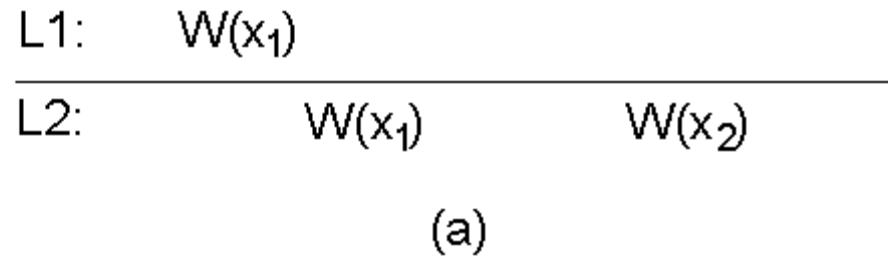
(b)

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

# Monotonic Writes (1)

- *A write operation by a process on a data item  $x$  is completed before any successive write operation on  $x$  by the same process*
- Similarity with FIFO consistency
- Example: updating a software library

# Monotonic Writes (2)

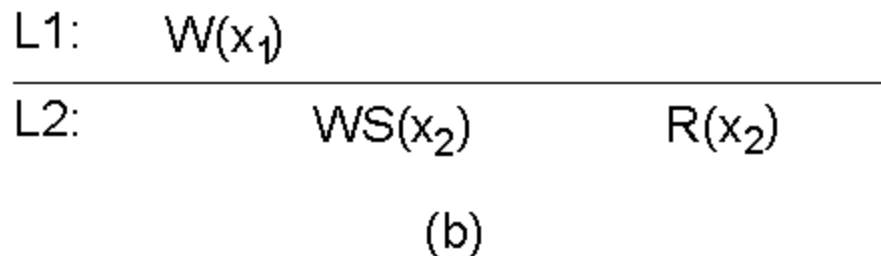
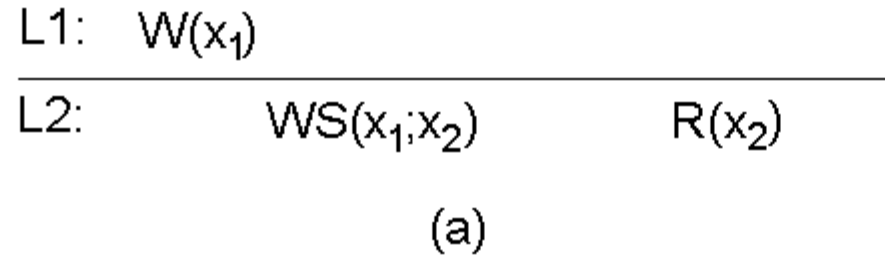


- a) A monotonic-write consistent data store.
- b) A data store that does not provide monotonic-write consistency.

# Read Your Writes (1)

- *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*
- Examples
  - Updating passwords
  - Updating web pages

# Read Your Writes (2)



- a) A data store that provides read-your-writes consistency.
- b) A data store that does not.

# Writes Follow Reads (1)

- *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*
- Example:
  - Network newsgroup that see a posting of a reaction to an article only after they have seen the original article

# Writes Follow Reads (1)

L1:	WS(x <sub>1</sub> )	R(x <sub>1</sub> )
<hr/>		
L2:	WS(x <sub>1</sub> ;x <sub>2</sub> )	W(x <sub>2</sub> )

(a)

L1:	WS(x <sub>1</sub> )	R(x <sub>1</sub> )
<hr/>		
L2:	WS(x <sub>2</sub> )	W(x <sub>2</sub> )

(b)

- a) A writes-follow-reads consistent data store
- b) A data store that does not provide writes-follow-reads consistency



# Examples Systems Consistency

- Microsoft's Windows Azure – strongly consistent storage
  - Clients see the latest written value for a data object
- Amazon Simple Storage Service (S3) - eventual consistency
  - Relaxed consistency for better performance and availability
  - Returned value for reads=object value at some past point in time but not necessarily the latest value
- Amazon's DynamoDB and Google App Engine Datastore – both eventually consistent reads and strongly consistent reads