CSci 5105

Introduction to Distributed Systems

Consistency

Consistency

- Case 1: Replication + Updates

- Data are replicated for availability

- Data are replicated for performance
  - Scaling in numbers
  - Scaling in geographical area

- Performance/Availability gain is not free

Consistency

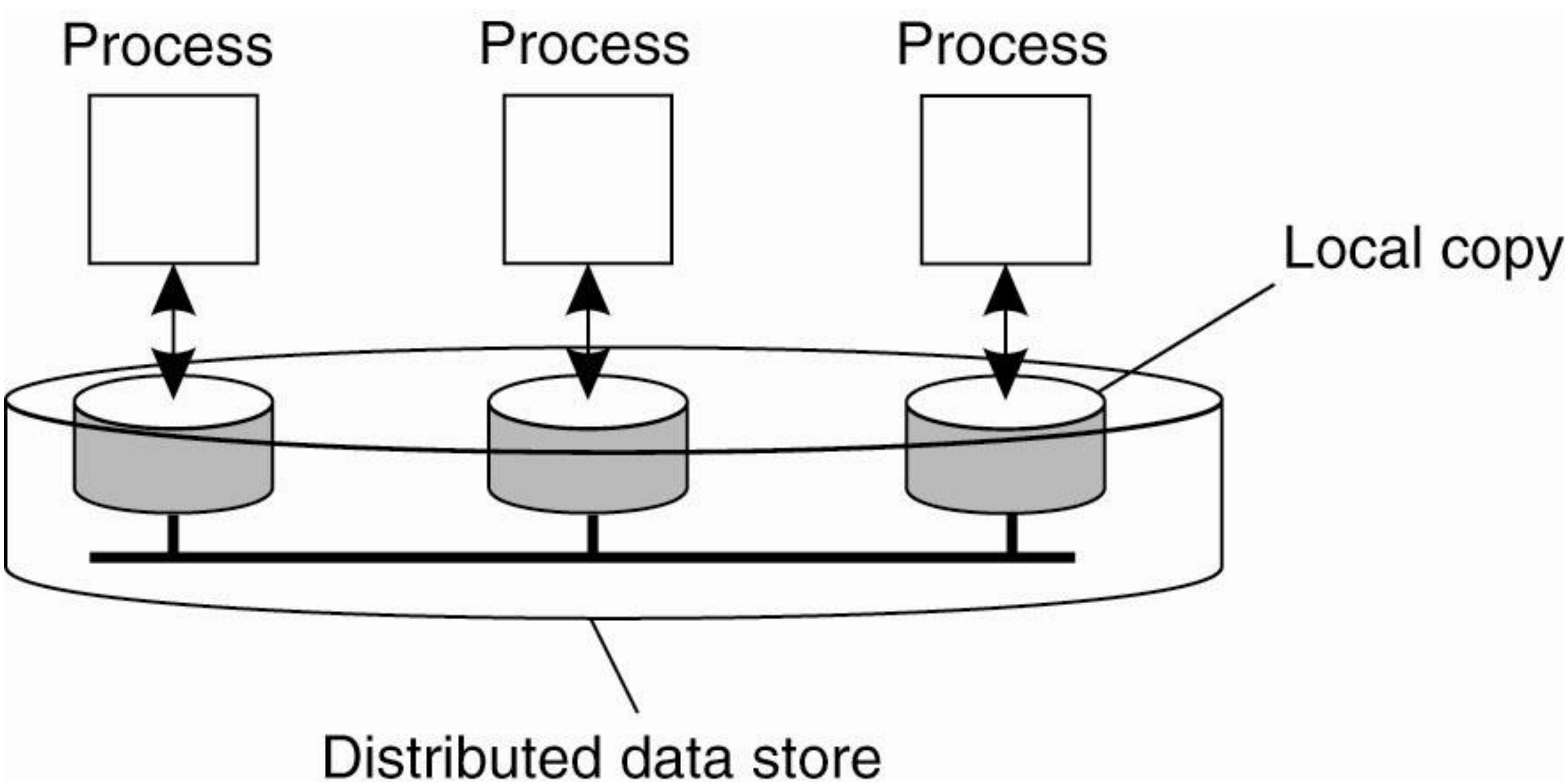
- Replication is not the full story

- Case 2: Single data accessed by many concurrent processes/users

- Two types of consistency
  - data centric, client centric

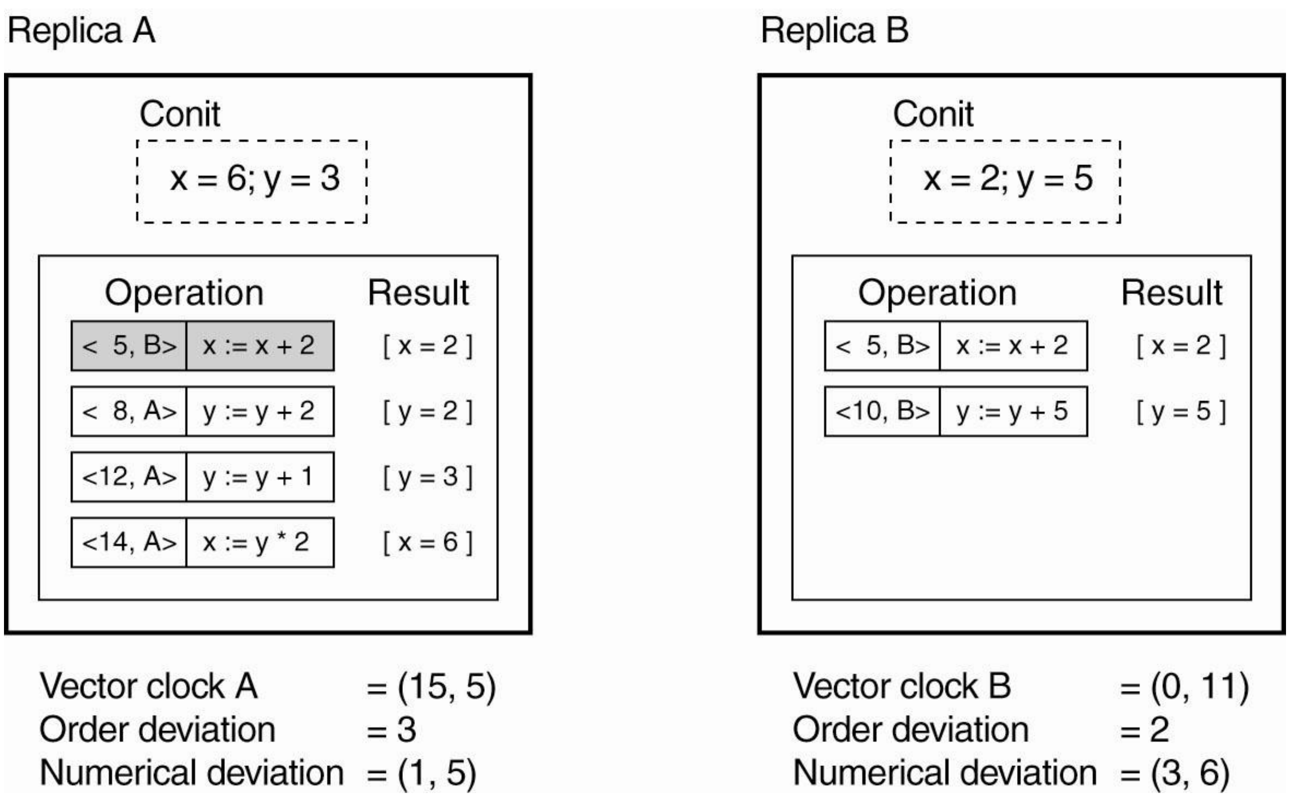
Data-centric Consistency Models

- Data is replicated at K servers



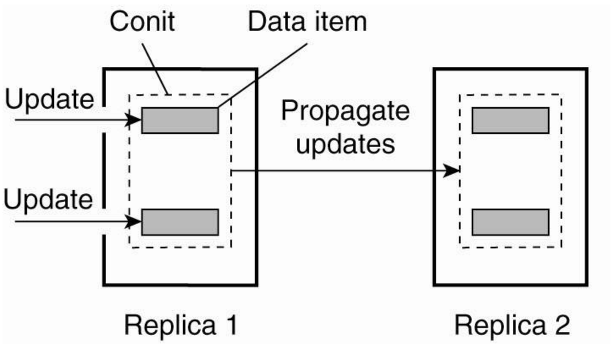
Continuous Consistency

- *Conit*: consistency unit
- value: numerical deviation
- staleness: order deviation



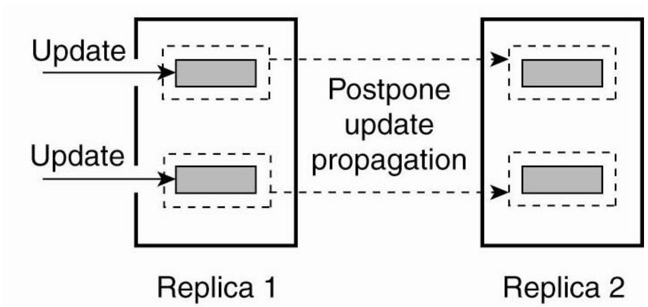
Granularity of Conit

- Policy: deviation at most one update
- Tradeoffs?



(a)

Small: (-) more overhead, (+) more accurate
Large: (-) false sharing, (+) less overhead



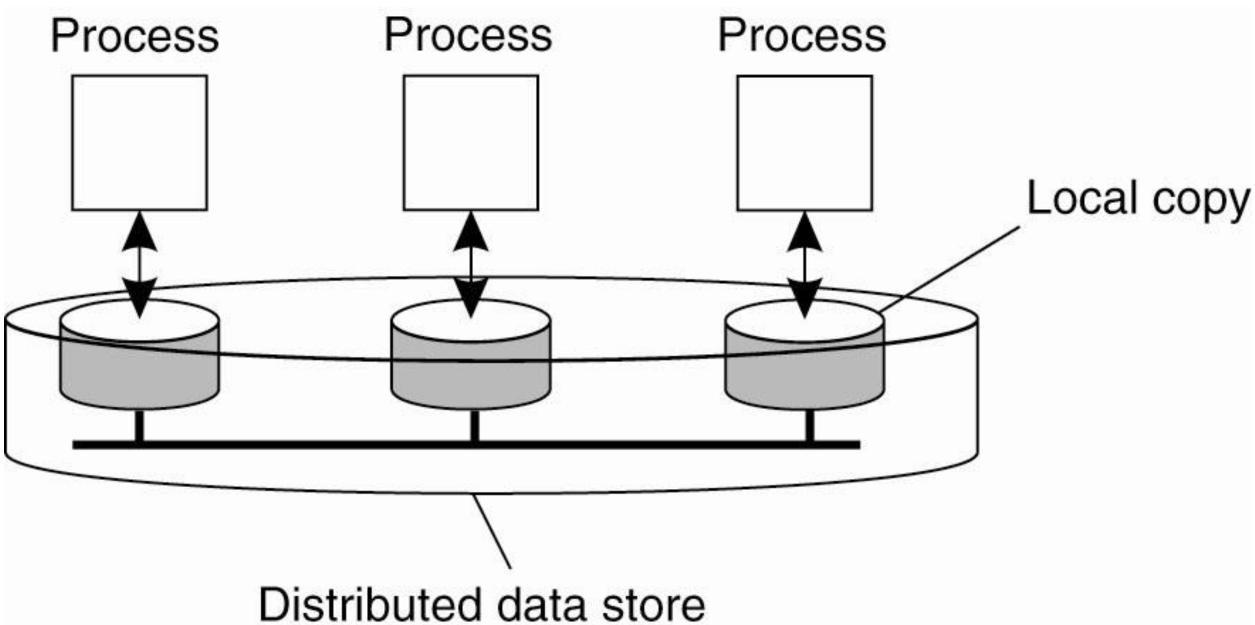
(b)

Qualitative Consistency

- Continuous consistency is quantitative
- Set a bound on order or numerical deviation
  - Nice! can be measured, but how to set it?
- Now, let’s look at qualitative consistency based on ordering

Data-centric Consistency Models

- Data is replicated at K servers
  - Each process assumed to access one of the data-store replicas



Sequential Consistency

- Concurrent access to replicated data
  - Each process sees their operations in the sequential order
  - All processes see (via read) same global interleaving of writes

Sequential Consistency Examples

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

(a)

P3 and P4 see W(x)b then W(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)

Violated – why?

Very expensive

- Why?

Sequential Consistency

- There are many valid sequences and many invalid ones

Process P1	Process P2	Process P3
$x \leftarrow 1$;	$y \leftarrow 1$;	$z \leftarrow 1$;
print(y, z);	print(x, z);	print(x, y);

Sequential Consistency

$x \leftarrow 1$;	$x \leftarrow 1$;	$y \leftarrow 1$;	$y \leftarrow 1$;
print(y, z);	$y \leftarrow 1$;	$z \leftarrow 1$;	$x \leftarrow 1$;
$y \leftarrow 1$;	print(x, z);	print(x, y);	$z \leftarrow 1$;
print(x, z);	print(y, z);	print(x, z);	print(x, z);
$z \leftarrow 1$;	$z \leftarrow 1$;	$x \leftarrow 1$;	print(y, z);
print(x, y);	print(x, y);	print(y, z);	print(x, y);

All valid as they represent legal interleavings
provided all processes see the same interleaving!

E.g. assignment must precede the `print` for each P_i

Stronger?

- Global/Total order
  - Time-stamp every operation
  - There can be only 1 order!

Strongest

- Strict
  - Every action occurs in the order prescribed by a real global clock
  - read sees most recent write based on actual clock

Serializability

- Stronger than sequentially consistent
- From database transactions
  - not just between reads and writes
- Execution appears to be some serial uninterrupted order of *each* P_i
- Example: $P_1P_2P_3$ or $P_2P_3P_1$

Weaker ~ Causal Consistency

- Causal ~ happened before; earlier write may have influenced later write

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

- 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

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

- Causally consistent
  - W(x)a => W(x)c (everyone must see a then c)
  - W(x)b, W(x)c concurrent
- It is also sequentially consistent?
  - No, all writes not seen in same order

Causal Consistency

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)

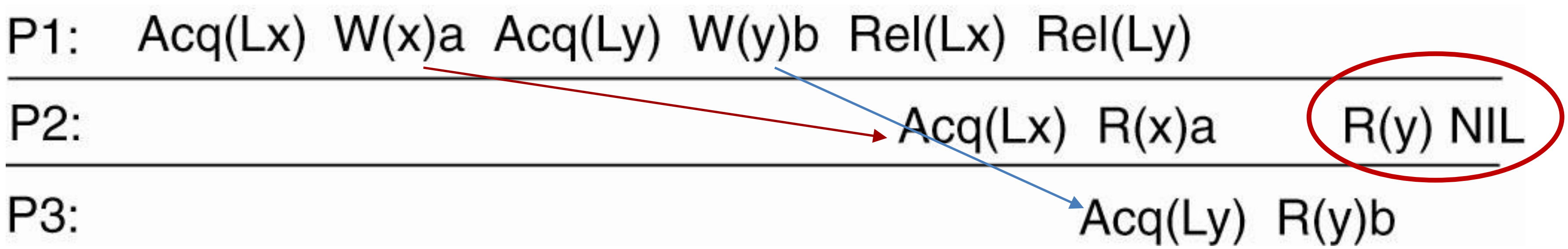
NOT OK

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

(b)

Why OK?

One More: Entry consistency



Enter a CS, data “owner” must transfer most up-to-date copy

Client-Centric

- Data-centric consistency based on ordering of events at the data
- Client-centric models are based on what **a** client (or **a** process) sees
- No simultaneous updates; single client, but client may move to another replica
- Notation: `WS` is the set of write operations seen at a replica

Eventual Consistency

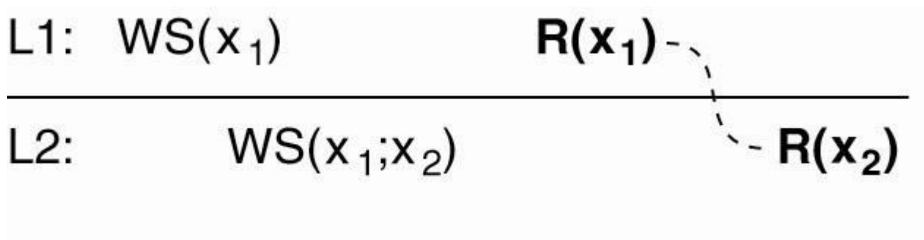
- Context: data replicas
- An update eventually propagates to all replicas; DNS binding update
- When it this ok from client viewpoint?
  - clients access the same replica
- Not ok?

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 AND
- If a process reads … at any replica; that replica must see every prior write that was read by the process

Monotonic Reads

think of X as a bulletin board

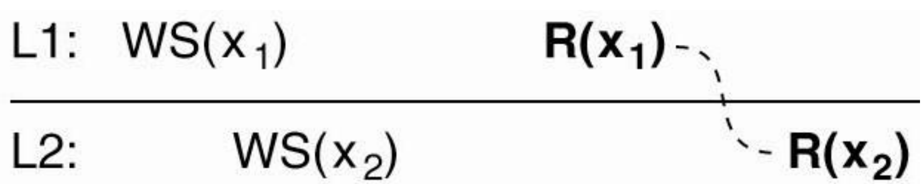


(a)

yes

see at post at L1 also see it later at L2

Time along x axis



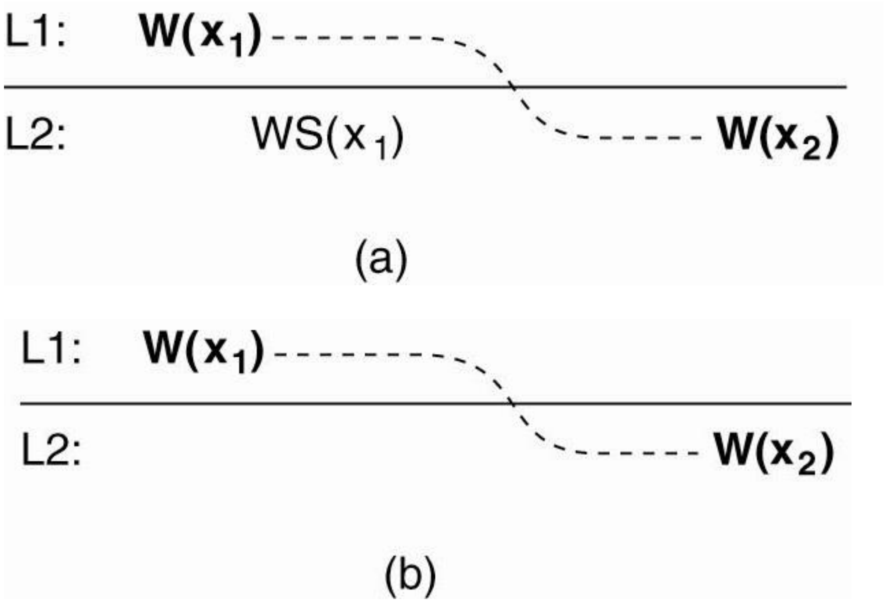
(b)

no

Li are replicas; single client or “process”

Monotonic Writes

- A write operation by a process on a data item x is completed before any successive write operation on x by the same process



yes

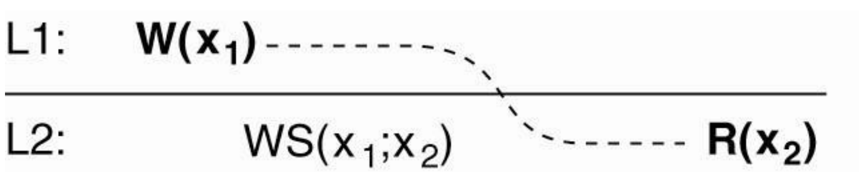
post at L1
post reply at L2

reply must follow
original post

no

Read Your Writes

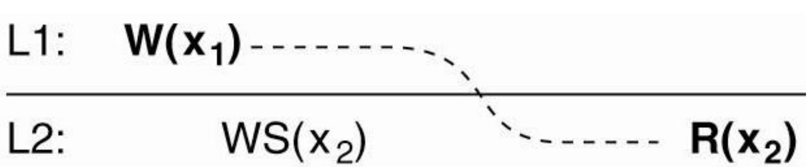
- The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process
  - write completes before read



(a)

yes

post at L1 seen later at L2

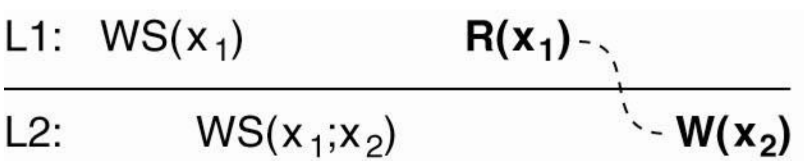


(b)

no

Writes Follow Reads

- 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
  - Read must complete before the write



(a)

yes

read at L1 seen later when posting at L2; follow up

(b)

no

Implementation

- How can we implement them?
  - Blocking
  - Vector time stamps

Next Time

Next topic: Replication

Read Chapter 7 TVS