

Distributed Systems

Consistency & Replication I

Olaf Landsiedel

Last Time

- Logical Clocks
- Vector Clocks
- Consistent Snapshots
- Passive Monitoring
- Active Monitoring

Questions

- Vector Clocks: How many clocks at each node?
 - One per node in the system
- How are they incremented?
 - Local clock: on each event
 - Remote clocks: merged on receive event

Today...

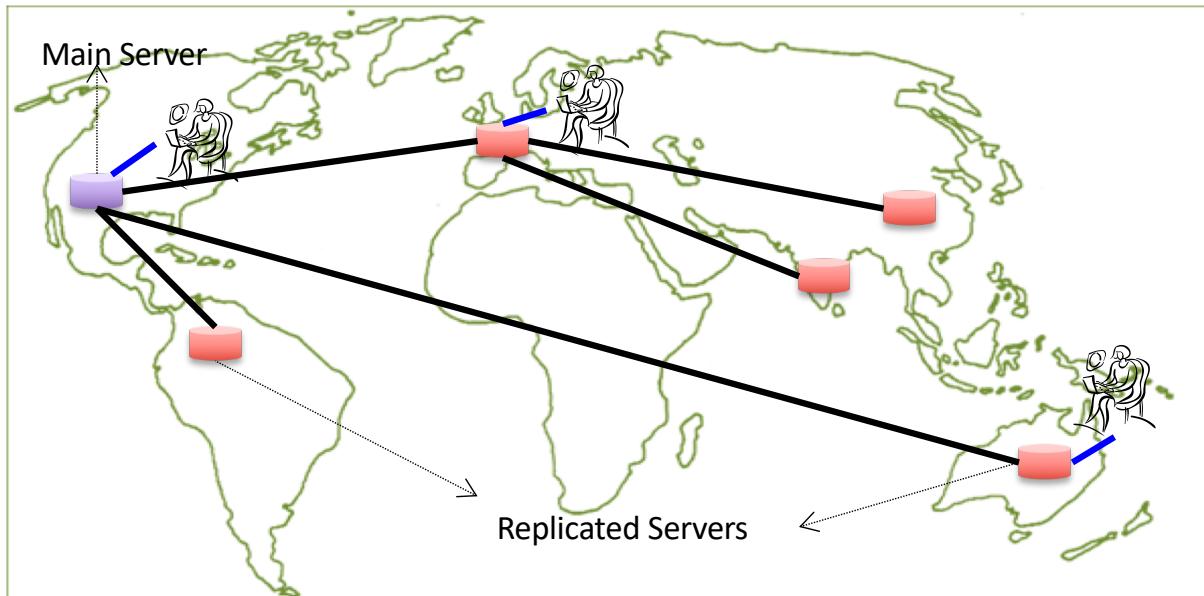
- Consistency and Replication
 - Introduction
 - Data-centric Consistency Models

Why Replication?

- Replication is the process of maintaining the data at multiple computers
- Replication is necessary for:
 - **Improving performance**
 - A client can access the replicated copy of the data that is near to its location
 - **Increasing the availability of services**
 - Replication can mask failures such as server crashes and network disconnection
 - **Enhancing the scalability of the system**
 - Requests to the data can be distributed to many servers which contain replicated copies of the data
 - **Securing against malicious attacks**
 - Even if some replicas are malicious, secure data can be guaranteed to the client by relying on the replicated copies at the non-compromised servers

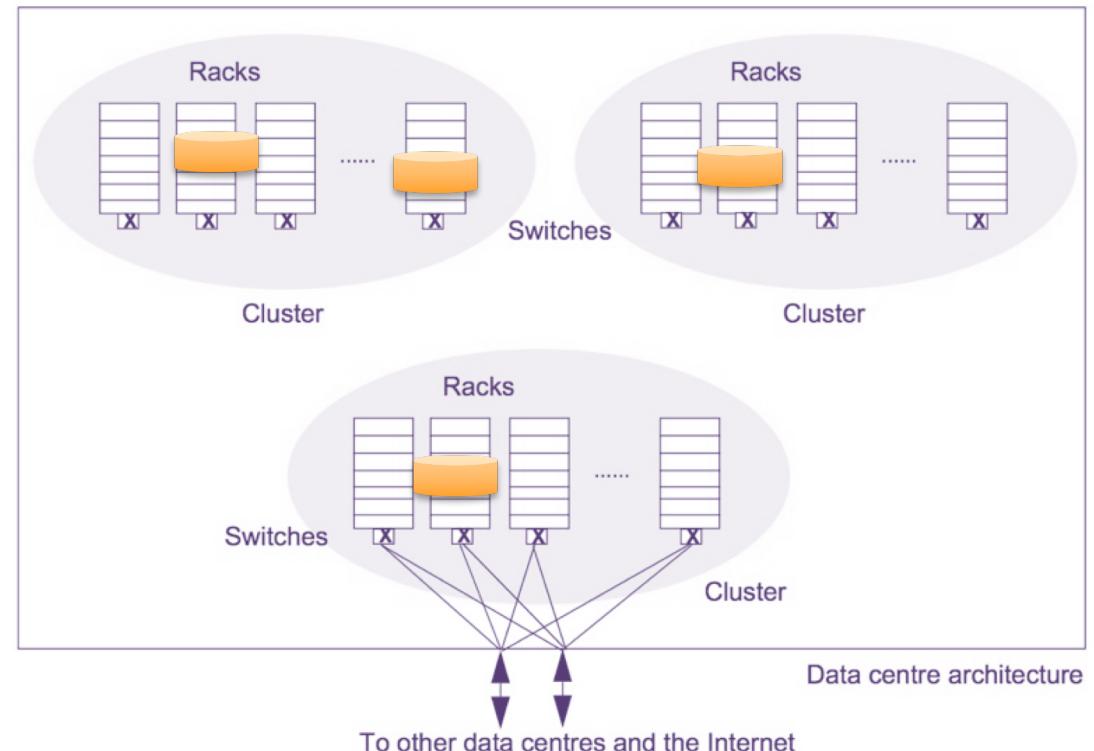
1. Replication for Improving Performance

- Example Applications
 - Caching webpages at the client browser
 - Caching IP addresses at clients and DNS Name Servers
 - Caching in Content Delivery Network (CDNs)
 - Commonly accessed contents, such as software and streaming media, are cached at various network locations



2. Replication for High-Availability

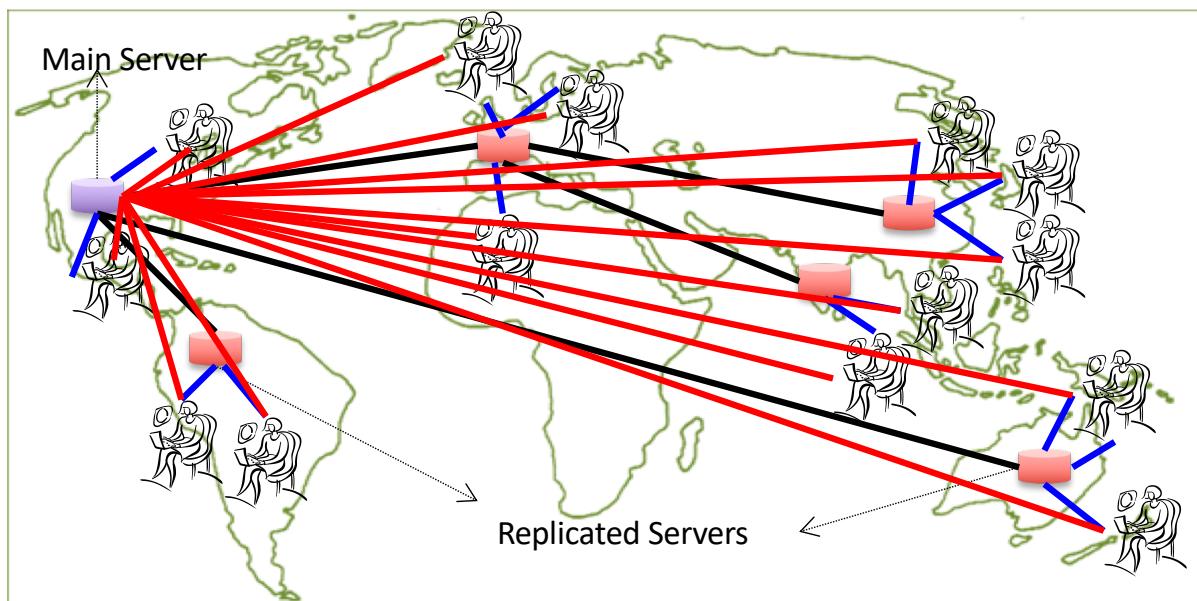
- Availability can be increased by storing the data at replicated locations (instead of storing one copy of the data at a server)



- Example:
Google File-System and Chubby replicate the data at computers across different racks, clusters and data-centers
 - If one computer or a rack or a cluster crashes, then a replica of the data can still be accessed

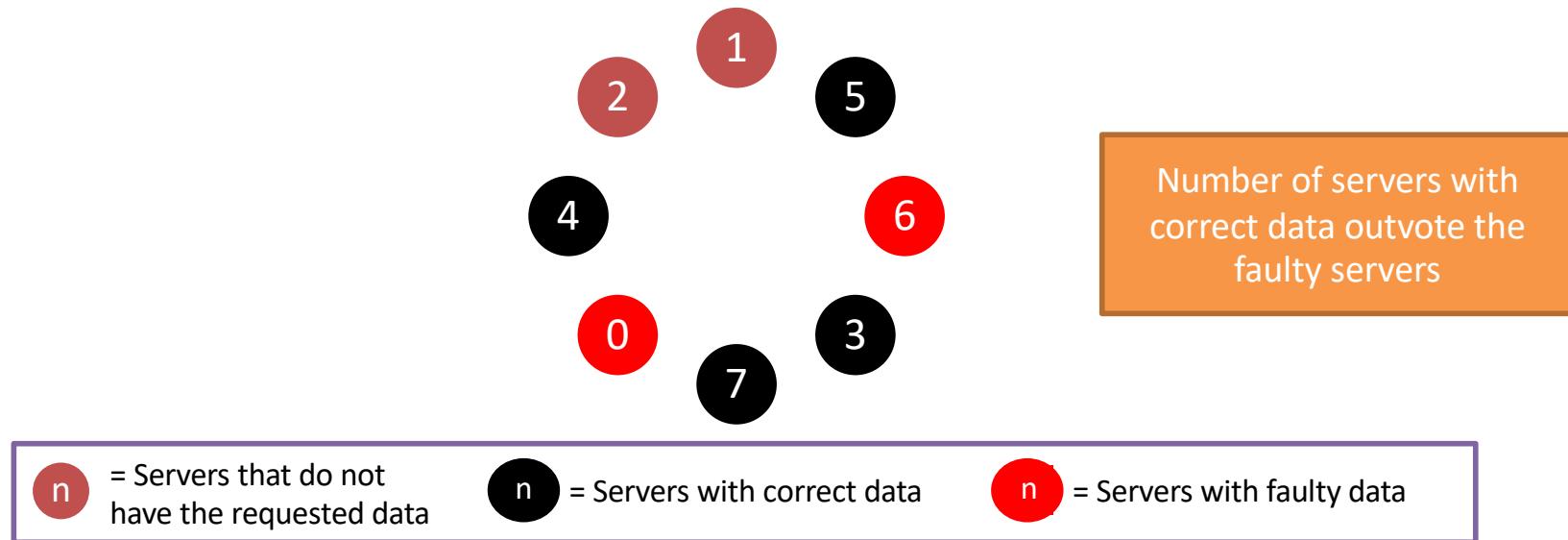
3. Replication for Enhancing Scalability

- Distributing the data across replicated servers helps in avoiding bottle-necks at the main server
 - It balances the load between the main and the replicated servers
- Example: Content Delivery Networks decrease the load on main servers of the website



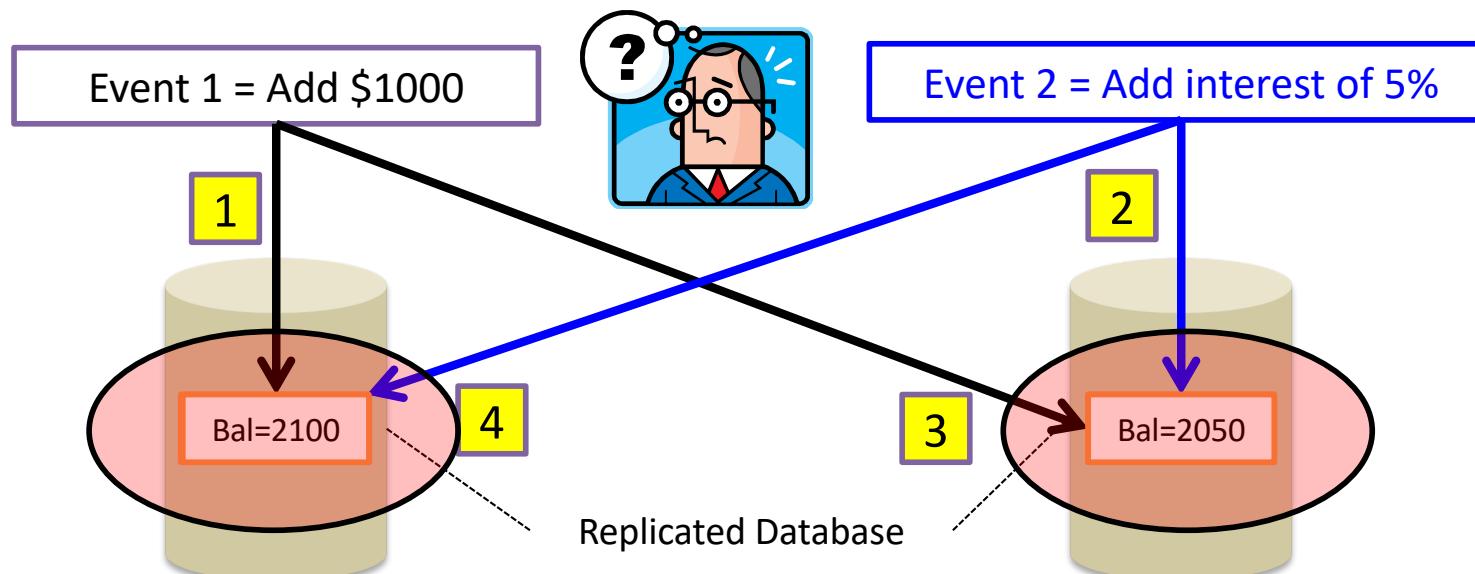
4. Replication for Securing Against Malicious Attacks

- If a minority of the servers that hold the data are malicious, the non-malicious servers can outvote the malicious servers, thus providing security.
- The technique can also be used to provide fault-tolerance against non-malicious but faulty servers



Why Consistency?

- In a DS with replicated data, one of the main problems is keeping the data consistent
- An example:
 - In an e-commerce application, the bank database has been replicated across two servers
 - Maintaining consistency of replicated data is a challenge



Overview of Consistency and Replication

Today's lecture

- Consistency Models
 - Data-Centric Consistency Models
 - Client-Centric Consistency Models
- Replica Management
 - When, where and by whom replicas should be placed?
 - Which consistency model to use for keeping replicas consistent?
- Consistency Protocols
 - We study various implementations of consistency models

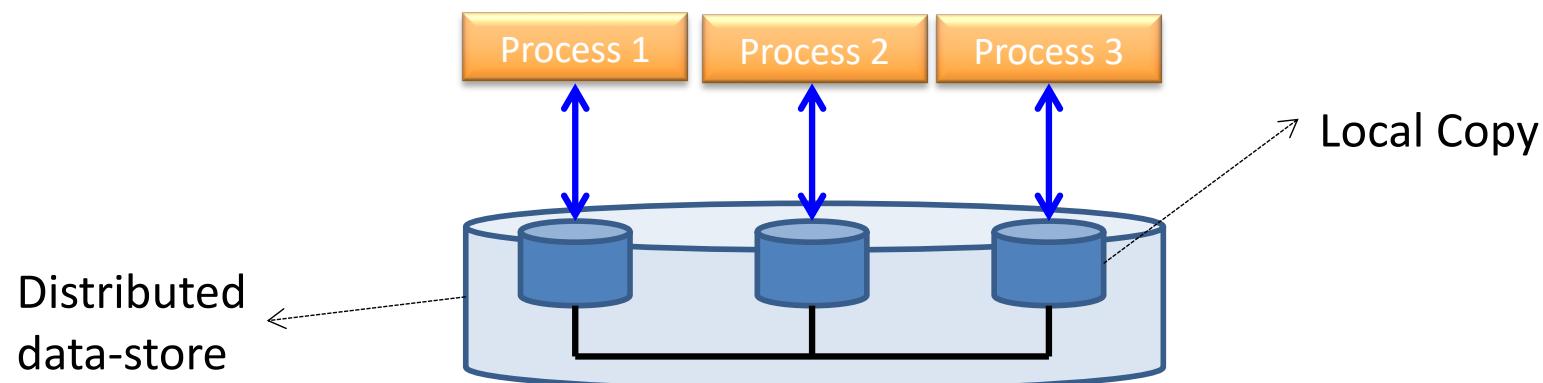
Next lectures

Overview

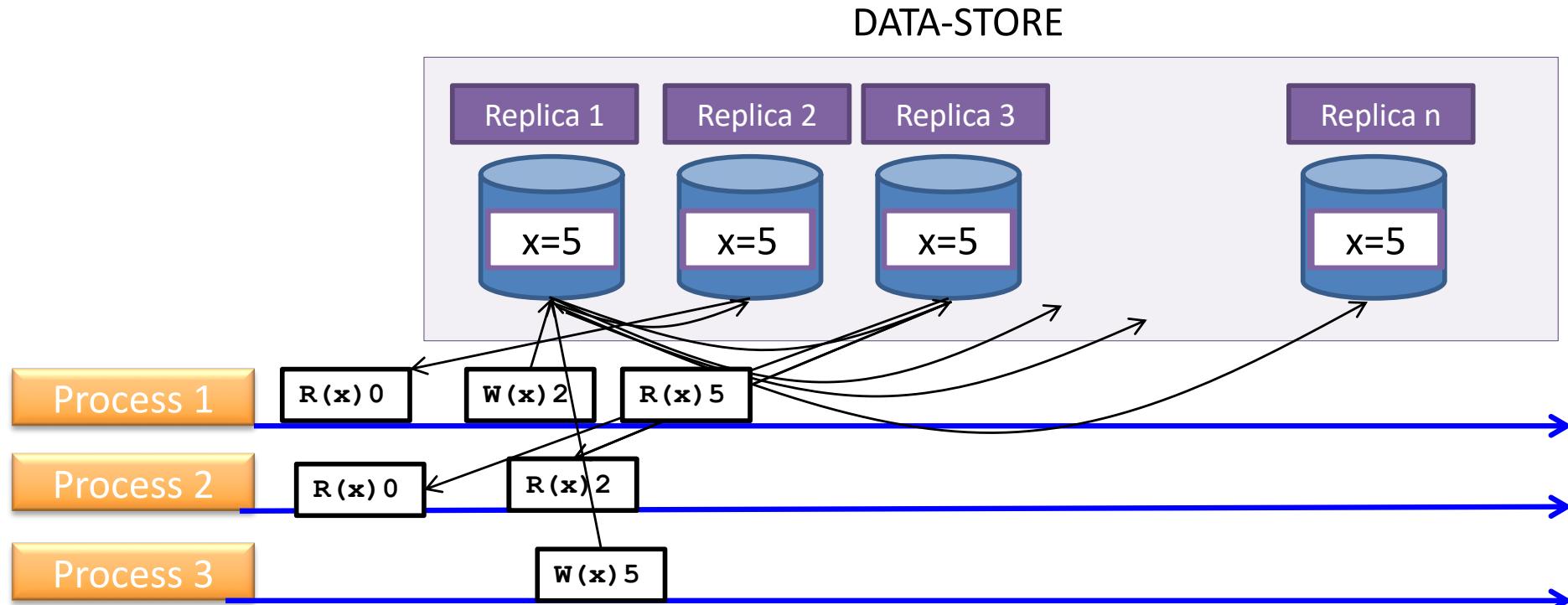
- **Consistency Models**
 - Data-Centric Consistency Models
 - Client-Centric Consistency Models
- Replica Management
- Consistency Protocols

Introduction to Consistency and Replication

- In a distributed system, shared data is typically stored in distributed shared memory, distributed databases or distributed file systems.
 - The storage can be distributed across multiple computers
 - Simply, we refer to a series of such data storage units as data-stores
- Multiple processes can access shared data by accessing any replica on the data-store
 - Processes generally perform read and write operations on the replicas



Maintaining Consistency of Replicated Data



Strict Consistency

- Data is always fresh
 - After a write operation, the update is propagated to all the replicas
 - A read operation will result in reading the most recent write
- If there are many writes, this leads to large overheads

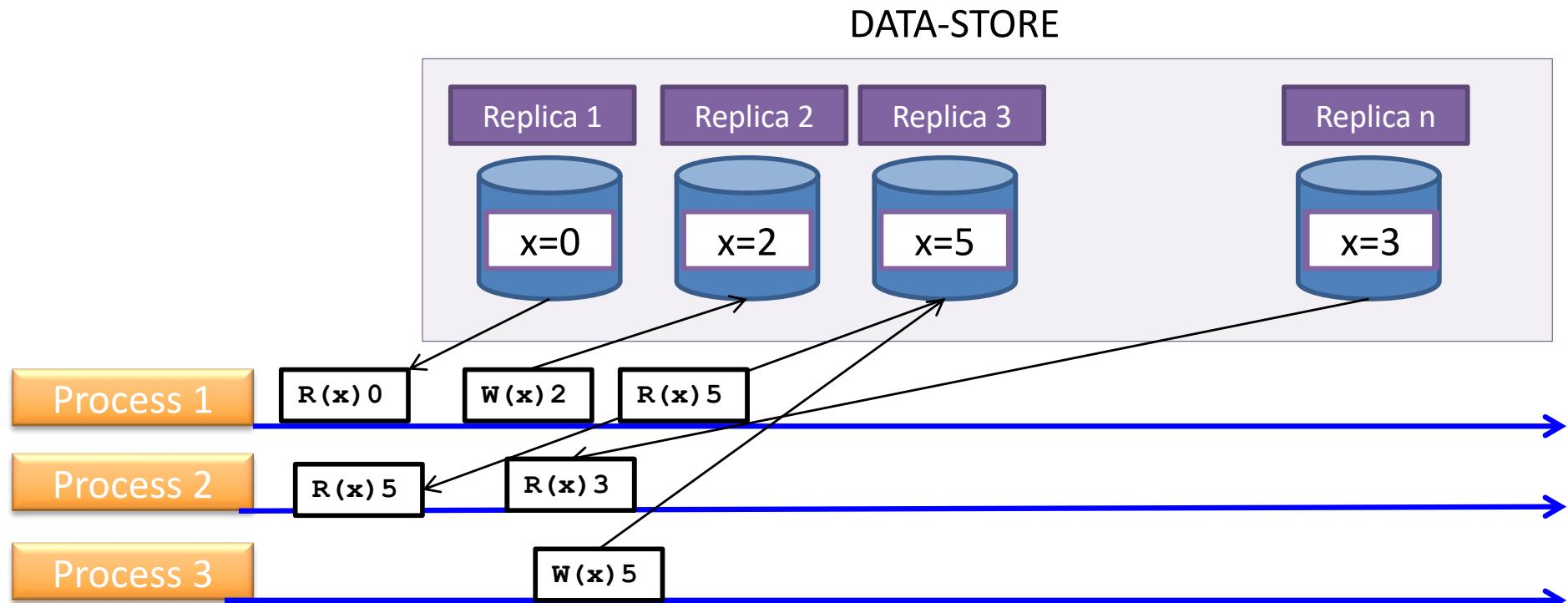
P1 = Process P1

→ = Timeline at P1

R(x)b = Read variable x;
Result is b

W(x)b = Write variable x;
Result is b

Maintaining Consistency of Replicated Data (cont'd)



- Data might be stale
 - A read operation may result in reading a value that was written long back
 - Replicas are generally out-of-sync
- The replicas may sync at coarse grained time, thus reducing the overhead

P1 = Process P1

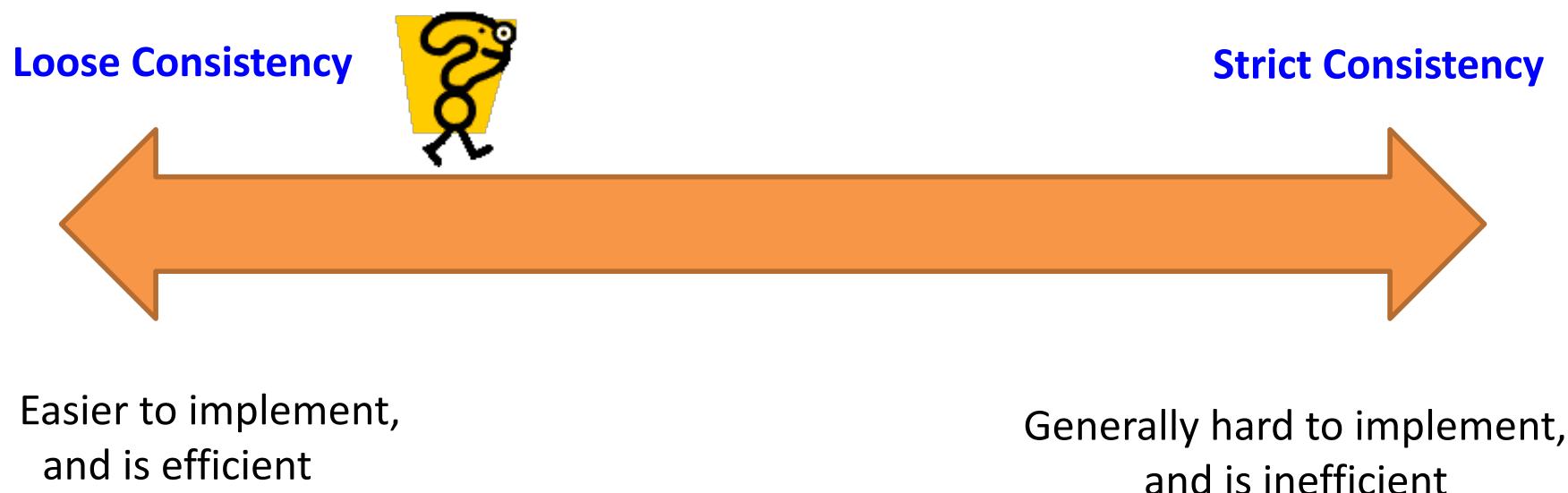
→ = Timeline at P1

$R(x) b$ = Read variable x;
Result is b

$W(x) b$ = Write variable x;
Result is b

Trade-offs in Maintaining Consistency

- Maintaining consistency should balance between the strictness of consistency versus efficiency
 - How much consistency do we need?
 - Good-enough consistency depends on your application



Questions

- Strict consistency vs. loose consistency?
 - Strict consistency: data is always up-to-date
 - Loose consistency: data maybe be outdated
- Which one to implement for
 - a Bank?
 - Facebook?

Consistency Model

- A consistency model is a contract between
 - the process that wants to use the data, and
 - the replicated data repository (or data-store)
- A consistency model states the level of consistency provided by the data-store to the processes while reading and writing the data

Types of Consistency Models

- Consistency models can be divided into two types:
 - Data-Centric Consistency Models
 - These models define how the data updates are propagated across the replicas to keep them consistent
 - Client-Centric Consistency Models
 - These models assume that clients connect to different replicas at each time
 - The models ensure that whenever a client connects to a replica, the replica is brought up to date with the replica that the client accessed previously

Overview

- **Consistency Models**
 - Data-Centric Consistency Models
 - Client-Centric Consistency Models
- **Replica Management**
- **Consistency Protocols**

Data-Centric Consistency Models

- Data-centric Consistency Models describe how the replicated data is kept consistent, and what the process can expect
- Under Data-centric Consistency Models, we study two types of models:
 - Consistency Specification Models:
 - These models enable specifying the consistency levels that are tolerable to the application
 - Models for Consistent Ordering of Operations:
 - These models specify the order in which the data updates are propagated to different replicas

Overview

- Consistency Models
 - Data-Centric Consistency Models
 - Consistency Specification Models
 - Models for Consistent Ordering of Operations
 - Client-Centric Consistency Models
- Replica Management
- Consistency Protocols

Consistency Specification Models

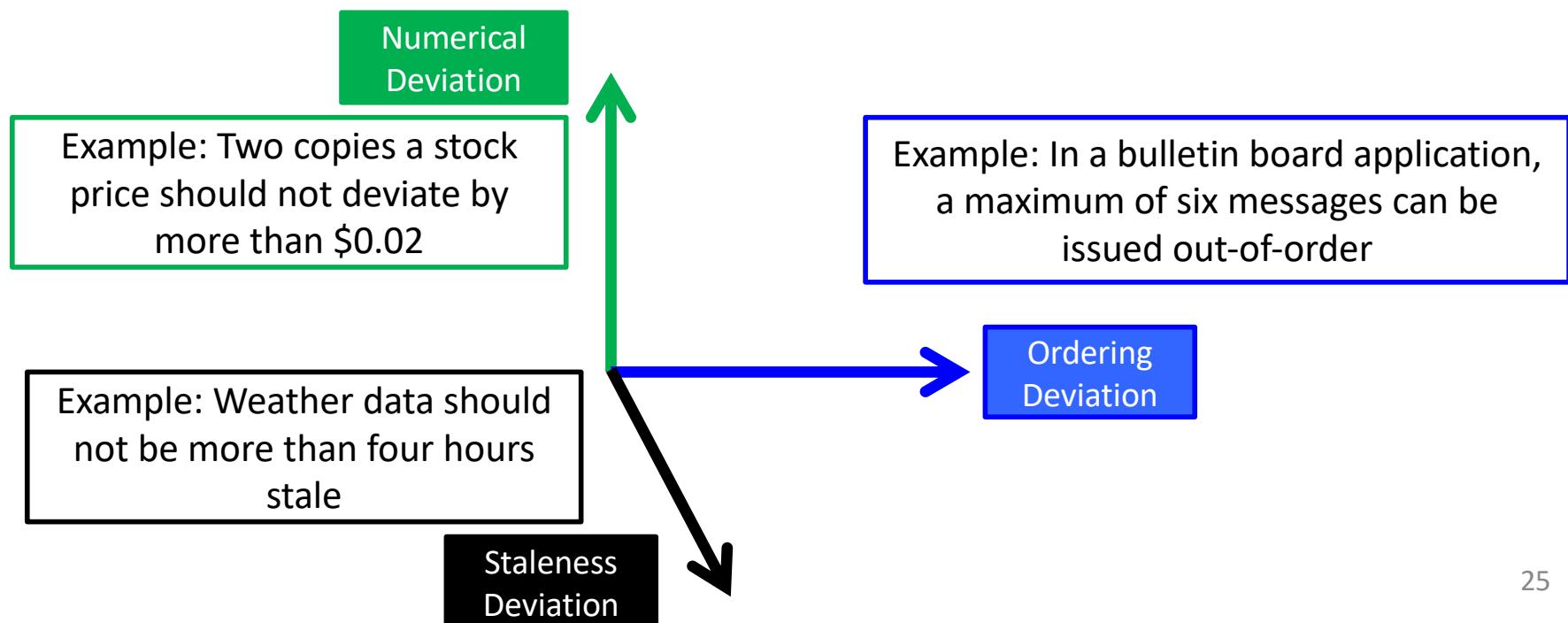
- In replicated data-stores, there should be a mechanism to:
 - Measure how inconsistent the data might be on different replicas
 - How replicas and applications can specify the tolerable inconsistency levels
- Consistency Specification Models enable measuring and specifying the level of inconsistency in a replicated data-store
- We study a Consistency Specification Model called Continuous Consistency Model

Continuous Consistency Model

- Continuous Consistency Model
 - is used to measure inconsistencies and
 - express what inconsistencies can be expected in the system

Continuous Consistency Ranges

- Level of consistency is defined over three independent axes:
 - **Numerical Deviation:** Deviation in the numerical values between replicas
 - **Order Deviation:** Deviation with respect to the ordering of update operations
 - **Staleness Deviation:** Deviation in the staleness between replicas



Consistency Unit (Conit)

- Consistency unit (Conit) specifies the data unit over which consistency is measured
 - For example, conit can be defined as a record representing a single stock
- Level of consistency is measured by each replica along the three dimensions
 - Numerical Deviation
 - For a given replica R, how many updates at other replicas are not yet seen at R? What is the effect of the non-propagated updates on local Conit values?
 - Order Deviation
 - For a given replica R, how many local updates are not propagated to other replicas?
 - Staleness Deviation
 - For a given replica R, how long has it been since updates were propagated?

Assume

Order De
not presen

There is one operation at B which is not yet pushed to A with a value difference of $x=2, y=0$

No operations at A are not yet pushed to B

Replica A			Replica B					
x	y	Ord	n(w)	x	y	VC	Ord	n(w)
0	0	(0,0)	0	0	0	(0,0)	0	0(0,0)
0	0	(0,0)	0	1(2,0)	2	(0,5)	1	0(0,0)
2	0	(1,5)	0	0(0,0)	2	(0,5)	0	0(0,0)
2	1	(10,5)	1	0(0,0)	2	(0,5)	0	1(0,1)
2	1	(10,5)	1	1(0,1)	2	(0,16)	1	1(0,1)
3	1	(14,5)	2	1(0,1)	2	(0,16)	1	2(1,1)
3	4	(23,5)	3	1(0,1)	2	(0,16)	1	3(1,4)

$<5, B>$ =

Operation performed at B when the vector clock was 5

$<m, n>$

= Uncommitted operation

$<m, n>$

= Committed operation

x; y

= A Conit

Conit and Consistency

Table of operations and measures

number of operations in R that are

One operation at B was not yet pushed to A

There are no operations at A which are not yet pushed to B

Replica A

x; y		Operation	Result
$<5, B>$	$x+=2$	x=2	
$<10, A>$	$y+=1$	y=1	
$<14, A>$	$x+=1$	x=3	
$<23, A>$	$y+=3$	y=4	

Replica B

x; y		Operation	Result
$<5, B>$	$x+=2$	x=2	
$<16, B>$	$y+=1$	y=1	

FOR PRINTING – WITHOUT BUBBLES

Assume a **global observer** keeps a **table of operations and measures**

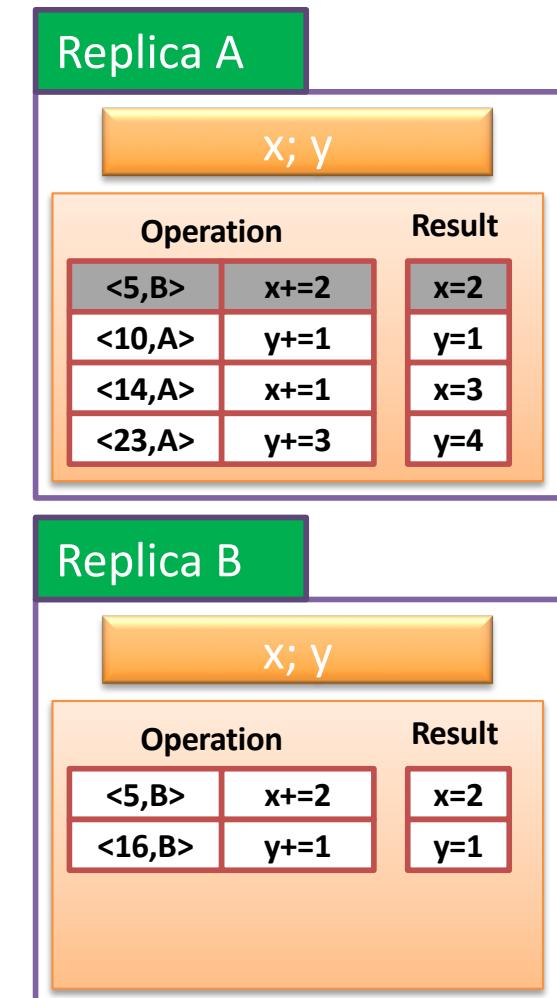
Order Deviation at a replica R is the number of operations in R that are not present at the other replicas

Numerical Deviation at replica R is defined as $n(w)$, where

n = # of operations at other replicas that are not yet seen by R,
 w = weight of the deviation

= list of update amount for all variables in a Conit

Replica A					Replica B				
x	y	VC	Ord	n(w)	x	y	VC	Ord	n(w)
0	0	(0,0)	0	0(0,0)	0	0	(0,0)	0	0(0,0)
0	0	(0,0)	0	1(2,0)	2	0	(0,5)	1	0(0,0)
2	0	(1,5)	0	0(0,0)	2	0	(0,5)	0	0(0,0)
2	1	(10,5)	1	0(0,0)	2	0	(0,5)	0	1(0,1)
2	1	(10,5)	1	1(0,1)	2	1	(0,16)	1	1(0,1)
3	1	(14,5)	2	1(0,1)	2	1	(0,16)	1	2(1,1)
3	4	(23,5)	3	1(0,1)	2	1	(0,16)	1	3(1,4)



<5,B> =

Operation performed at B when the vector clock was 5

<m,n>

= Uncommitted operation

<m,n>

= Committed operation

x:y

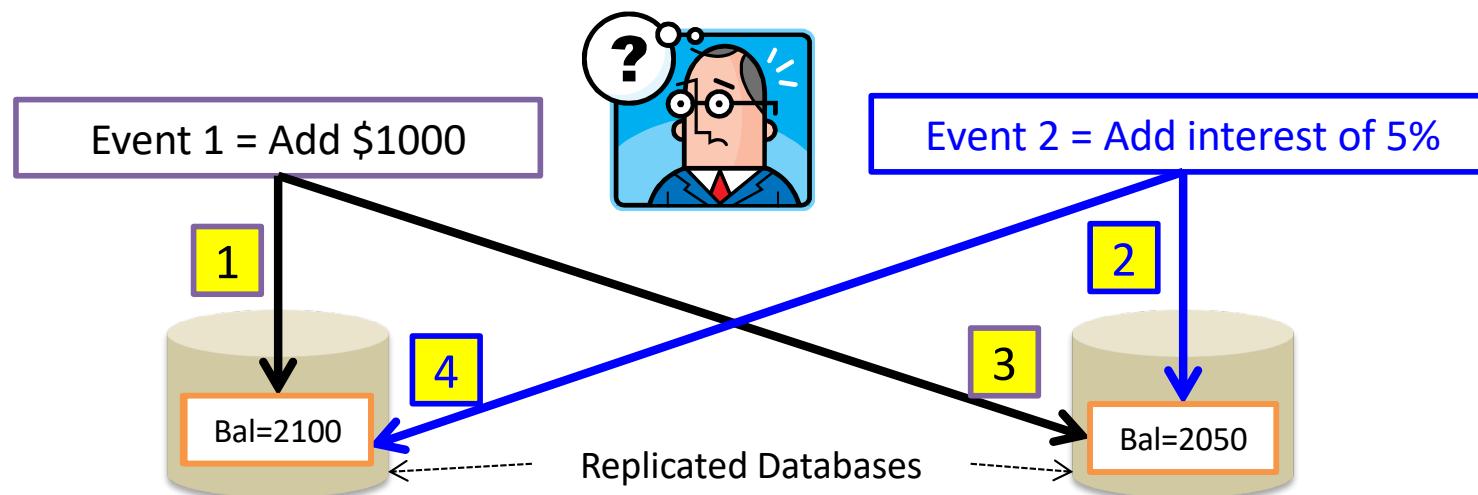
= A Conit

Overview

- Consistency Models
 - Data-Centric Consistency Models
 - Continuous Specification Models
 - Models for Consistent Ordering of Operations
 - Client-Centric Consistency Models
- Replica Management
- Consistency Protocols

Why is Consistent Ordering Required in Replication?

- In several applications, the order or the sequence in which the replicas commit to the data store is critical
- Example:



- Continuous Specification Models defined how inconsistency is measured
 - However, the models did not enforce any order in which the data is committed

Consistent Ordering of Operations (cont'd)

- Whenever a replica is updated, it propagates the updates to other replicas at some point in time
- Updating different replicas is carried out by passing messages between the replica data-stores
- We will study different types of ordering and consistency models arising from these orderings

Types of Ordering

- We will study three types of ordering of messages that meet the needs of different applications:
 - Total Ordering
 - Sequential Ordering
 - Sequential Consistency Model
 - Causal Ordering
 - Causal Consistency Model

Questions

- How do we measure consistency?
 - Global observer
- Metrics to measure consistency?
 - **Numerical Deviation**: Deviation in the numerical values between replicas
 - **Order Deviation**: Deviation with respect to the ordering of update operations
 - Staleness Deviation: Deviation in the staleness between replicas

Types of Ordering

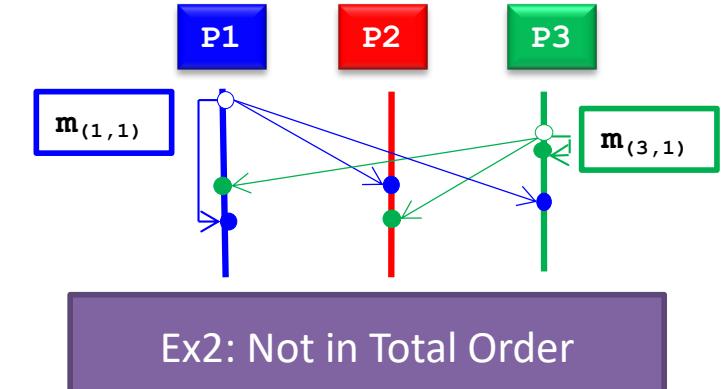
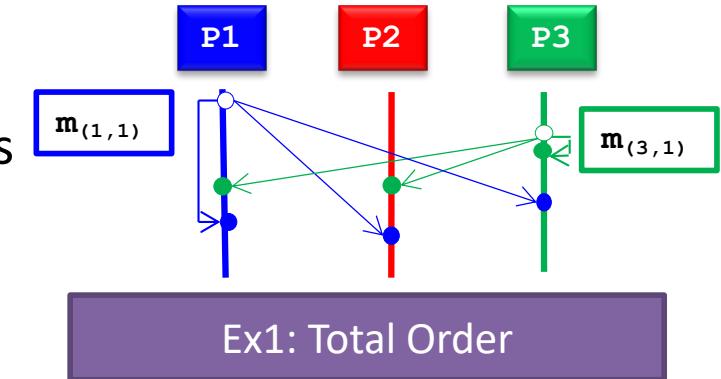
- Total Ordering
- Sequential Ordering
- Causal Ordering

Total Ordering

– Total Order

- If process P_i sends a message m_i and P_j sends m_j , and if one correct process delivers m_i before m_j then every correct process delivers m_i before m_j
- Messages can contain replica updates, such as passing the read or write operation that needs to be performed at each replica, example:
 - if P_1 issues the operation $m_{(1,1)}$: `x=x+1` ; and
 - If P_3 issues $m_{(3,1)}$: `print(x)` ;
 - Then, at all replicas P_1 , P_2 , P_3 execute in the same order, for example:

```
print(x);  
x=x+1;
```

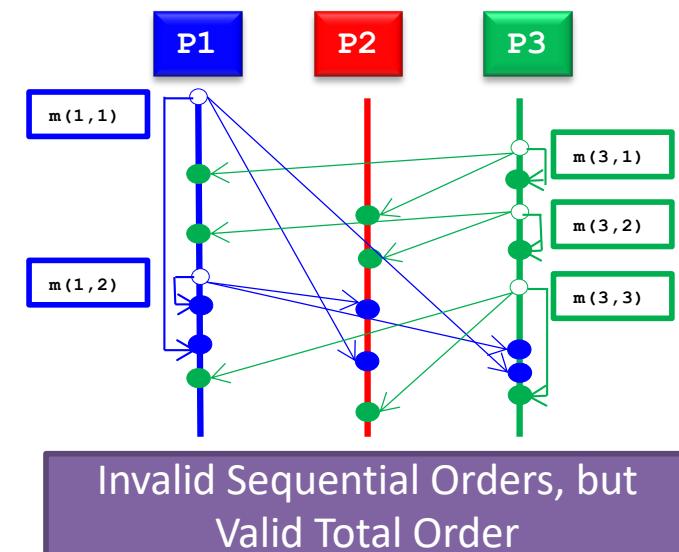
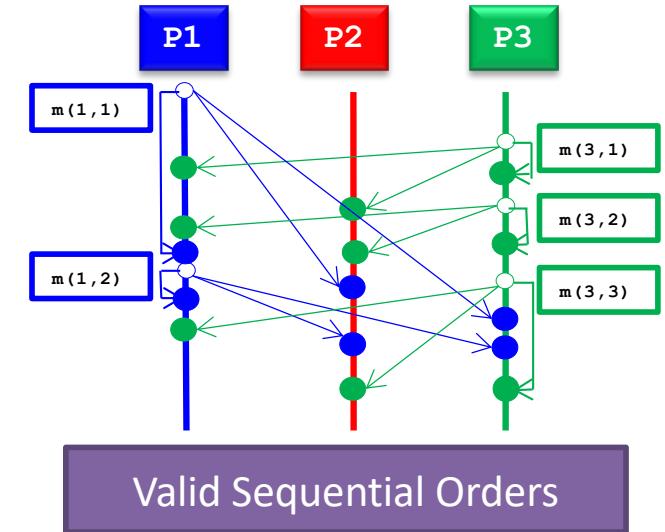


Types of Ordering

- Total Ordering
- Sequential Ordering
- Causal Ordering

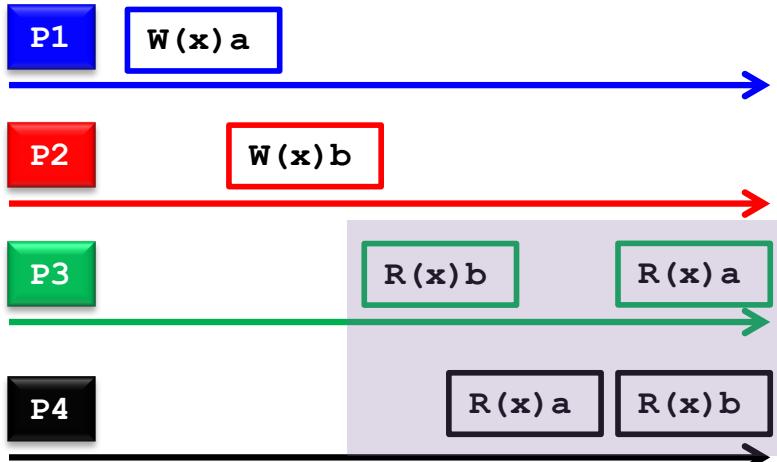
Sequential Ordering

- + If a process P_i sends a sequence of messages $m_{(i,1)}, \dots, m_{(i,n_i)}$, and
- + Process P_j sends a sequence of messages $m_{(j,1)}, \dots, m_{(j,n_j)}$,
- + Then, :
 - + At any process, the set of messages received are in the same sequential order
 - + Messages from each individual process appear in this sequence in the order sent by the sender
 - + At every process, $m_{i,1}$ should be delivered before $m_{i,2}$, which is delivered before $m_{i,3}$ and so on...
 - + At every process, $m_{j,1}$ should be delivered before $m_{j,2}$, which is delivered before $m_{j,3}$ and so on...
- + sequential ordering:
 - + also known as FIFO ordering

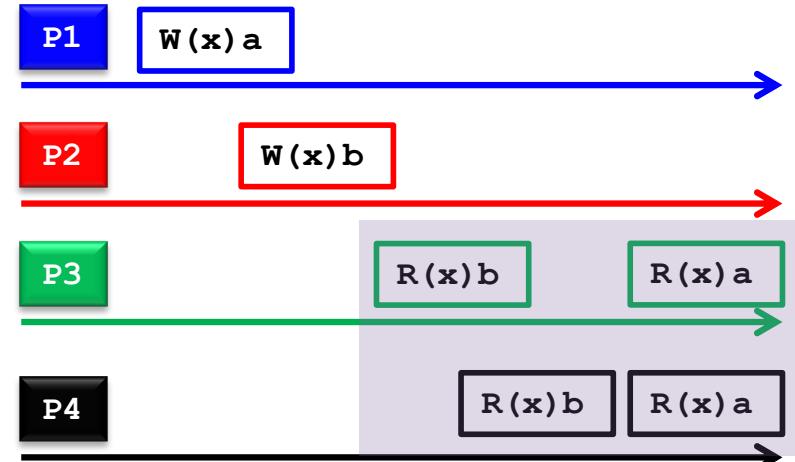


Sequential Consistency Model

- Sequential Consistency Model enforces that all the update operations are executed at the replicas in a sequential order
 - “all read operations should see writes in the same order”
- Consider a data-store with variable **x** (Initialized to **NULL**)
 - In the two data-stores below, identify the sequentially consistent data-store



✗ Results while operating on DATA-STORE-1



✓ Results while operating on DATA-STORE-2

P1 =Process P1

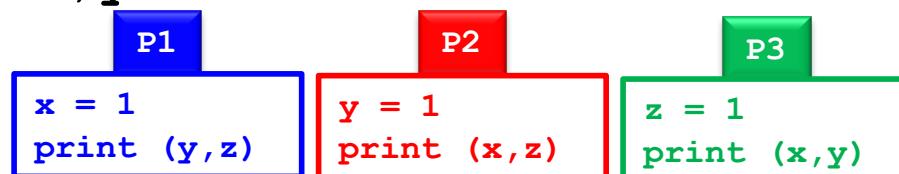
→ =Timeline at P1

R(x) b =Read variable x;
Result is b

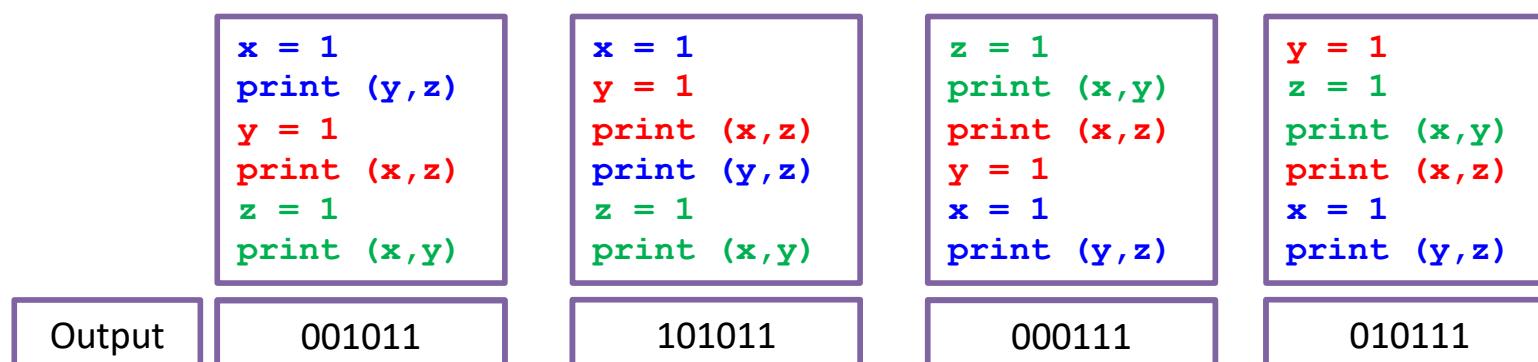
W(x) b = Write variable x;
Result is b

Sequential Consistency (cont'd)

- Consider three processes P_1 , P_2 and P_3 executing multiple instructions on three shared variables x , y and z
 - Assume that x , y and z are set to zero at start



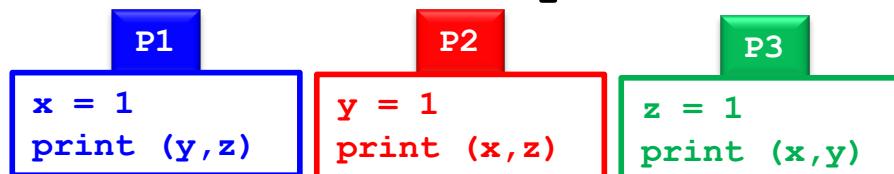
- There are many valid sequences in which operations can be executed at the replica respecting sequential consistency
 - Identify the output?



Sequential Consistency

<http://olafland.polldaddy.com/s/seqcons>

- Consider three processes P_1 , P_2 and P_3 and three shared variables x , y and z
 - Assume that x , y and z are set to 1



- There are many valid sequences in the replica respecting sequential consistency
 - Identify the output?
 - Which ones brakes the seq. order?



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

Output

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

101011

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

000111

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

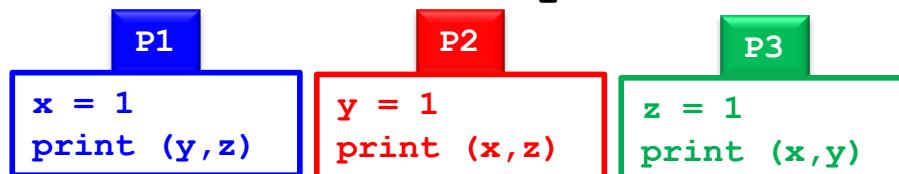
010111



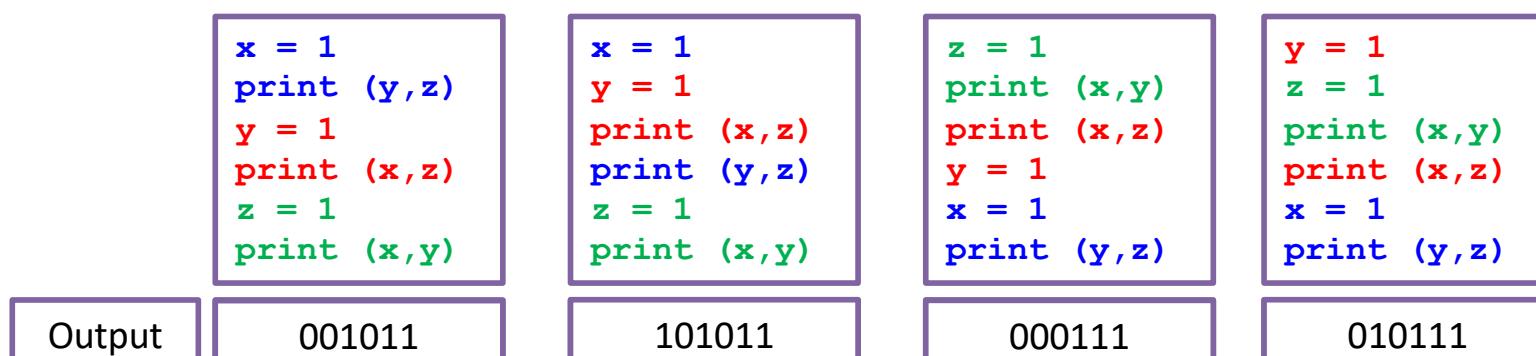
<http://olafland.polldaddy.com/s/seqcons>

Sequential Consistency (cont'd)

- Consider three processes P_1 , P_2 and P_3 executing multiple instructions on three shared variables x , y and z
 - Assume that x , y and z are set to zero at start



- There are many valid sequences in which operations can be executed at the replica respecting sequential consistency
 - Identify the output?
 - Which ones brakes the seq. order?

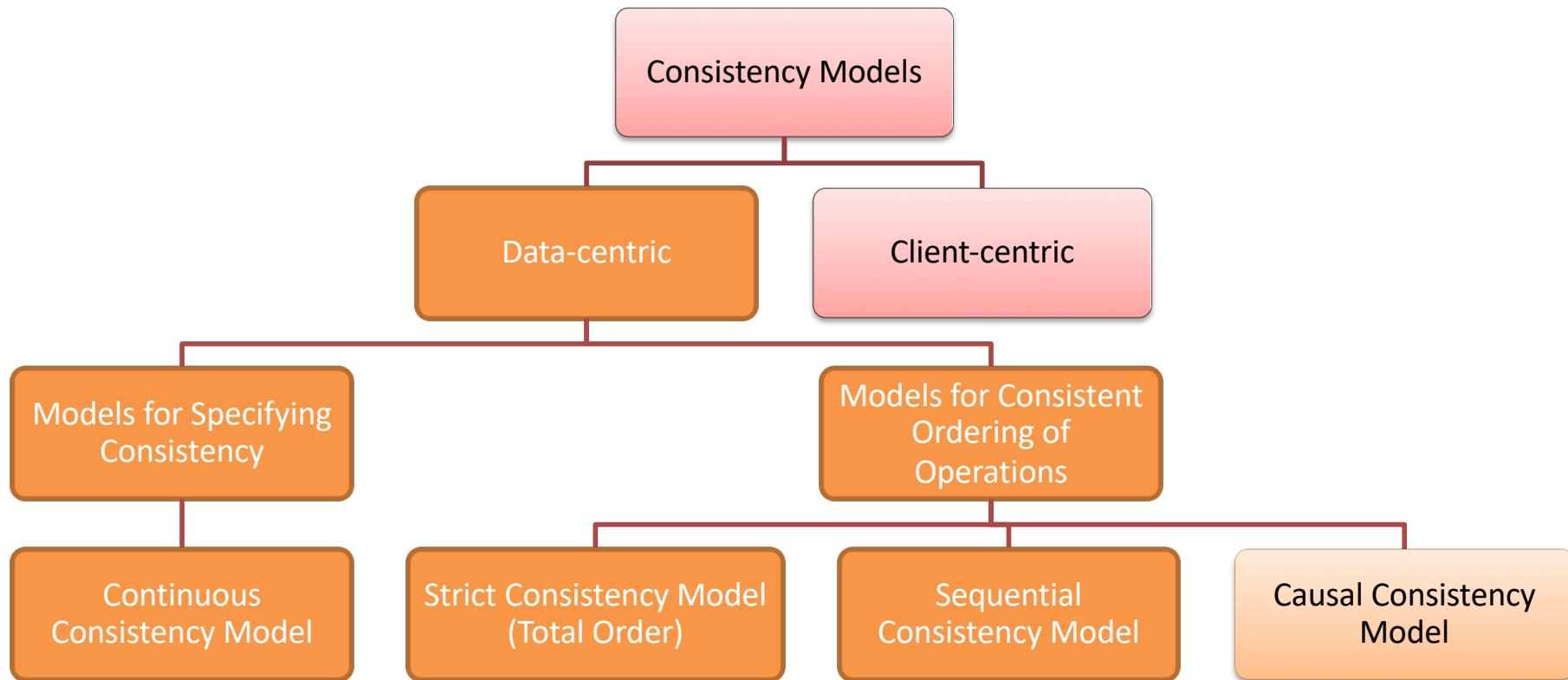


Implications of Adopting Sequential Consistency Model for Applications

- There might be several different sequentially consistent combinations of ordering
 - Number of combinations for a total of n instructions = $O(n!)$
- The contract between the process and the distributed data-store is that the process must accept all of the sequential orderings as valid results
 - A process that works for some of the sequential orderings and does not work correctly for others is INCORRECT

Recap: Consistency Models

- A consistency model states the level of consistency provided by the *data-store* to the processes while reading and writing the data



Questions

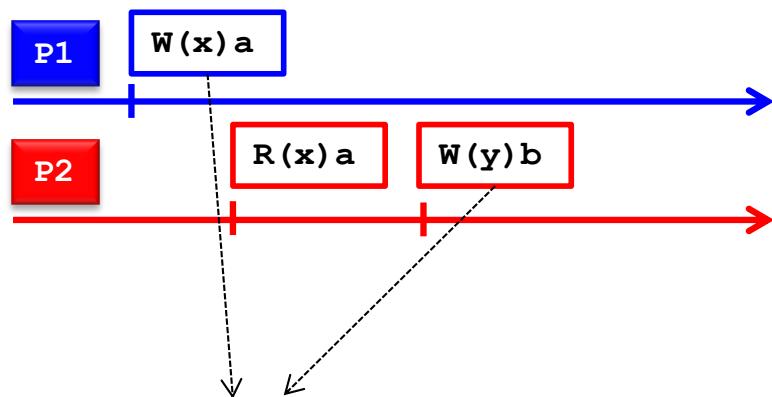
- Total Order?
 - Same order on all nodes
- Sequential Ordering?
 - At any process, the set of messages received are in the same sequential order
 - Messages from each individual process appear in this sequence in the order sent by the sender

Causality (Recap)

- Causal relation between two events
 - If **a** and **b** are two events such that **a happened-before b** or $a \rightarrow b$, and
 - If the (logical) time when event **a** and **b** is received at a process P_i is denoted by $C_i(a)$ and $C_i(b)$
 - Then, if we can infer that $a \rightarrow b$ by observing that $C_i(a) < C_i(b)$, then **a** and **b** are causally related
- How can we implement causality?
 - Logical Clocks or Vector Clocks (see prev. lectures)

Causal vs. Concurrent events

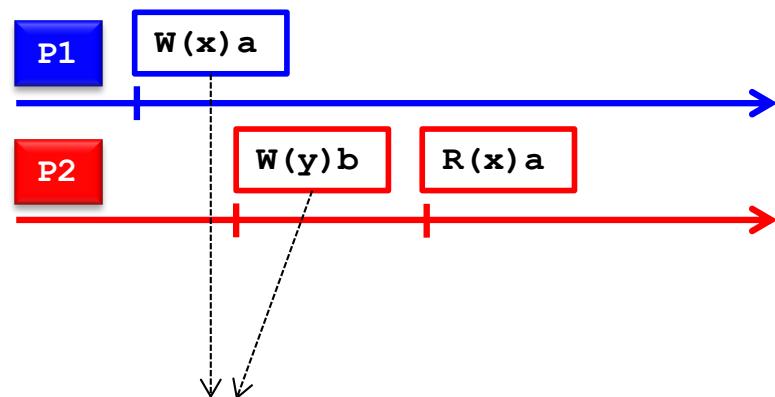
- Consider an interaction between processes P_1 and P_2 operating on replicated data x and y



Events are causally related

Events are not concurrent

- Computation of y at P_2 may have depended on value of x written by P_1
(As P_2 reads x before writing y)



Events are not causally related

Events are concurrent

- Computation of y at P_2 does not depend on value of x written by P_1

P1 =Process P1

→ =Timeline at P1

R(x) b

=Read variable x;
Result is b

W(x) b

= Write variable x;
Result is b

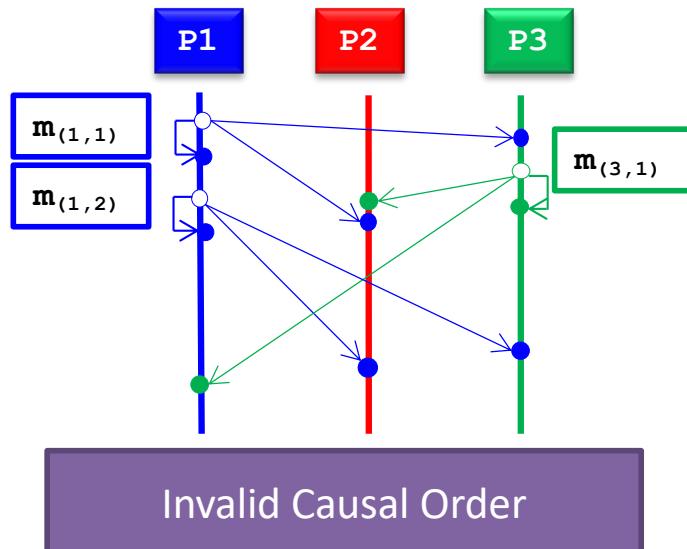
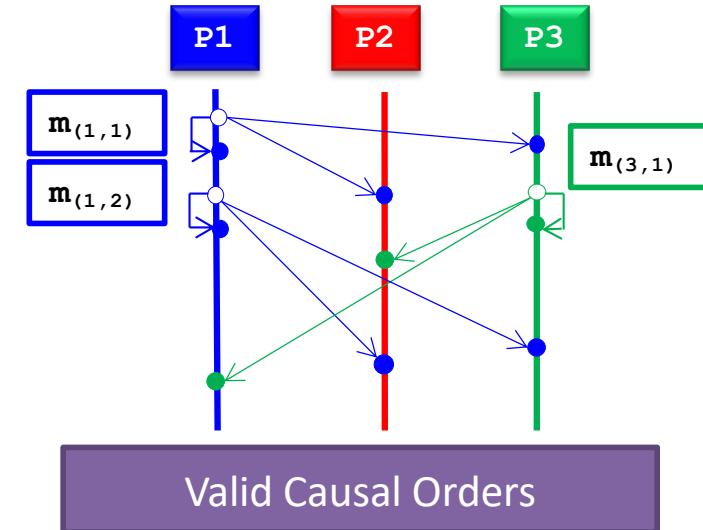
Causal Ordering

Causal Order

- If process P_i sends a message m_i and P_j sends m_j , and if $m_i \rightarrow m_j$ (operator ' \rightarrow ' is **happened-before** relation) then any correct process that delivers m_j will deliver m_i before m_j

Drawback:

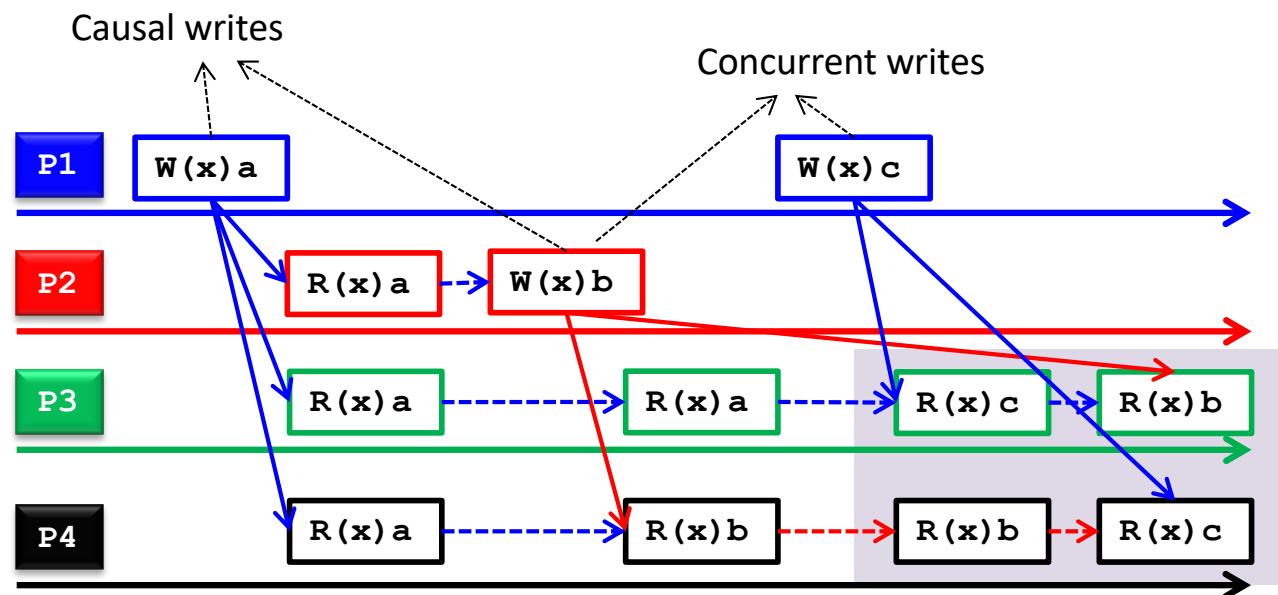
- The **happened-before** relation between m_i and m_j should be induced before communication



Causal Consistency Model

- A data-store is causally consistent if:
 - Writes that are potentially causally related must be seen by all the processes in the same order
 - Concurrent writes may be seen in a different order on different machines

Example of a Causally Consistent Data-store



A Causally Consistent
Data-Store

But not a Sequentially
Consistent Data-Store

P1 =Process P1

→ =Timeline at P1

R(x) b

=Read variable x;
Result is b

W(x) b

= Write variable x;
Result is b

Causally Consistent Data-Store?

<http://olafland.polldaddy.com/s/causcons>

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

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

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





<http://olafland.polldaddy.com/s/causcons>

Causally Consistent Data-Store?

P1:	W(x)a				
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

This sequence is allowed with a causally-consistent store, but not with sequentially consistent store.

Note: $W_1(x)a \rightarrow W_2(x)b$, but not $W_2(x)b \rightarrow W_1(x)c$

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 violation of a causally-consistent store.

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

A correct sequence of events in a causally consistent store.

Implications of adopting a Causally Consistent Data-store for Applications

- Processes have to keep track of which processes have seen which writes
- This requires maintaining a dependency graph between write and read operations
 - Vector clocks provide a way to maintain causal consistency

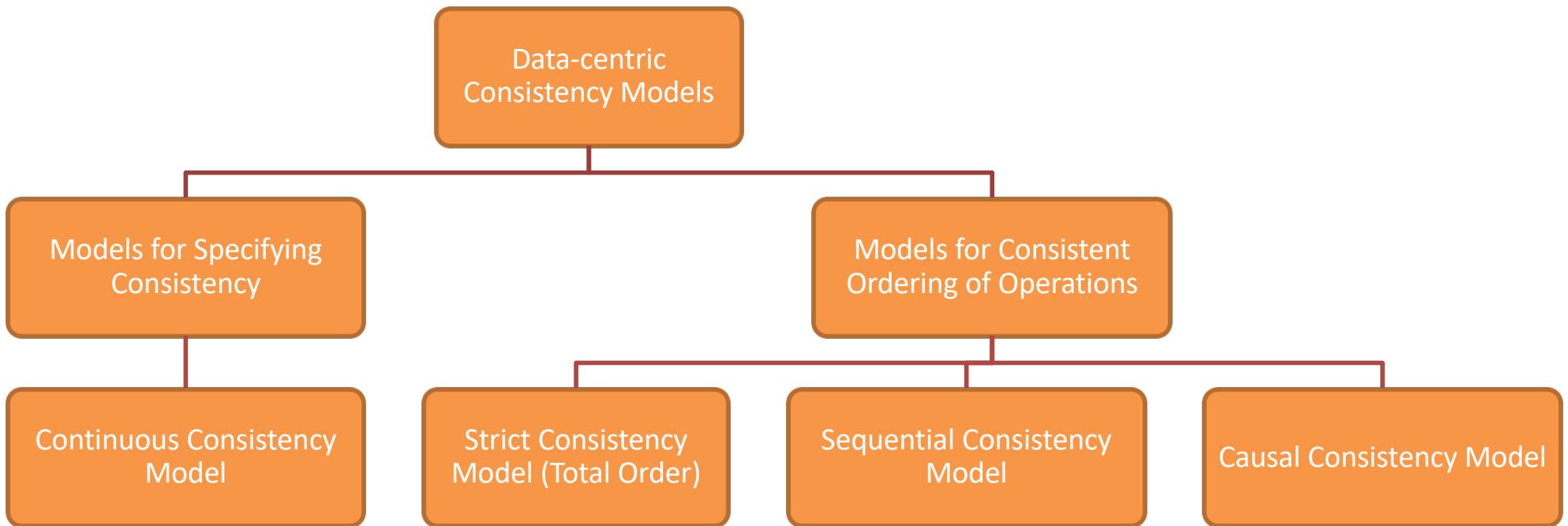
Note

- **Causal ordering**
 - implies **Sequential ordering**
 - does **not** imply total ordering.
- **Sequential ordering**
 - does **not** imply total ordering
 - does **not** imply causal ordering
- **Total ordering**
 - does **not** imply causal ordering
 - does **not** imply Sequential ordering

Summary

- Total Order?
 - Same order on all nodes
- Sequential Ordering?
 - Same order as sent by the sender
- Causal Order
 - Same order on all nodes only for causally related msg.

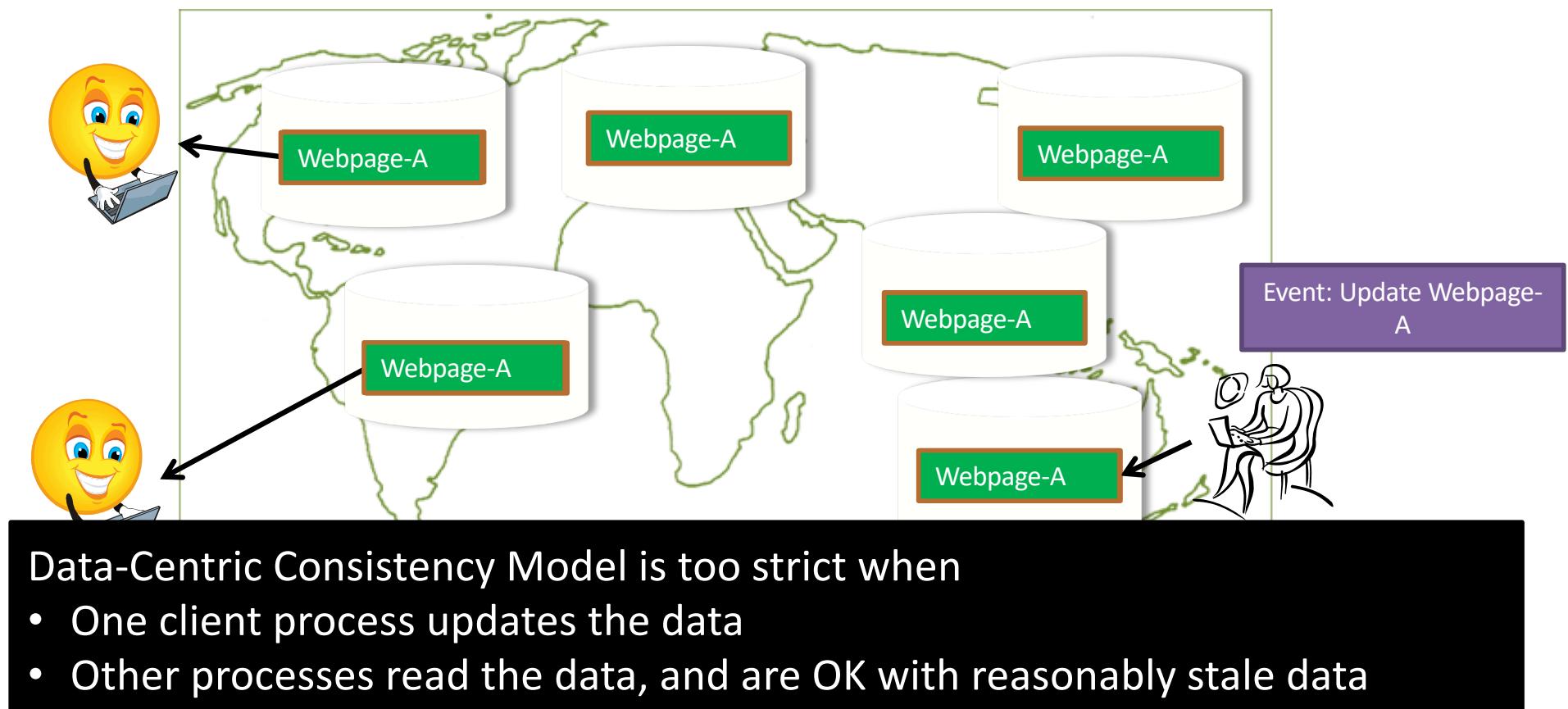
Topics Covered in Data-centric Consistency Models



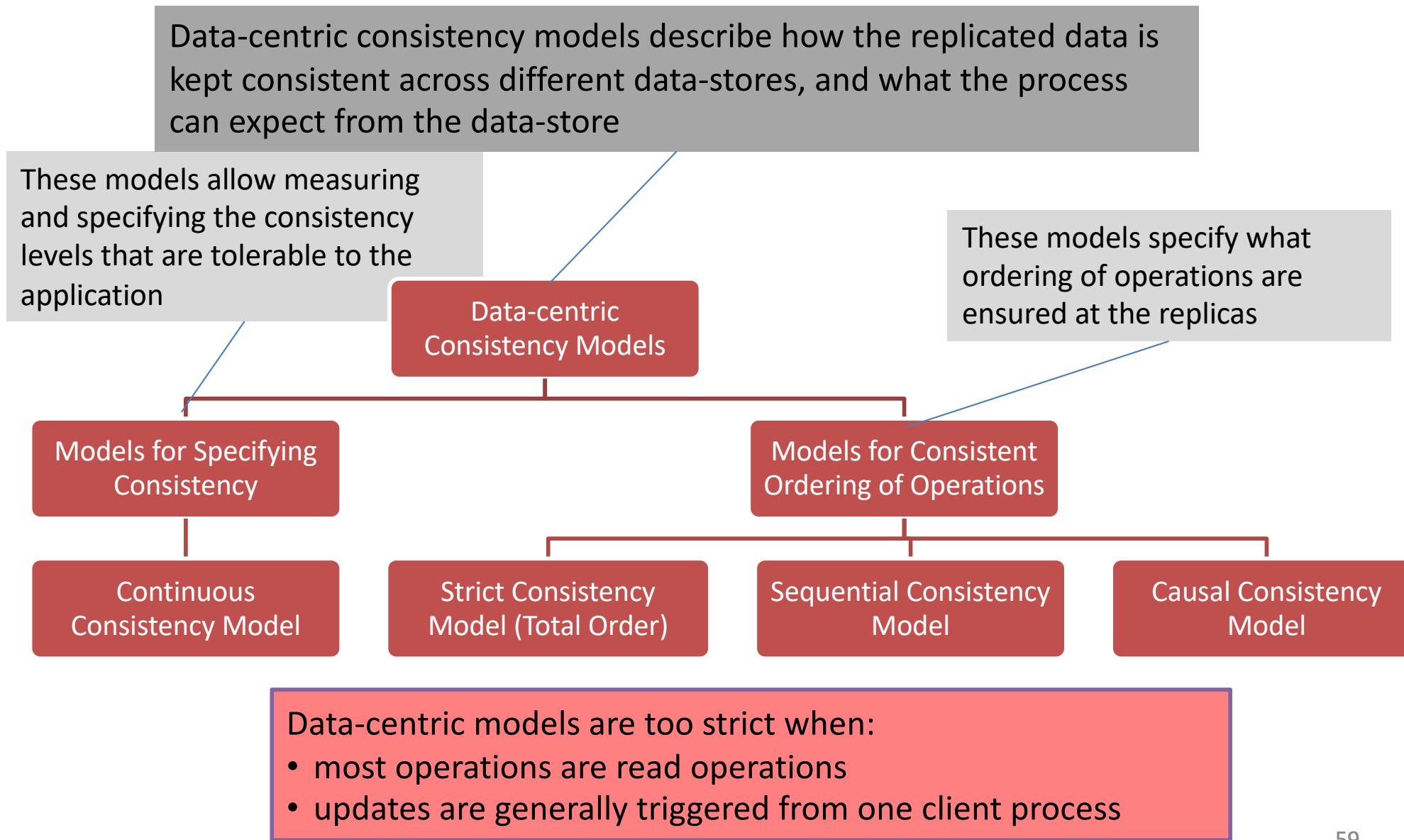
But, is Data-centric Consistency Model good for all applications?

Applications that can use Data-centric Models

- Data-centric models are applicable when many processes are concurrently updating the data-store
- But, do all applications need all replicas to be consistent?



Summary of Data-Centric Consistency Models



Summary

- Replication is necessary for improving performance, scalability and availability, and for providing fault-tolerance
- Replicated data-stores should be designed after carefully evaluating the trade-off between tolerable data inconsistency and efficiency
- Consistency Models describe the contract between the data-store and process about what form of consistency to expect from the system
- Data-centric consistency models:
 - Continuous Consistency Models provide mechanisms to measure and specify inconsistencies
 - Consistency Models can be defined based on the type of ordering of operations that the replica guarantees the application

Next Classes

- Consistency Models
 - Client-Centric Consistency Models
- Replica Management
 - Replica management studies:
 - when, where and by whom replicas should be placed
 - which consistency model to use for keeping replicas consistent
- Consistency Protocols
 - We study selected implementations of consistency models

Questions?

In part, inspired from / based on slides from

- Vinay Kolar
- A. S. Tanenbaum, M. v. Steen