分布式系统

07-分布式系统的一致性与复制 Consistency and Replication in DS

Weixiong Rao 饶卫雄
Tongji University 同济大学软件学院
2023 秋季
wxrao@tongji.edu.cn

同济大学软件学院
School of Software Engineering, Tongji University

Today…

- Last Session
  - Synchronization: Mutual Exclusion and Election Algorithms

- Today's session
  - Consistency and Replication
    - Introduction
    - Data-centric and Client-Centric Consistency Models

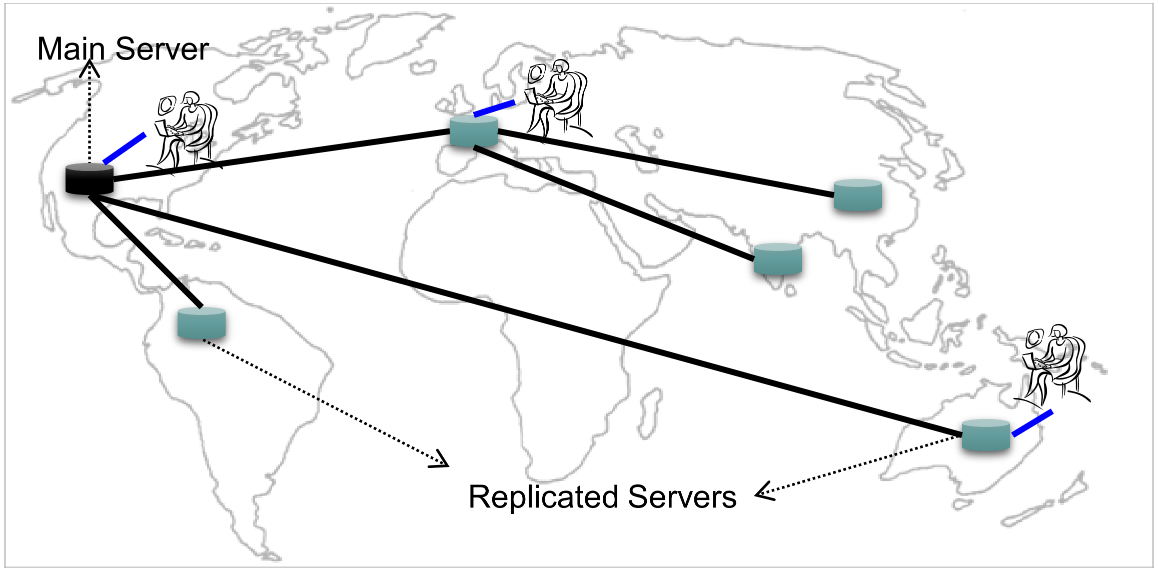
Today

Why Replication?

- Replication is the process of maintaining the data at multiple computers
- Replication is necessary for:
  1. Improving performance
     - A client can access the replicated copy of the data that is near to its location
  2. Increasing the availability of services
     - Replication can mask failures such as server crashes and network disconnection
  3. Enhancing the scalability of the system
     - Requests to the data can be distributed to many servers which contain replicated copies of the data
  4. 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

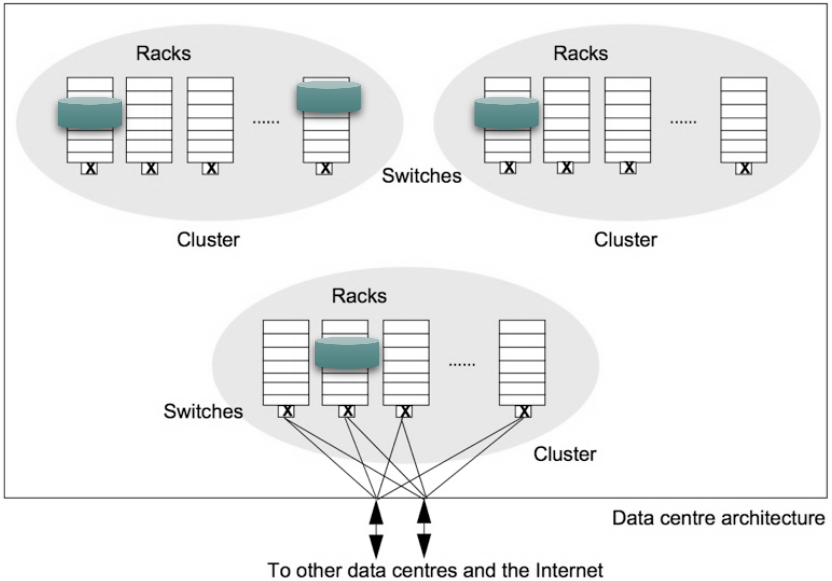
1. Replication for Improving Performance

- 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



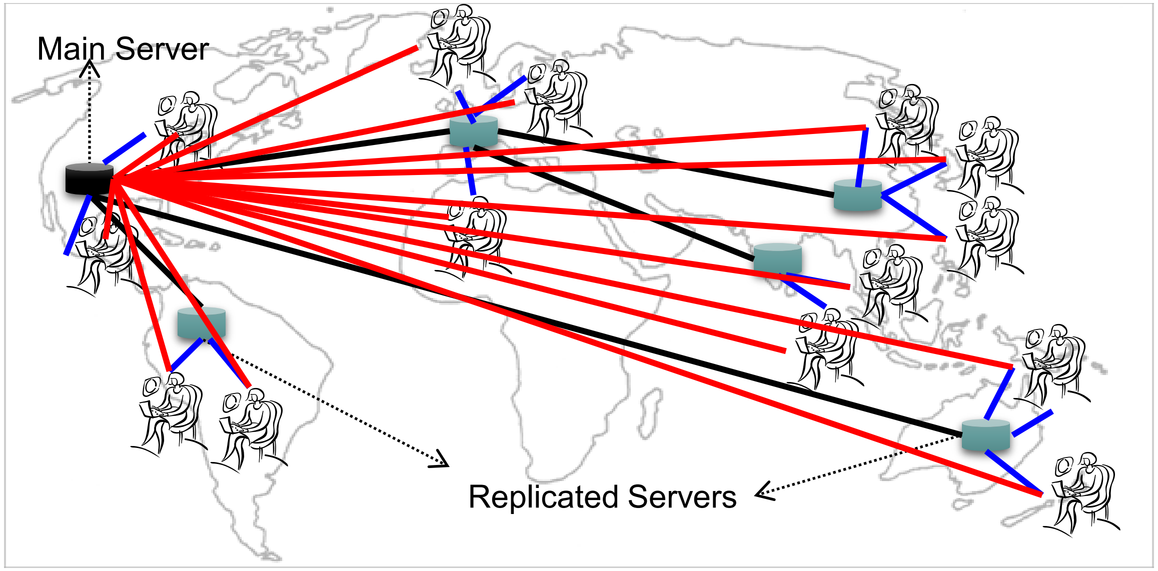
2. Replication for High-Availability

- 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 replicates the data at computers across different racks, clusters and data-centers
  - If one computer or a rack or a cluster crashes, then the data can still be accessed from another source



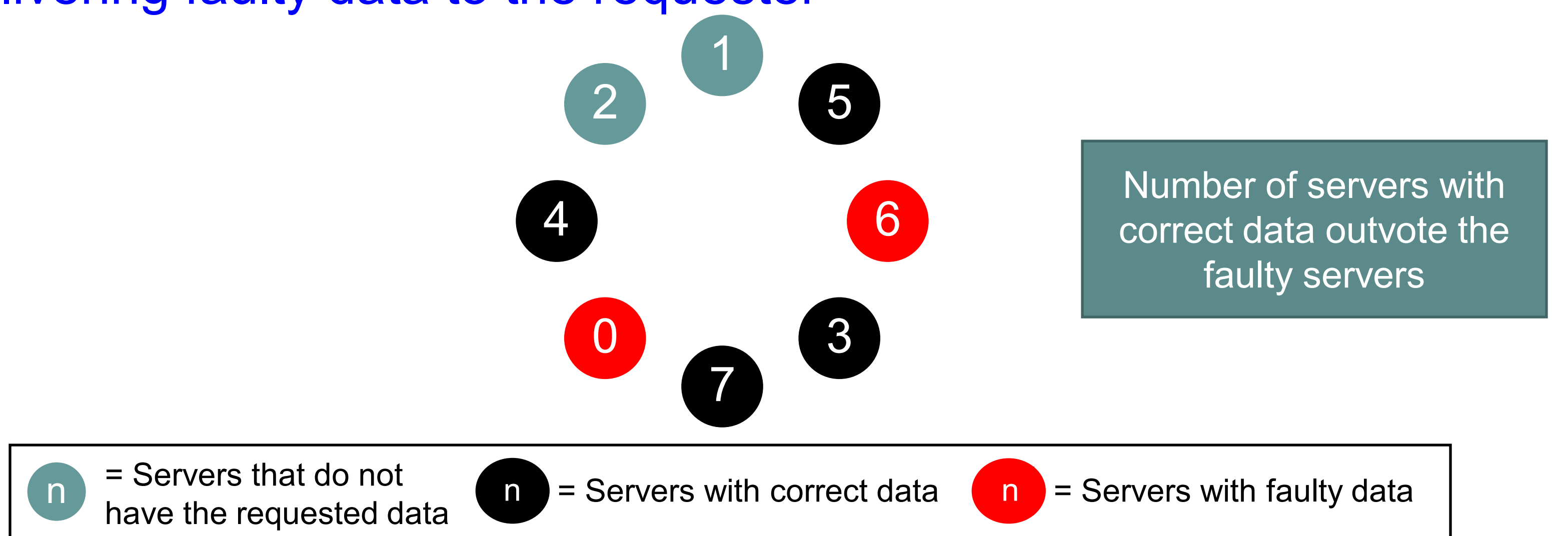
3. Replication for Enhancing Scalability

- 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



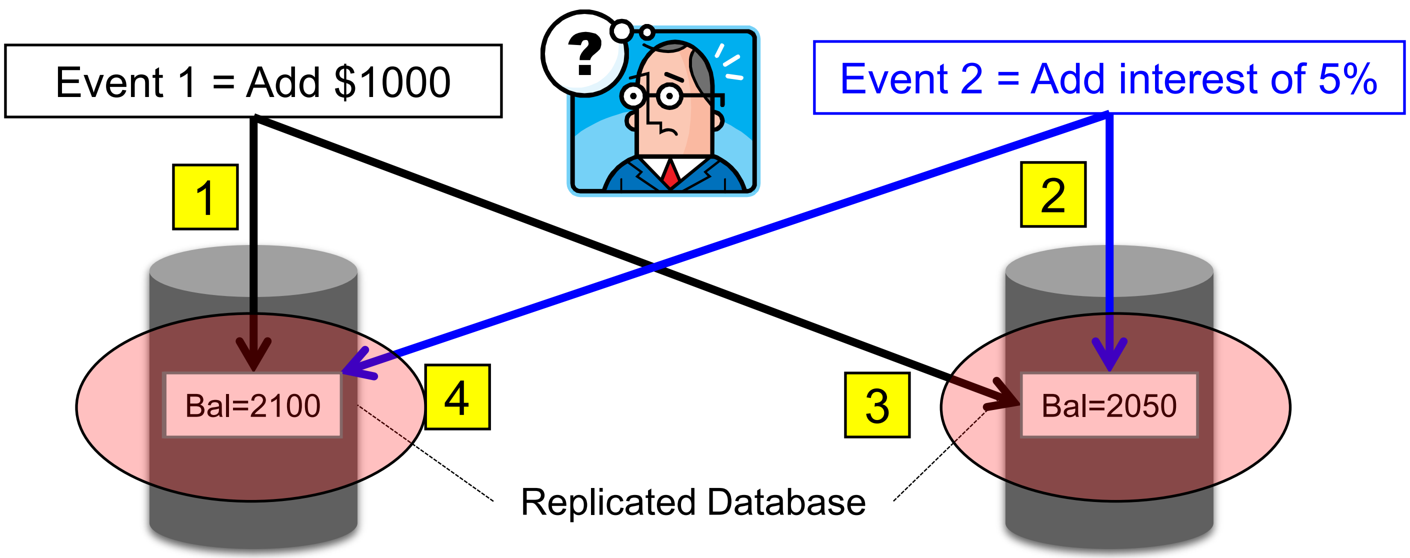
4. Replication for Securing Against Malicious Attacks

- 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
- Example: In a peer-to-peer system, peers can coordinate to prevent delivering faulty data to the requester



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

- stency 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

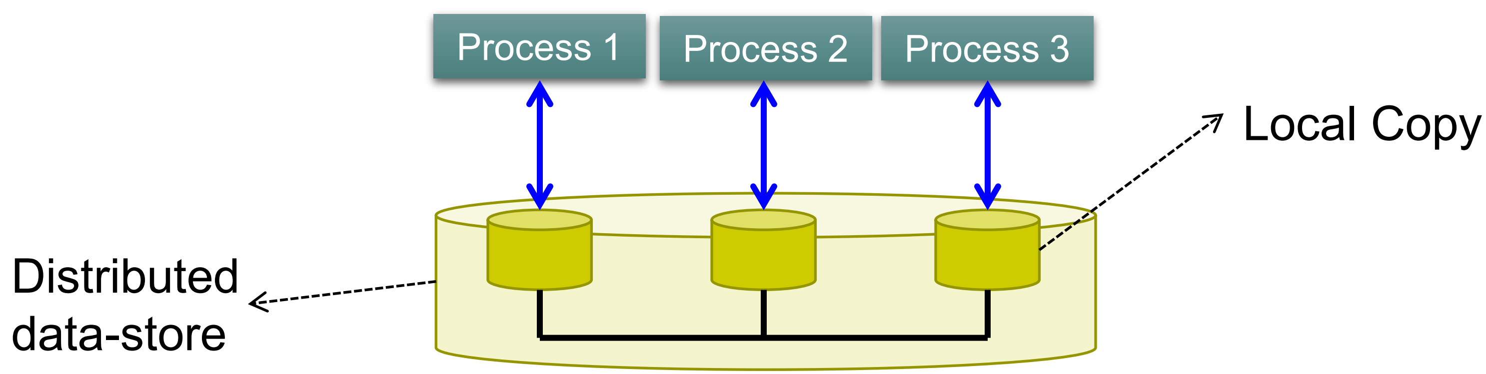
Next lectures

Overview

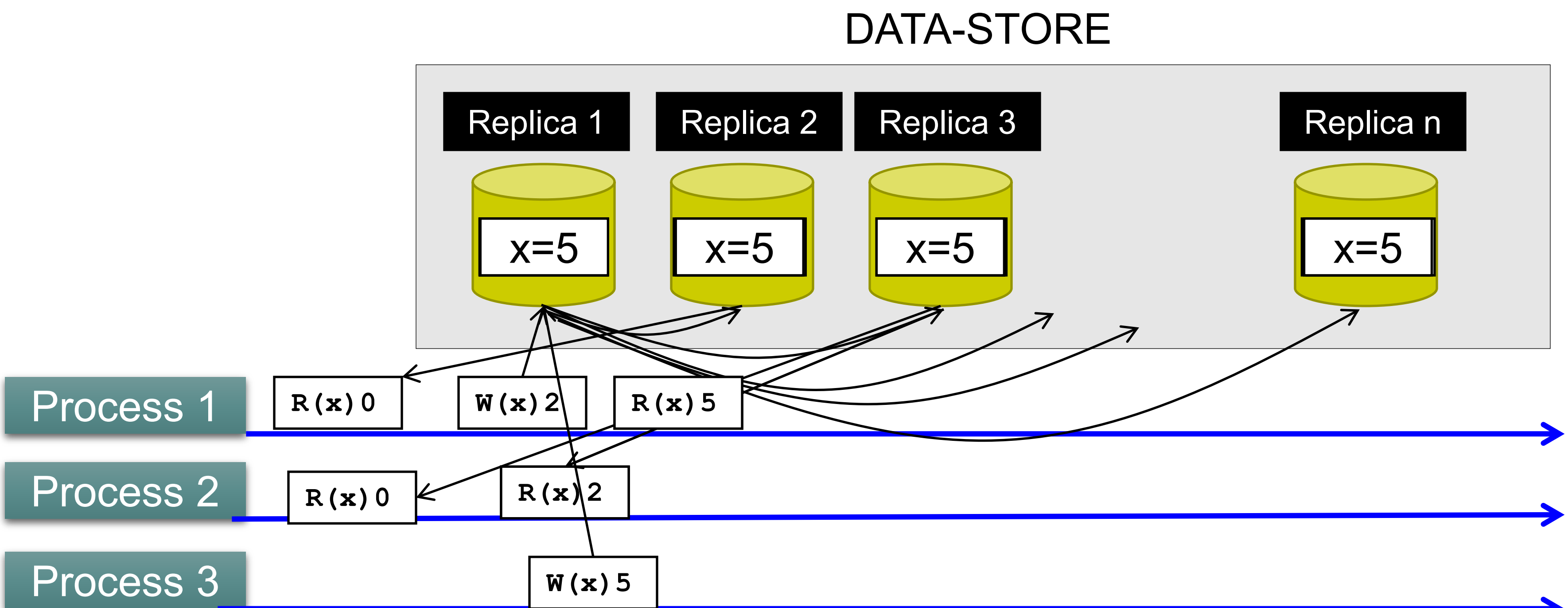
- Consistency Models
  - Data-Centric Consistency Models
  - Client-Centric Consistency Models
- Replica Management
- Consistency Protocols

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

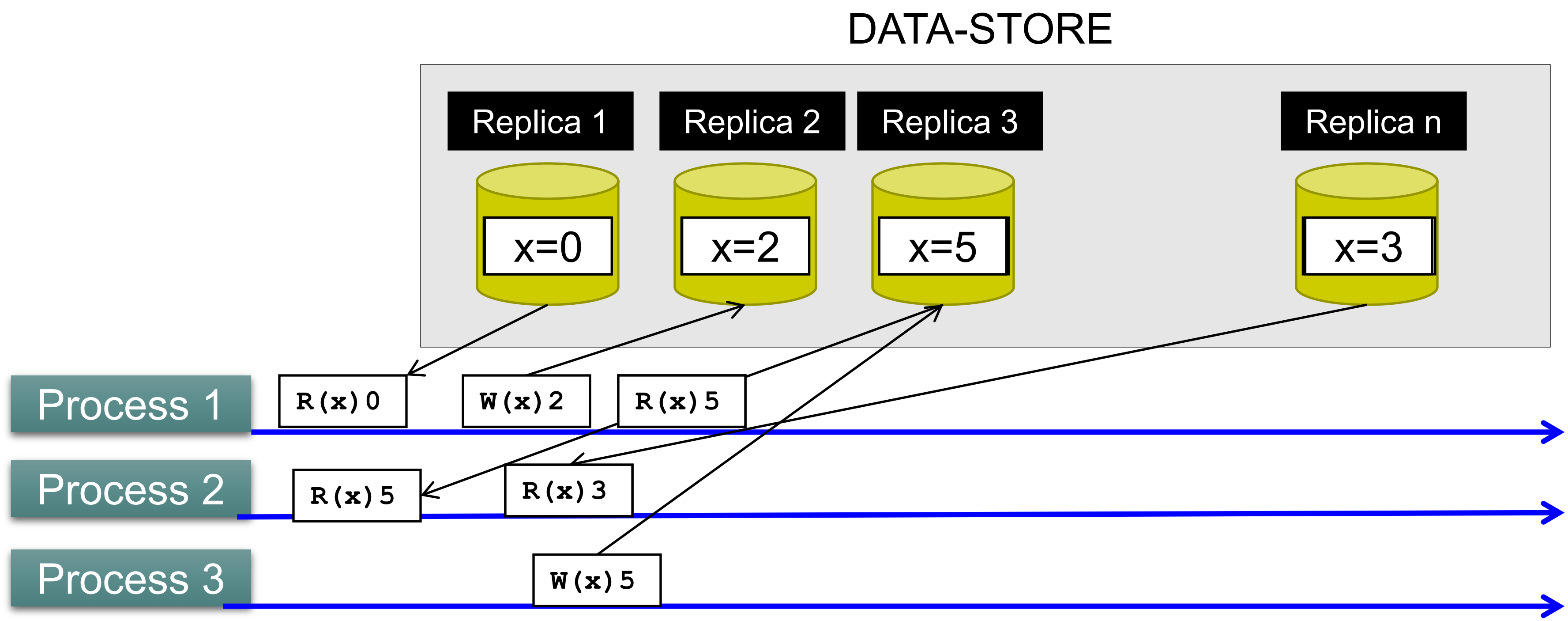


<u>Strict Consistency</u>

- 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 occassional writes and reads, 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)



Loose Consistency

- 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
  - Good-enough consistency depends on your application

Loose Consistency **Strict Consistency**

Easier to implement, and is efficient

Generally hard to implement, and is inefficient

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 different times
    - 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 processes 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*

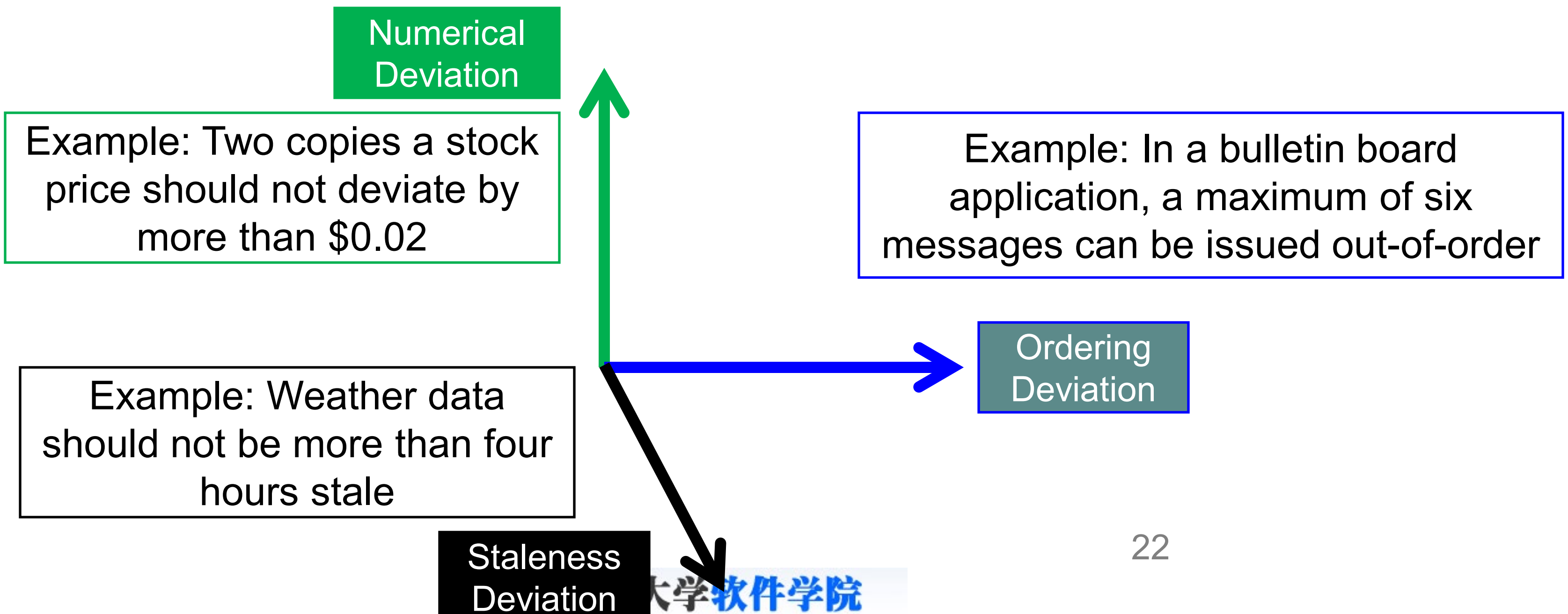
Continuous Consistency Model

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

- Yu and Vahdat [1] provided a framework for measuring and expressing consistency in replicated data-stores

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?

Example of Conit and Consistency 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
  = max(update amount of all variables in a Conit)

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

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

<5,B> = Operation performed at B when the vector clock was **5**

<m,n> = Uncommitted operation

<m,n> = Committed operation

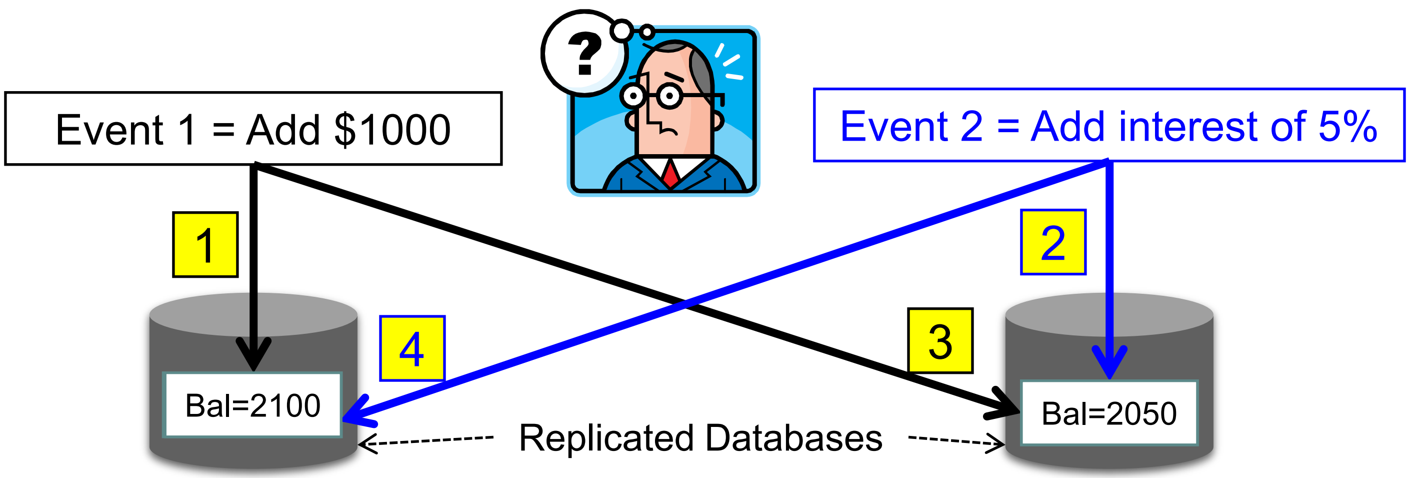
x;y = A Conit

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

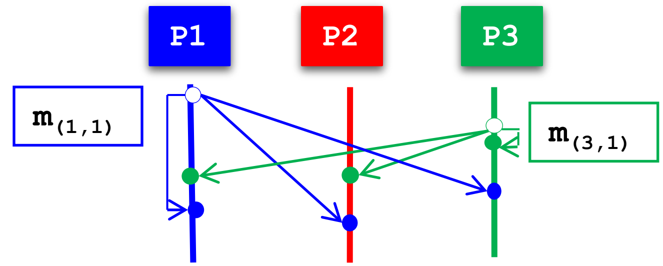
- We will study three types of ordering of messages that meet the needs of different applications:
    1. Total Ordering
    2. Sequential Ordering
        i. Sequential Consistency Model
    3. Causal Ordering
        i. Causal Consistency Model

Types of Ordering

1. Total Ordering
2. Sequential Ordering
3. Causal Ordering

Total Ordering

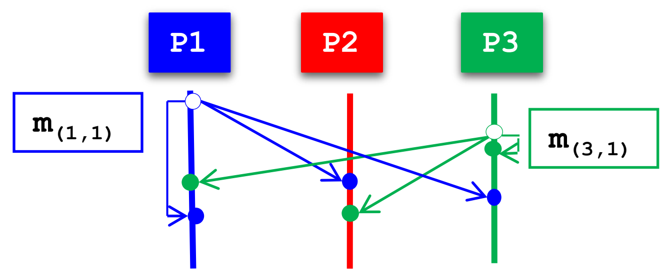
- Total Order
  - If process $\mathbf{P_i}$ sends a message $\mathbf{m_i}$ and $\mathbf{P_j}$ sends $\mathbf{m_j}$, and if one correct process delivers $\mathbf{m_i}$ before $\mathbf{m_j}$ then every correct process delivers $\mathbf{m_i}$ before $\mathbf{m_j}$



Ex1: Total Order

- Messages can contain replica updates, such as passing the read or write operation that needs to be performed at each replica
  - In the example Ex1, if $\mathbf{P_1}$ issues the operation $\mathbf{m_{(1,1)}}$: `x=x+1;` and
  - If $\mathbf{P_3}$ issues $\mathbf{m_{(3,1)}}$: `print(x);`
  - Then, at all replicas $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ the following order of operations are executed

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



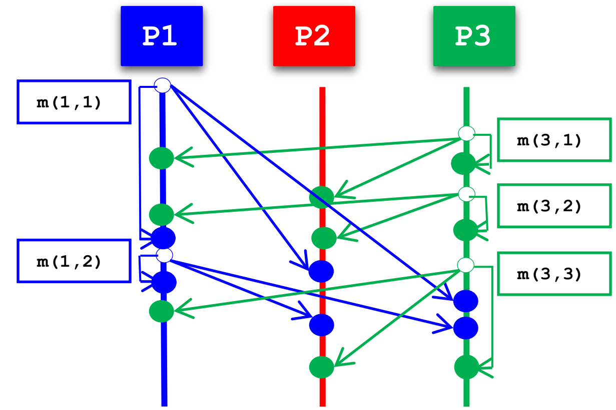
Ex2: Not in Total Order

Types of Ordering

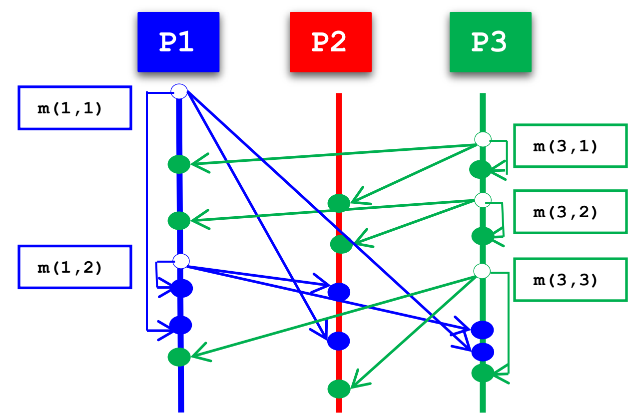
1. Total Ordering
2. Sequential Ordering
3. Causal Ordering

Sequential Ordering

- If a process **Pi** sends a sequence of messages $m_{(i,1)},\ldots,m_{(i,ni)}$, and
- Process **Pj** sends a sequence of messages $m_{(j,1)},\ldots,m_{(j,nj)}$,
- Then, :
  - At any process, the set of messages received are in some 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...



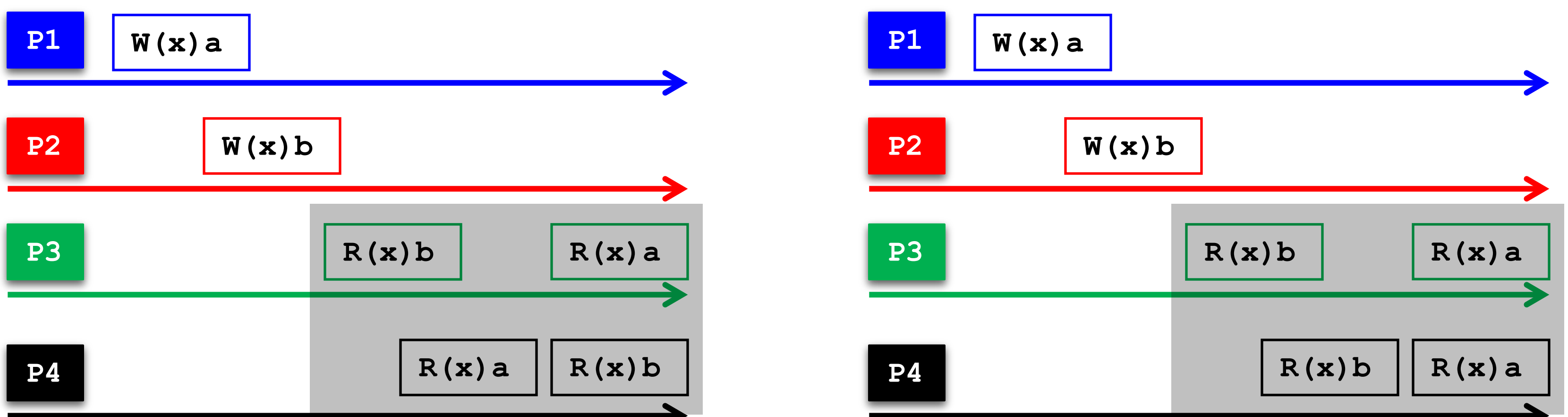
Valid Sequential Orders



Invalid Sequential Orders, but Valid Total Order

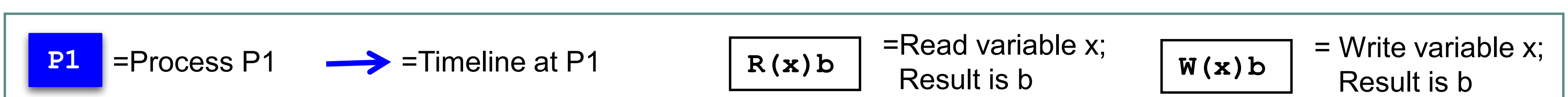
Sequential Consistency Model

- Sequential Consistency Model enforces that all the update operations are executed at the replicas in a sequential 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



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

P1	P2	P3
`x = 1` `print (y,z)`	`y = 1` `print (x,z)`	`z = 1` `print (x,y)`

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

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

Output	001011	101011	000111	010111

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

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 applications
    - We studied Sequential Consistency Model

Next Class

- Consistency Models
  - Causal Consistency Model
  - 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 various implementations of consistency models

References

- [1] Haifeng Yu and Amin Vahdat, “Design and evaluation of a conit-based continuous consistency model for replicated services”
- [2] http://tech.amikelive.com/node-285/using-content-delivery-networks-cdn-to-speed-up-content-load-on-the-web/
- [3] http://en.wikipedia.org/wiki/Replication_(computer_science)
- [4] http://en.wikipedia.org/wiki/Content_delivery_network
- [5] http://www.cdk5.net
- [6] http://www.dis.uniroma1.it/~baldoni/ordered%2520communication%25202008.ppt
- [7] http://www.cs.uiuc.edu/class/fa09/cs425/L5tmp.ppt

Today' session

- Last Session
  - Consistency and Replication
    - Introduction and Data-Centric Consistency Models
- **Today's session**
  - Consistency and Replication – Part II
    - Finish Data-centric Consistency Models
    - Client-Centric Consistency Models

Recap: Trade-offs in Maintaining Consistency

- Maintaining consistency should balance between the strictness of consistency versus efficiency
  - How much consistency is “good-enough” depends on the application

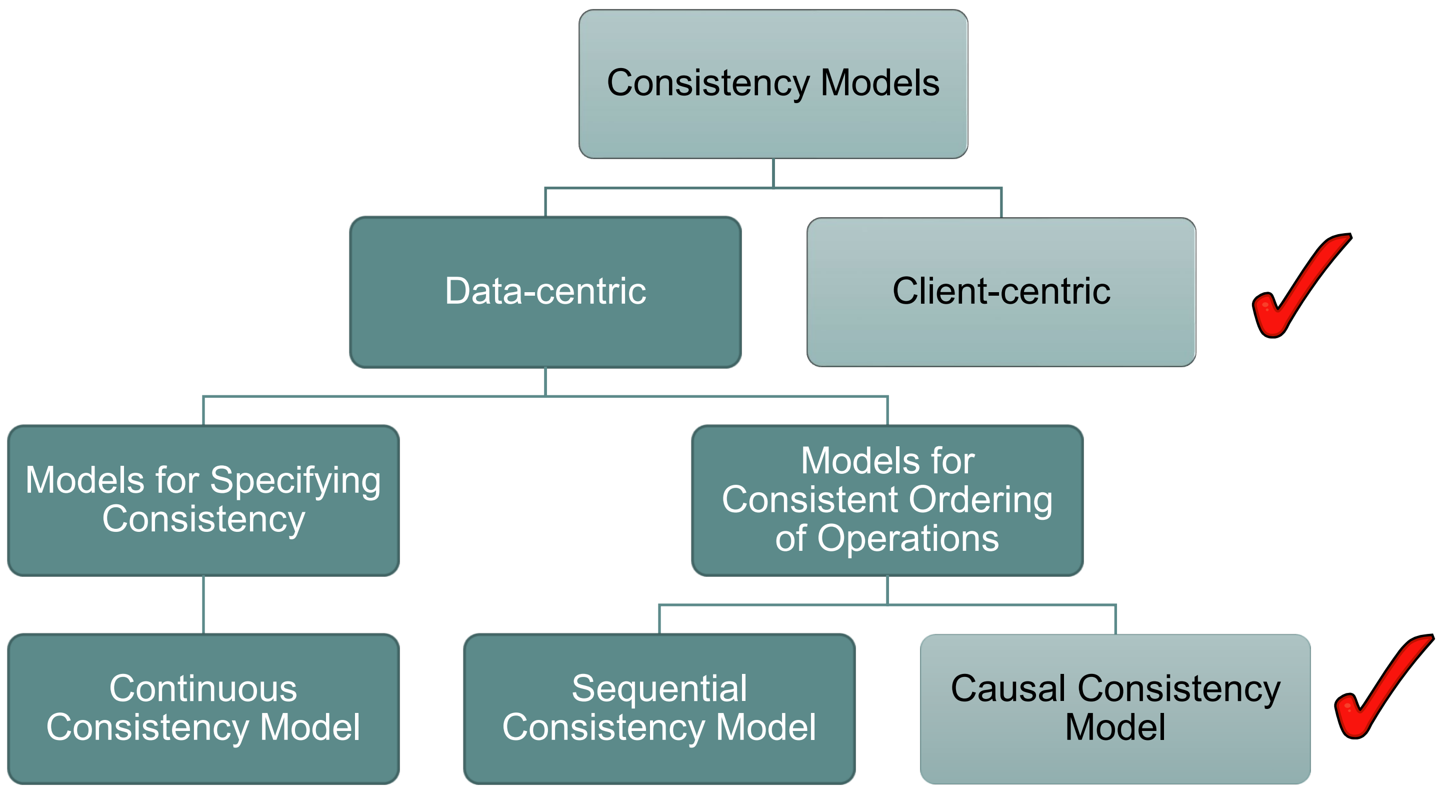
Loose Consistency **Strict Consistency**

Easier to implement,
and is efficient

Generally hard to implement,
and is inefficient

Consistency Models

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



Types of Ordering

1. Total Ordering
2. Sequential Ordering
3. Causal Ordering

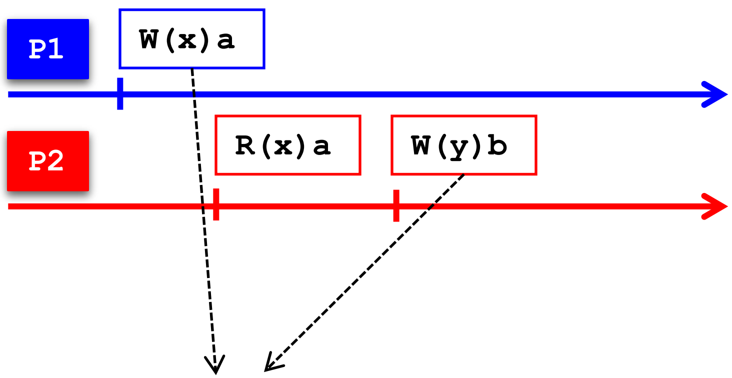
Causality (Recap)

- Causal relation between two events
  - If **a** and **b** are two events such that **a happened-before b** (i.e., **a→b**), and
  - If the (logical) times when events **a** and **b** occur at a process P_i are denoted as $C_i(a)$ and $C_i(b)$
  - Then, if we can infer that **a→b** by observing that $C_i(a) < C_i(b)$, then **a** and **b** are *causally* related

- Causality can be implemented using *Vector Clocks*

Causal vs. Concurrent events

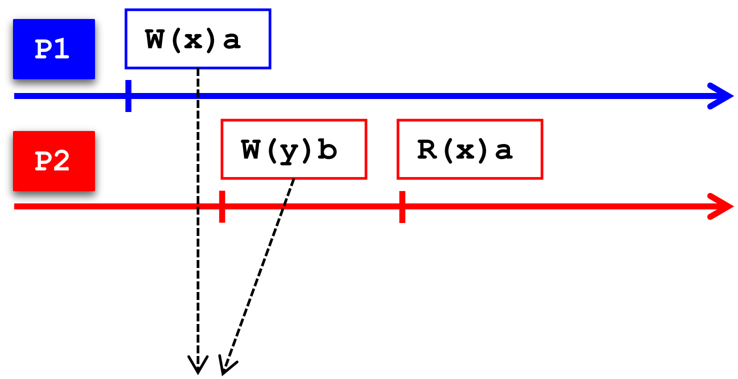
- Consider an interaction between processes $\mathbf{P_1}$ and $\mathbf{P_2}$ operating on replicated data $\mathbf{x}$ and $\mathbf{y}$



Events are causally related

Events are not concurrent

- Computation of $\mathbf{y}$ at $\mathbf{P_2}$ may have depended on value of $\mathbf{x}$ written by $\mathbf{P_1}$



Events are not causally related

Events are concurrent

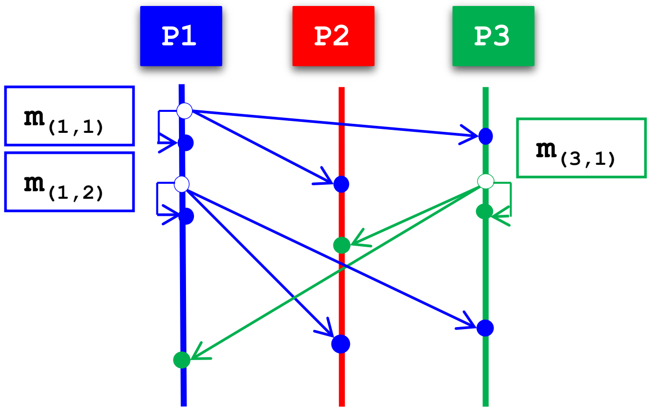
- Computation of $\mathbf{y}$ at $\mathbf{P_2}$ does not depend on value of $\mathbf{x}$ written by $\mathbf{P_1}$

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

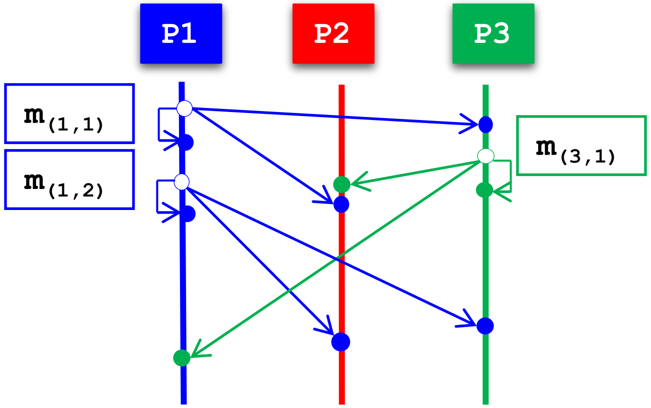
- Causal Order
  - If process P_i sends a message m_i and P_j sends m_j, and if $m_i \rightarrow m_j$ (operator '$\rightarrow$' is Lamport's **happened-before** relation) then any correct process that delivers m_j will deliver m_i before m_j

- In the example, $m_{(1,1)}$ and $m_{(3,1)}$ are in Causal Order



Valid Causal Orders

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

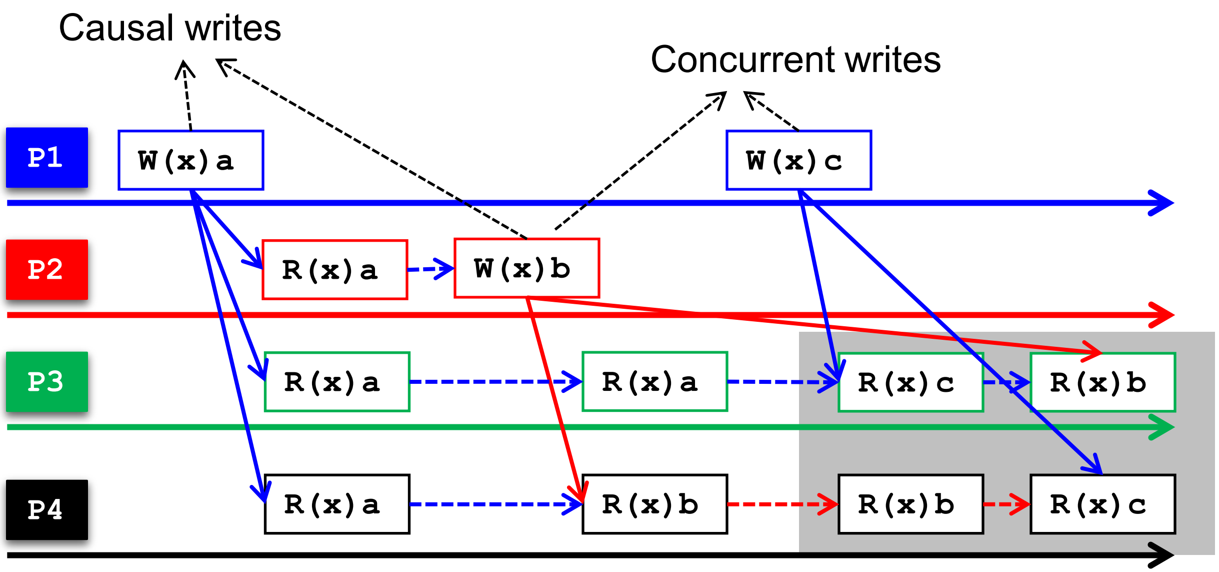


Invalid Causal Order

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

同济大学软件学院
School of Software Engineering, Tongji University

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 provides a way to maintain causally consistent data-base

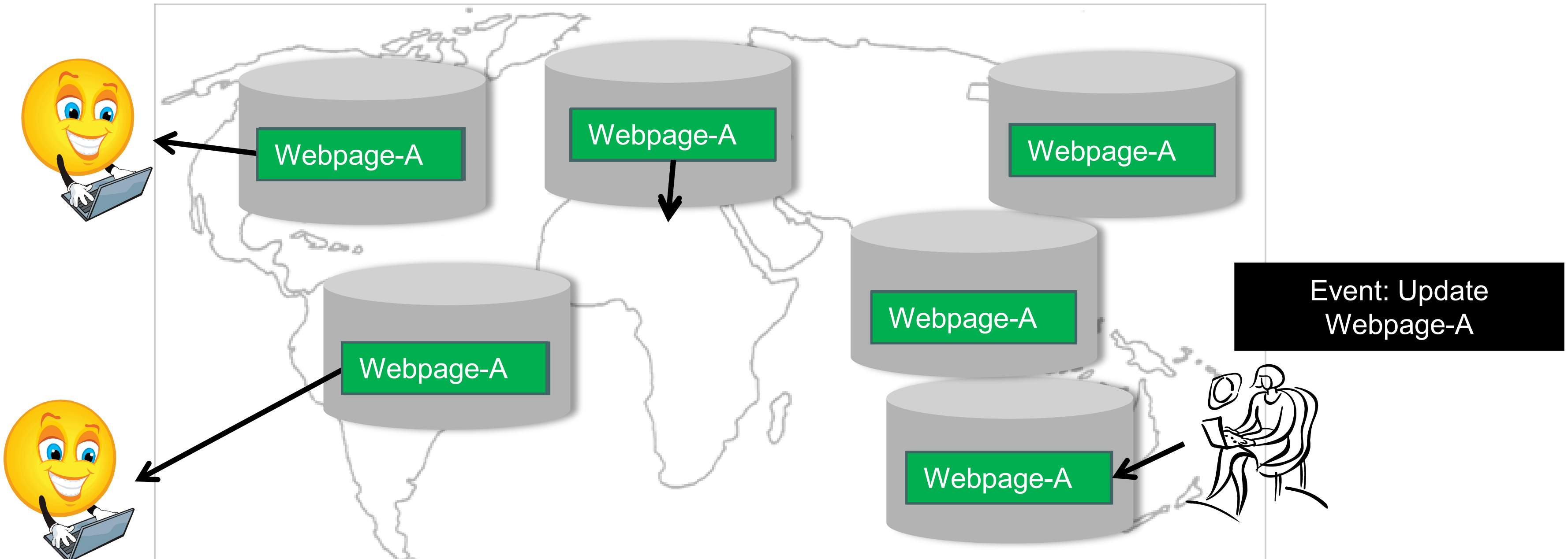
Topics Covered in Data-centric Consistency Models

- Data-centric Consistency Models
  - Models for Specifying Consistency
    - Continuous Consistency Model
  - Models for Consistent Ordering of Operations
    - Sequential Consistency Model
    - Causal Consistency Model

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?



Data-Centric Consistency Model is too strict when

- One client process updates the data
- Other processes read the data, and are OK with reasonably stale data

Summary of Data-Centric Consistency Models

Data-centric consistency models describe how the replicated data is kept consistent across different data-stores, and what a process can expect from the data-store

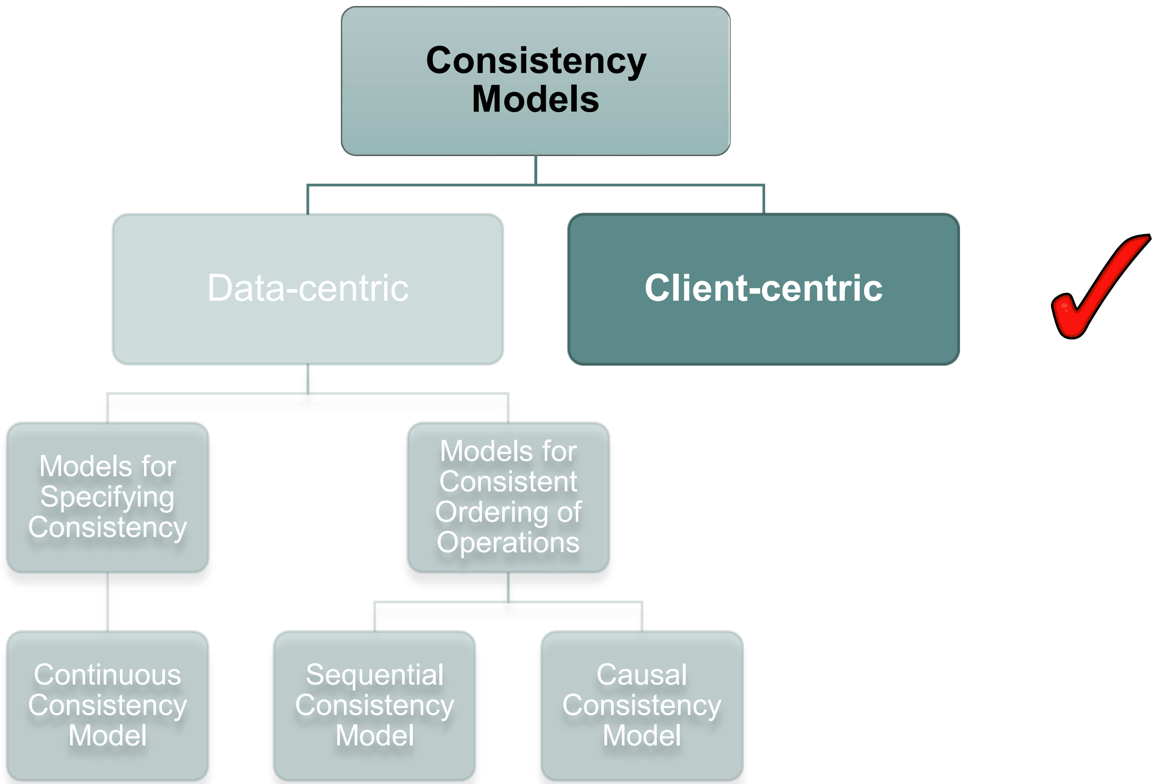
These models allow measuring and specifying the consistency levels that are tolerable to the application

These models specify what ordering of operations are ensured at the replicas

- Data-centric Consistency Models
  - Models for Specifying Consistency
    - Continuous Consistency Model
  - Models for Consistent Ordering of Operations
    - Sequential Consistency Model
    - Causal Consistency Model

Data-centric models are too strict when:
- Most operations are read operations
- Updates are generally triggered from one client process

Overview


Consistency Models
Data-centric
Client-centric
Models for Specifying Consistency
Models for Consistent Ordering of Operations
Continuous Consistency Model
Sequential Consistency Model
Causal Consistency Model

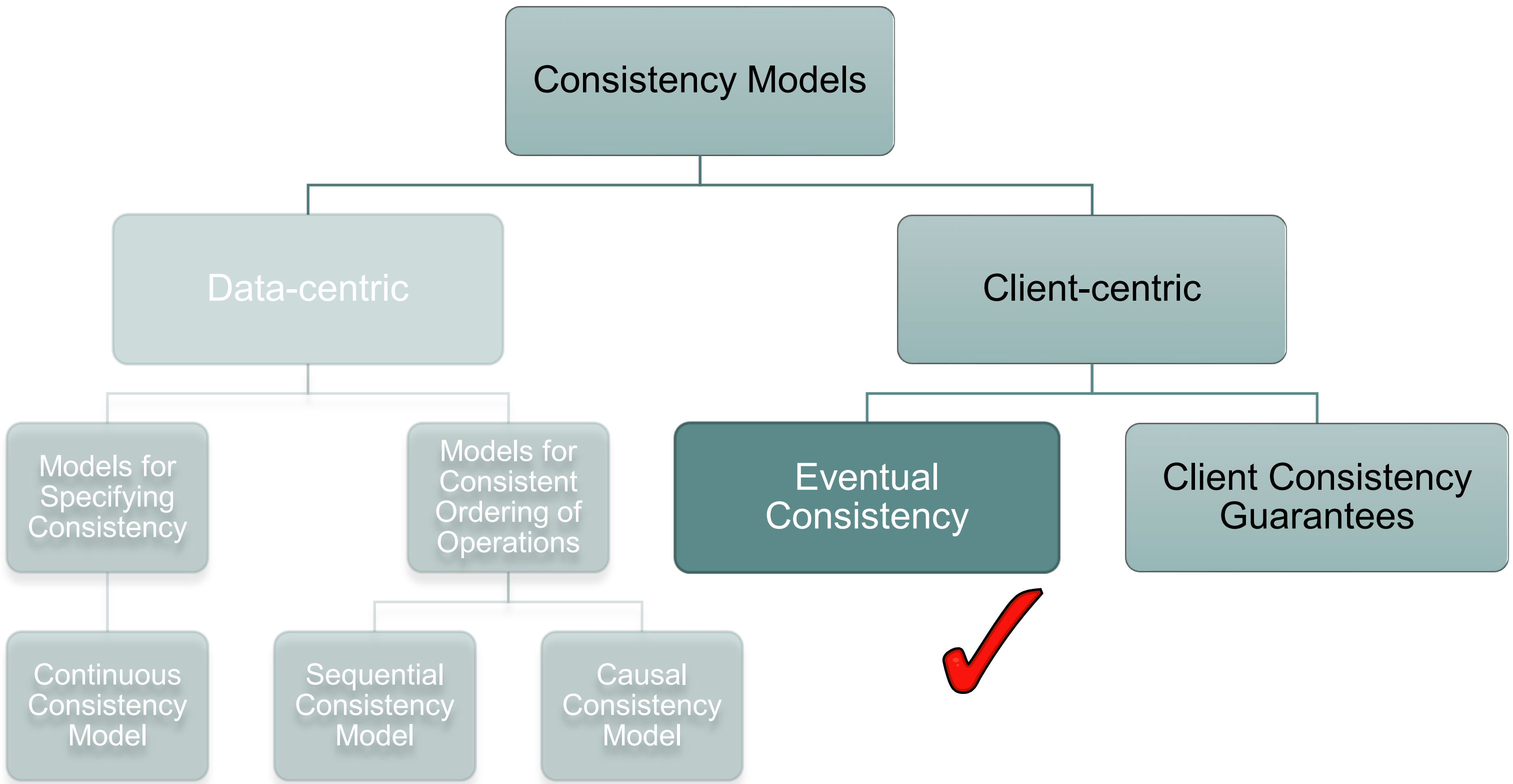
Client-Centric Consistency Models

- Data-centric models lead to excessive overheads in applications where:
  - a majority operations are reads, and
  - updates occur frequently, and are often from one client process

- For such applications, a weaker form of consistency called *Client-centric Consistency* is employed for improving efficiency

- Client-centric consistency models specify two requirements:
  1. Client Consistency Guarantees
     - A client events should be guaranteed some level of consistency while accessing the data value at different replicas
  2. Eventual Consistency
     - All the replicas should *eventually* converge on a final value

Overview


Consistency Models
Data-centric
Client-centric
Models for Specifying Consistency
Models for Consistent Ordering of Operations
Eventual Consistency
Client Consistency Guarantees
Continuous Consistency Model
Sequential Consistency Model
Causal Consistency Model

Eventual Consistency

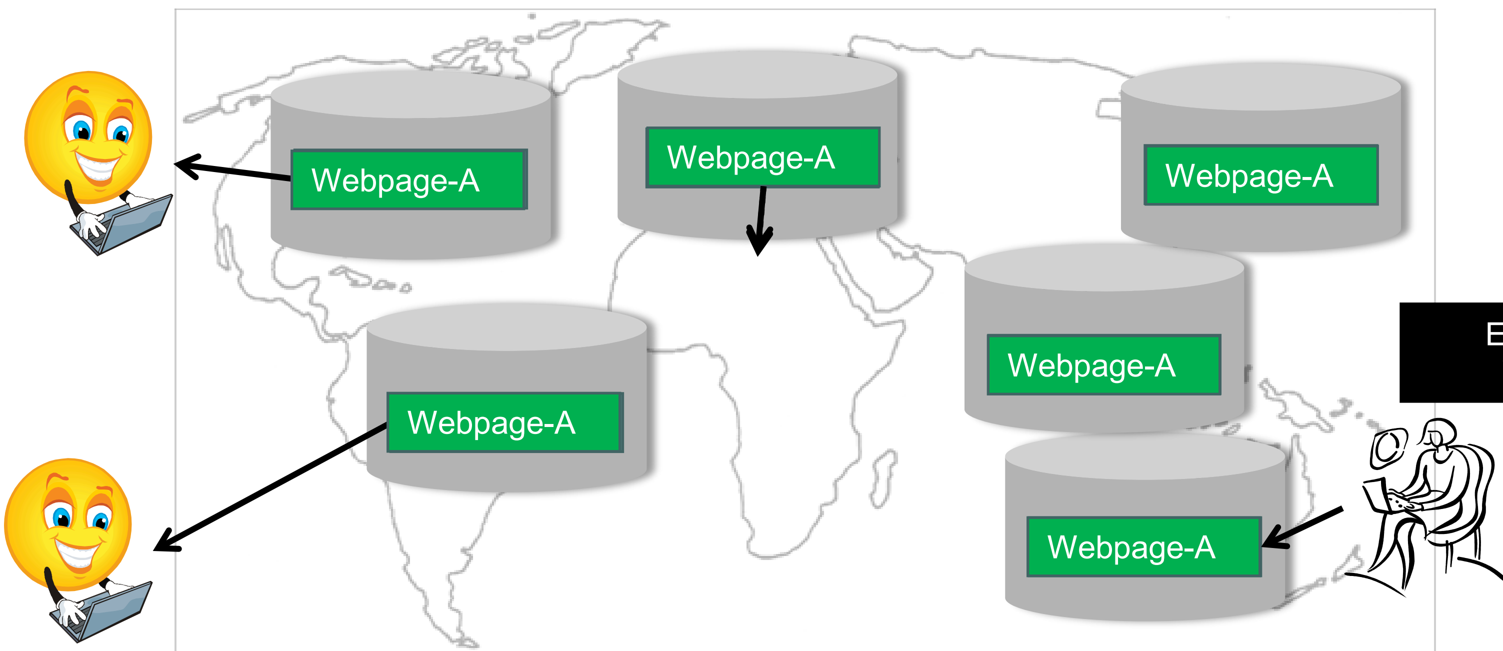
- Many applications can tolerate inconsistency for a long time
  - Webpage updates, Web Search – Crawling, indexing and ranking, Updates to DNS Server
- In such applications, it is acceptable and efficient if replicas in the data-store rarely exchange updates
- A data-store is termed as *Eventually Consistent* if:
  - All replicas will gradually become consistent in the absence of updates
- Typically, updates are propagated infrequently in eventually consistent data-stores

Designing Eventual Consistency

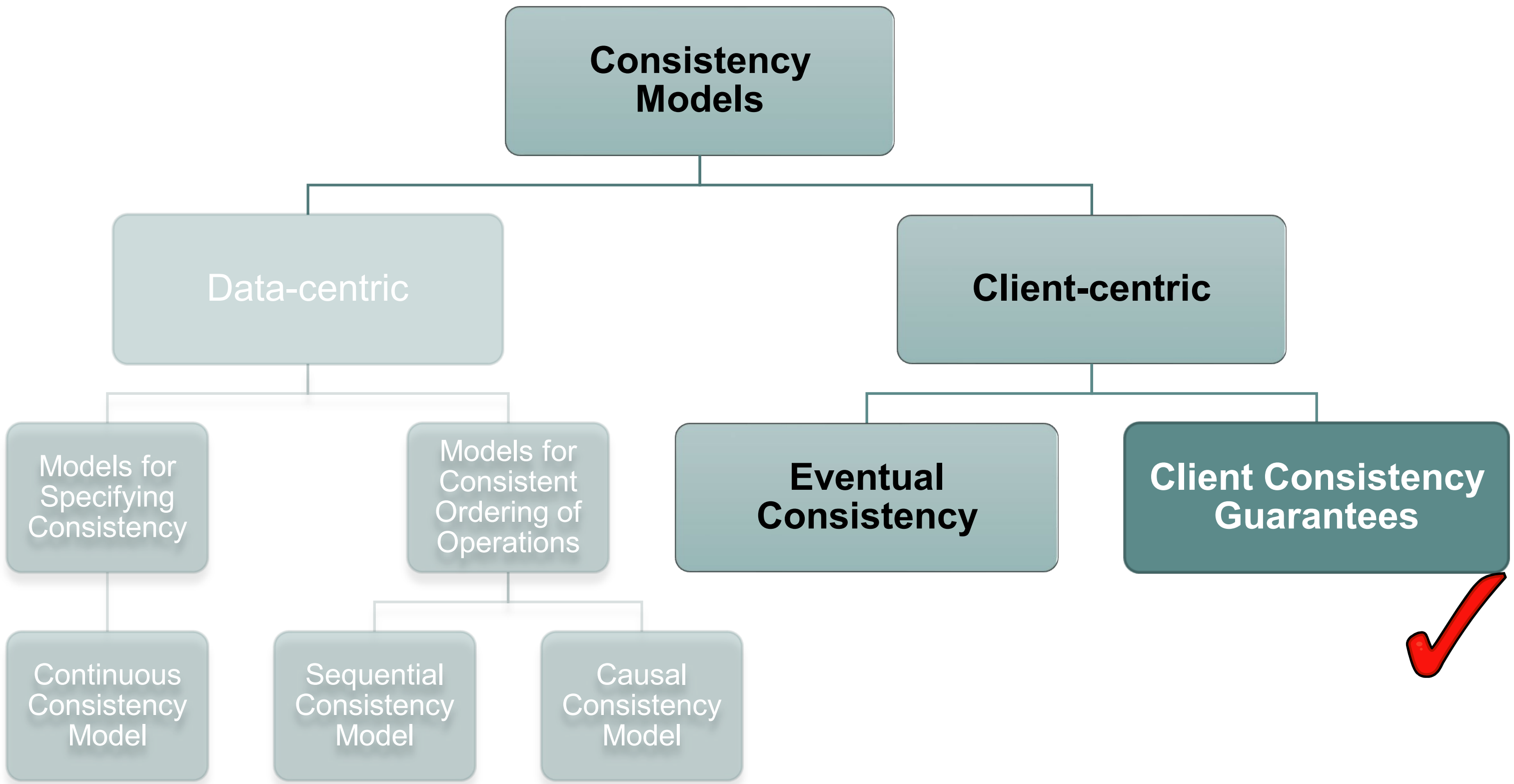
- In eventually consistent data-stores,
  - *Write-write conflicts* are rare
    - Two processes that write the same value are rare
    - Generally, one client updates the data value
      - e.g., One DNS server updates the name to IP mapping
    - Such rare conflicts can be handled through simple mechanisms, such as mutual exclusion
  - *Read-write conflicts* are more frequent
    - Conflicts where one process is reading a value, while another process is writing a value to the same variable
    - Eventual Consistency Design has to focus on efficiently resolving such conflicts

Challenges in Eventual Consistency

- Eventual Consistency is not good-enough when the client process accesses data from different replicas
  - We need consistency guarantees for a single client while accessing the data-store



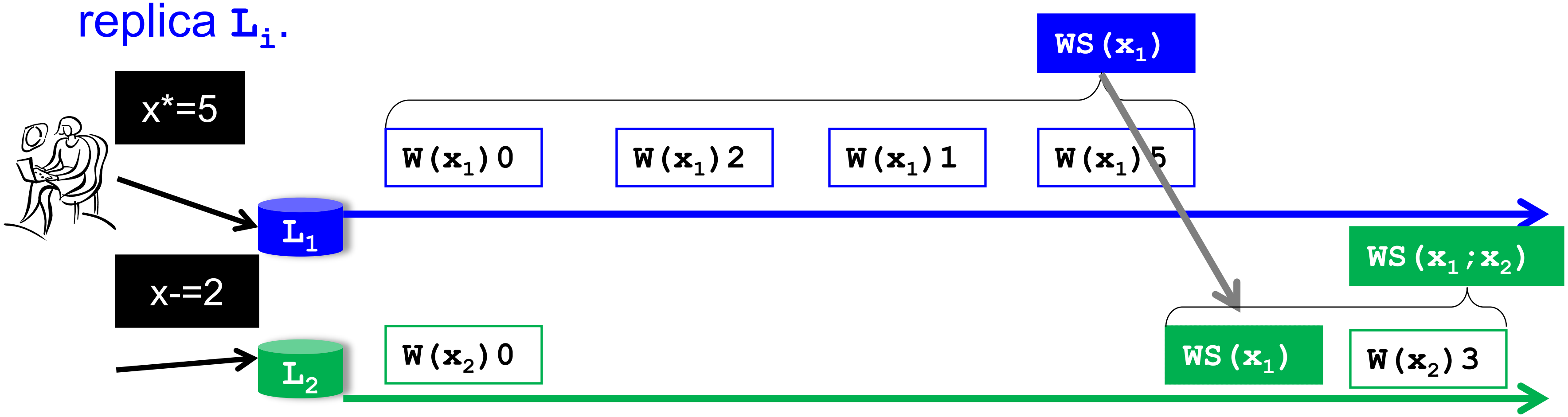
Overview


Consistency Models
Data-centric
Client-centric
Models for Specifying Consistency
Models for Consistent Ordering of Operations
Eventual Consistency
Client Consistency Guarantees
Continuous Consistency Model
Sequential Consistency Model
Causal Consistency Model

Client Consistency Guarantees

- Client-centric consistency provides guarantees for a single client for its accesses to a data-store

- Example: Providing consistency guarantee to a client process for data **x** replicated on two replicas. Let $\mathbf{x_i}$ be the local copy of a data **x** at replica $\mathbf{L_i}$.



WS(x_1) = **Write Set for x_1** = Series of ops being done at some replica that reflects how L_1 updated x_1 till this time

WS(x_1;x_2) = **Write Set for x_1 and x_2** = Series of ops being done at some replica that reflects how L_1 updated x_1 and, later on, how x_2 is updated on L_2

L_i = Replica i	R(x_i)b = Read variable x at replica i; Result is b	W(x)b = Write variable x at replica i; Result is b
WS(x_i) = Write Set		

Client Consistency Guarantees

- We will study four types of client-centric consistency models[1]
  1. Monotonic Reads
  2. Monotonic Writes
  3. Read Your Writes
  4. Write Follow Reads

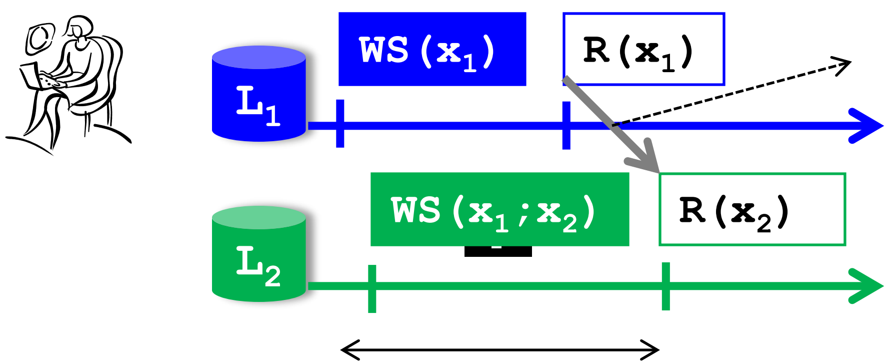
1. The work is based on the distributed database system built by Terry et al. [1]

Overview


Consistency Models
Data-centric
Client-centric
Eventual Consistency
Client Consistency Guarantees
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

Monotonic Reads

- The model provides guarantees on successive reads
- If a client process reads the value of data item **x**, then any successive read operation by that process should return the same or a more recent value for **x**



Order in which client process carries out the operations

Result of **R(x_2)** should at least be as recent as **R(x_1)**

Monotonic Reads - Puzzle

Recognize data-stores that provide monotonic read guarantees

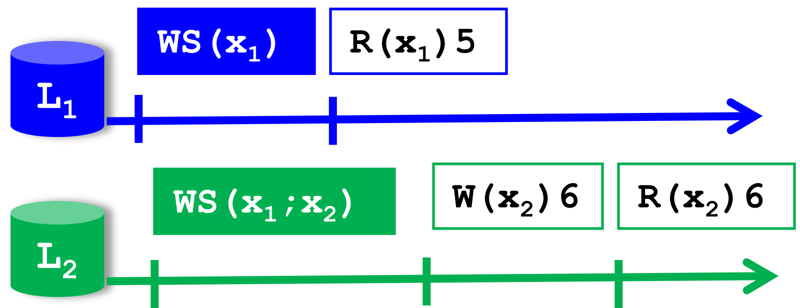


图 1

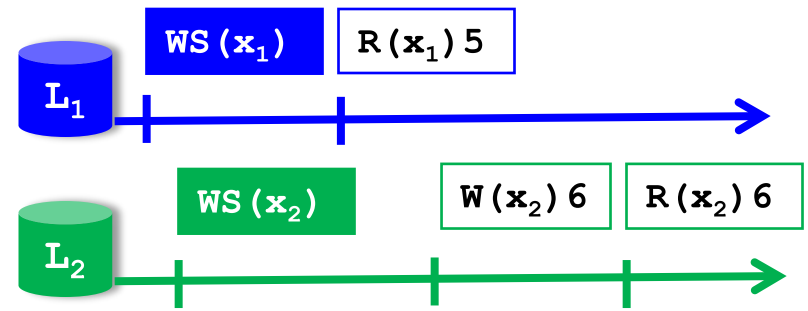


图 2

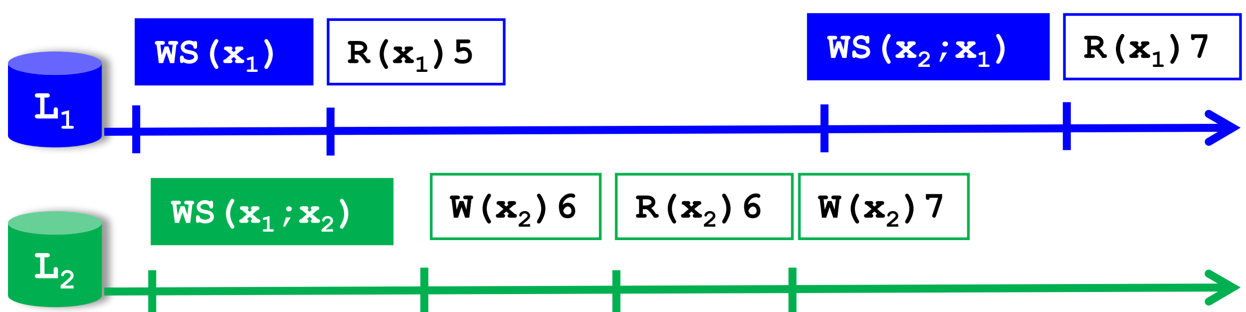


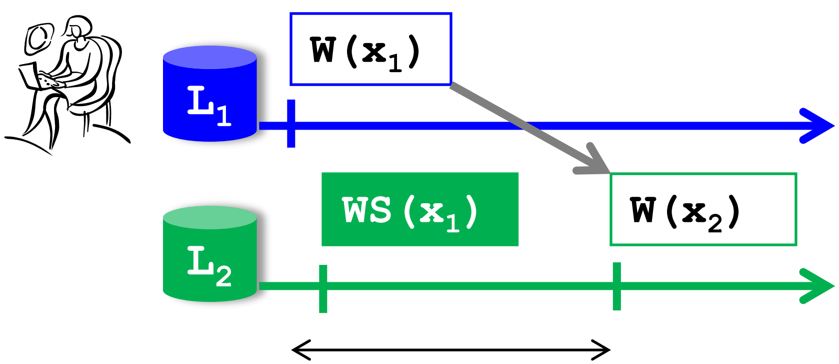
图 3

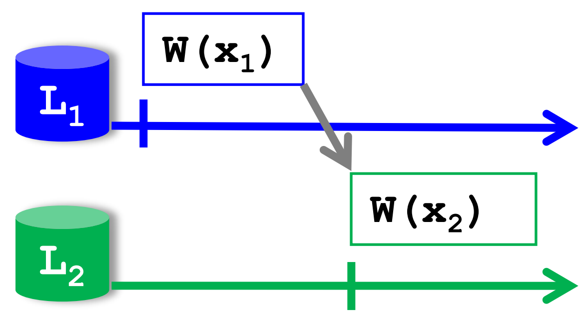
Overview

Consistency Models
Data-centric
Client-centric
Eventual Consistency
Client Consistency Guarantees
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

Monotonic Writes

- This consistency model assures that writes are monotonic
- A write operation by a client process on a data item **x** is completed <u>before any successive write</u> operation on **x** by the <u>same process</u>
  - A new write on a replica should wait for all old writes on any replica





$W(x_2)$ operation should be performed only after the result of $W(x_1)$ has been updated at L_2

The data-store does not provide monotonic write consistency

Monotonic Writes – An Example

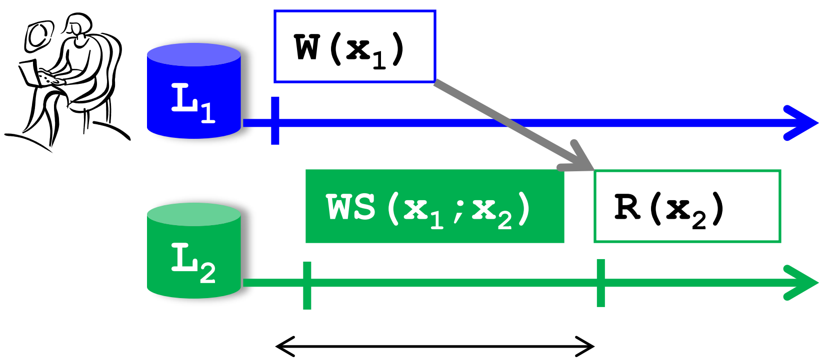
- Example: Updating individual libraries in a large software source code which is replicated
  - Updates can be propagated in a lazy fashion
  - Updates are performed on a part of the data item
    - Some functions in an individual library is often modified and updated
  - Monotonic writes: If an update is performed on a library, then all preceding updates on the same library are first updated
- Question: If the update overwrites the complete software source code, is it necessary to update all the previous updates?

Overview

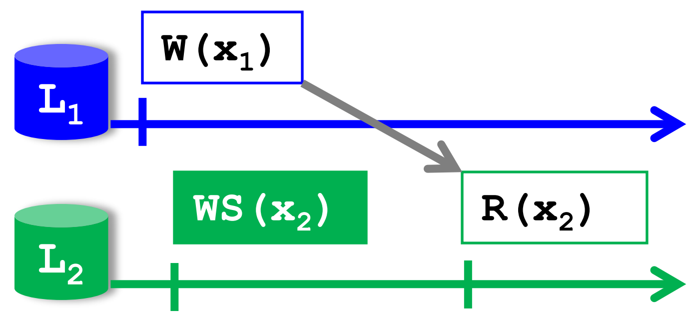
Consistency Models
Data-centric
Client-centric
Eventual Consistency
Client Consistency Guarantees
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

Read Your Writes

- The effect of a write operation on a data item **x** by a process will always be seen by a successive read operation on **x** by the same process
- Example scenario:
  - In systems where password is stored in a replicated data-base, the password change should be seen immediately

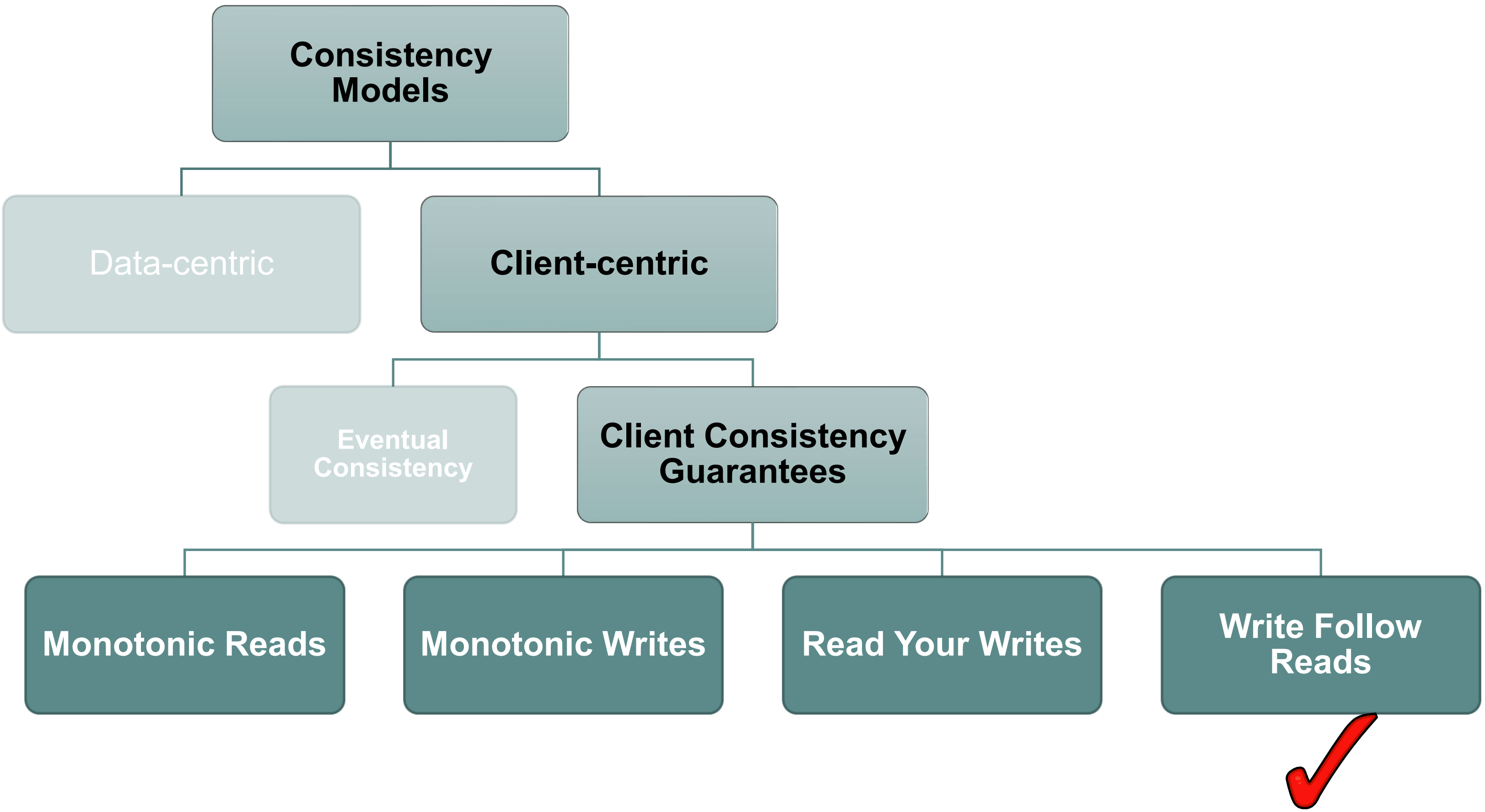


R(x$_2$) operation should be performed only after the updating of the Write Set **WS(x$_1$)** at **L$_2$**



A data-store that does not provide *Read Your Write* consistency

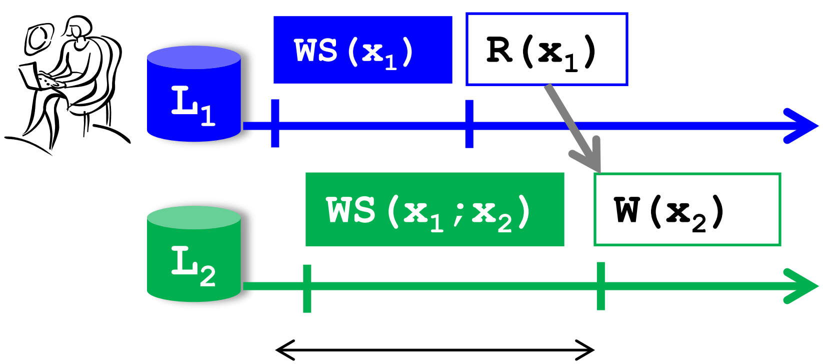
Overview

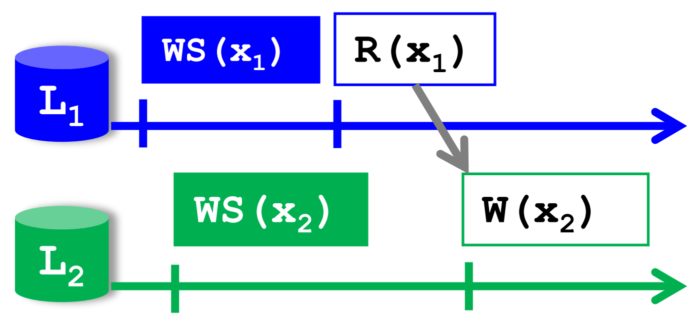


Consistency Models
Data-centric
Client-centric
Eventual Consistency
Client Consistency Guarantees
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

Write 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
- Example scenario:
  - Users of a newsgroup should post their comments only after they have read all previous comments





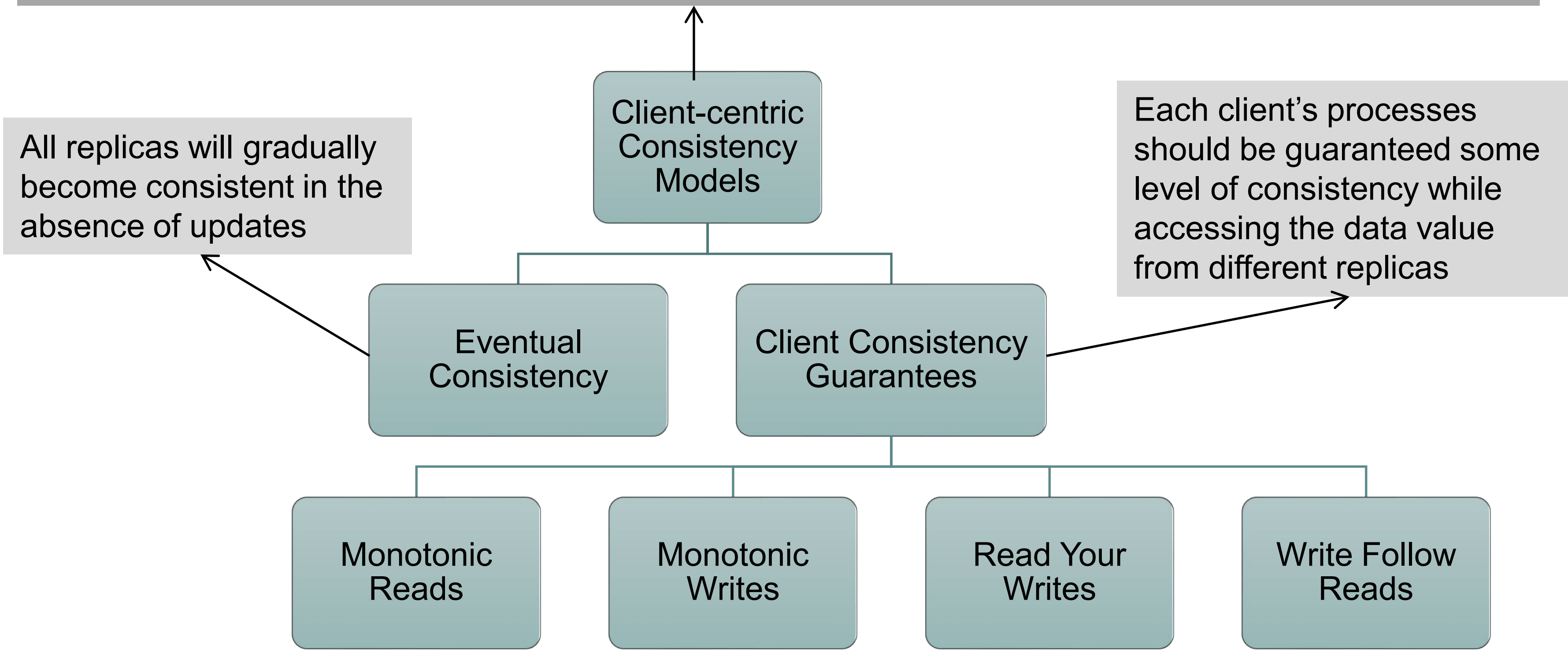
$W(x_2)$ operation should be performed only after all previous writes have been seen

A data-store that does not guarantee Write Follow Read Consistency Model

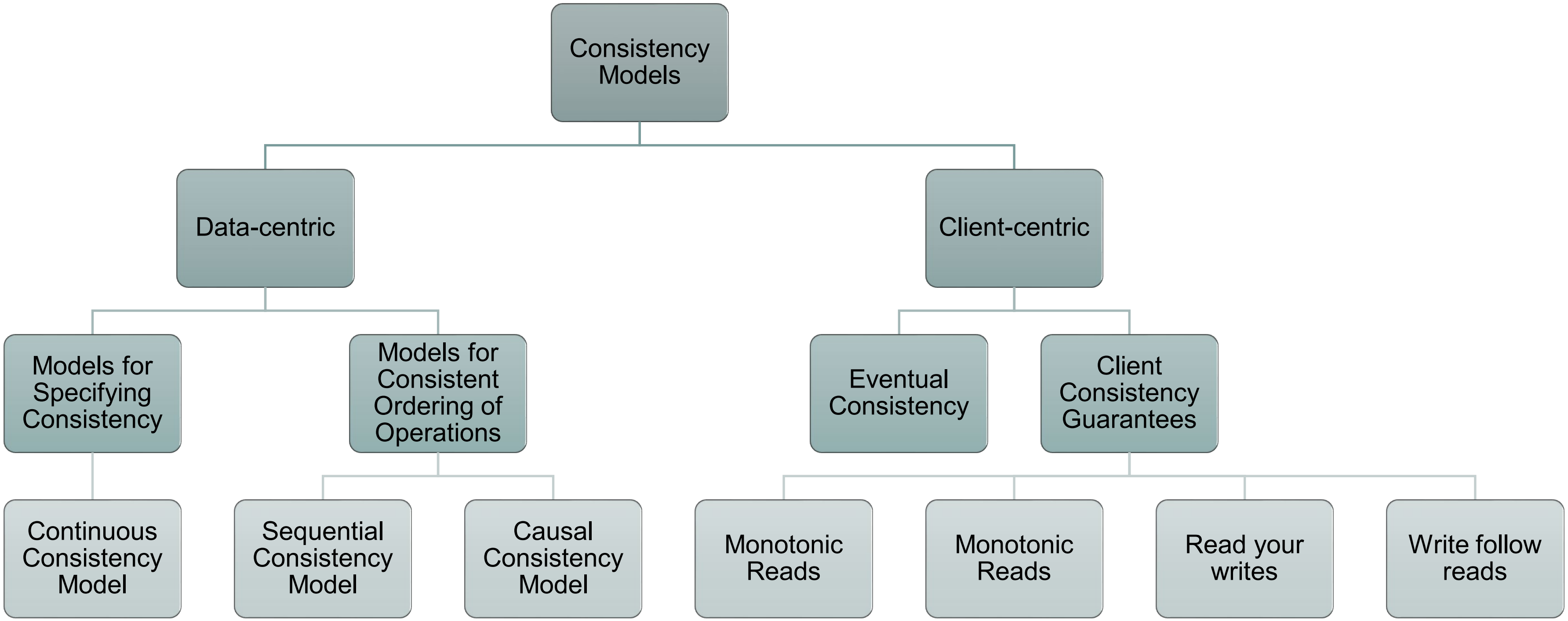
Summary of Client-centric Consistency Models

Client-centric Consistency Model defines how a data-store presents the data value to an individual client when the client process accesses the data value across different replicas.
It is generally useful in applications where:

- one client always updates the data-store.
- read-to-write ratio is high.



Topics Covered in Consistency Models



Summary of Consistency Models

- Different applications require different levels of consistency
  - Data-centric consistency models
    - Define how replicas in a data-store maintain consistency
  - Client-centric consistency models
    - Provide an efficient, but *weaker* form of consistency
    - One client process updates the data item, and many processes read the replica

Next Class

- Replica Management
  - Describes where, when and by whom replicas should be placed
- Consistency Protocols
  - We study “how” consistency is ensured in distributed systems

References

- [1] Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M.J., Theimer, M.M., Welch, B.B., "Session guarantees for weakly consistent replicated data", Proceedings of the Third International Conference on Parallel and Distributed Information Systems, 1994
- [2] Lili Qiu, Padmanabhan, V.N., Voelker, G.M., “On the placement of Web server replicas”, Proceedings of IEEE INFOCOM 2001.
- [3] Rabinovich, M., Rabinovich, I., Rajaraman, R., Aggarwal, A., “A dynamic object replication and migration protocol for an Internet hosting service”, Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS), 1999
- [4] http://www.cdk5.net