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

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

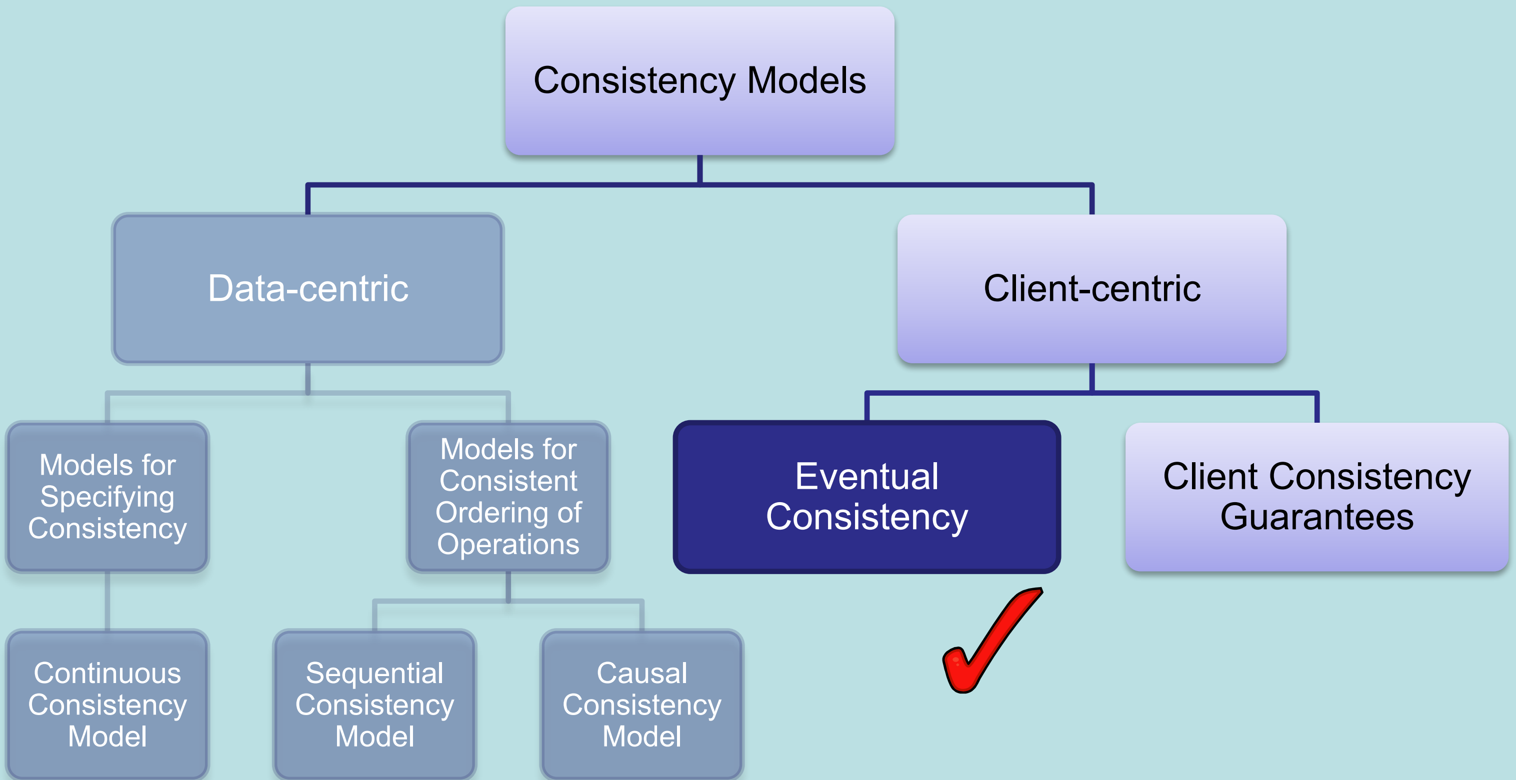
- For such applications, a weaker form of consistency called *Client-centric Consistency* is employed for improving efficiency

- Client-centric consistency models specify two requirements:

1. Eventual Consistency
   - All the replicas should *eventually* converge on a final value

2. Client Consistency Guarantees
   - Each client processes should be guaranteed some level of consistency while accessing the data value from different replicas

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 a 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

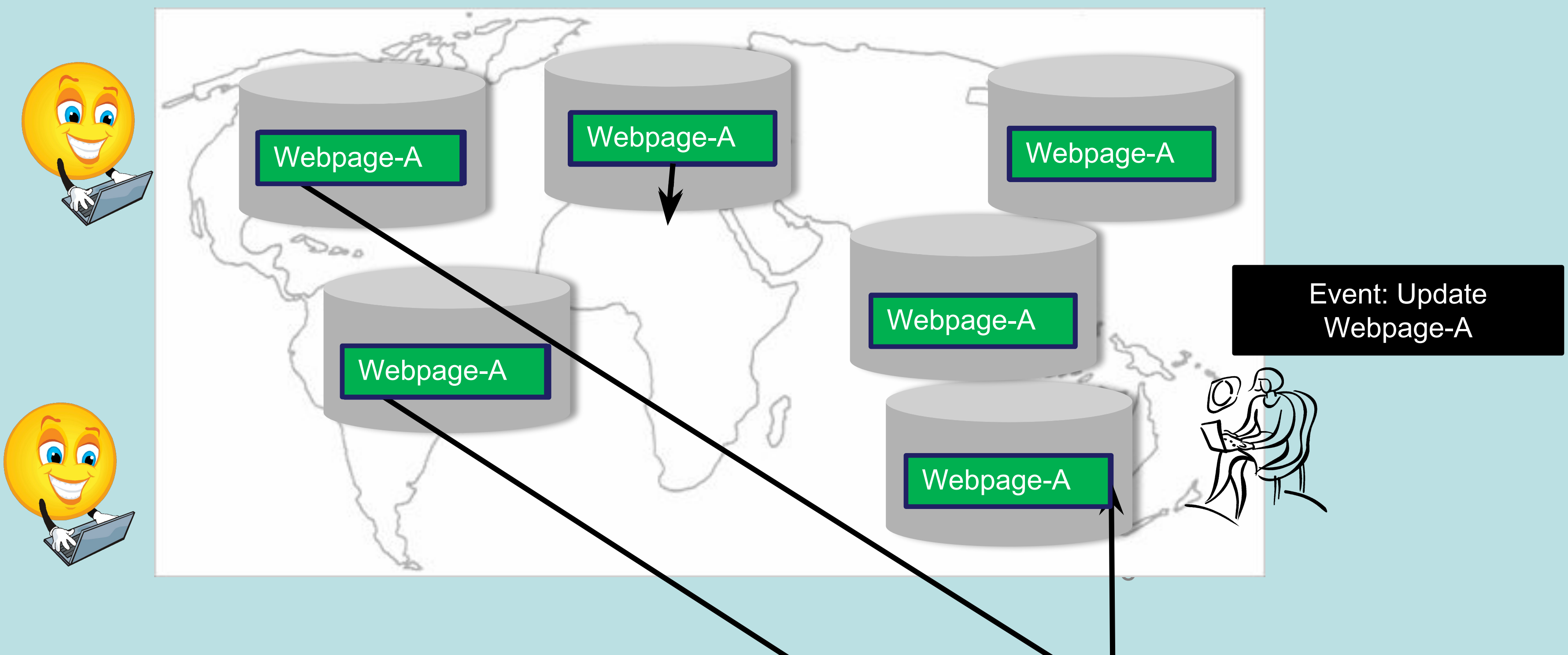
Designing Eventual Consistency

- 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

  - R*ead-write conflict* 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

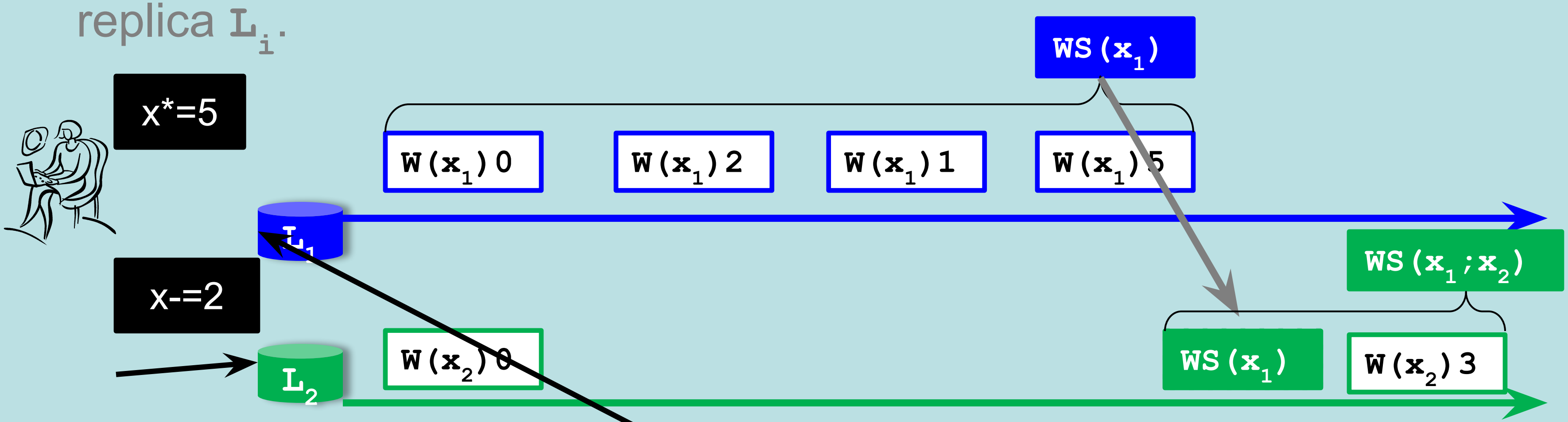
- Eventual Consistency is not good-enough when the client process accesses data from different replicas
  - We need consistency guarantees for a single client while accessing the data-store



Client Consistency Models

- 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** $\mathbf{x}_1$ = Series of ops being done at some replica that reflects how $\mathbf{L}_1$ updated $\mathbf{x}_1$ till this time

WS(x_1;x_2) = **Write Set for** $\mathbf{x}_1$ **and** $\mathbf{x}_2$ = Series of ops being done at some replica that reflects how $\mathbf{L}_1$ updated $\mathbf{x}_1$ and, later on, how $\mathbf{x}_2$ is updated on $\mathbf{L}_2$

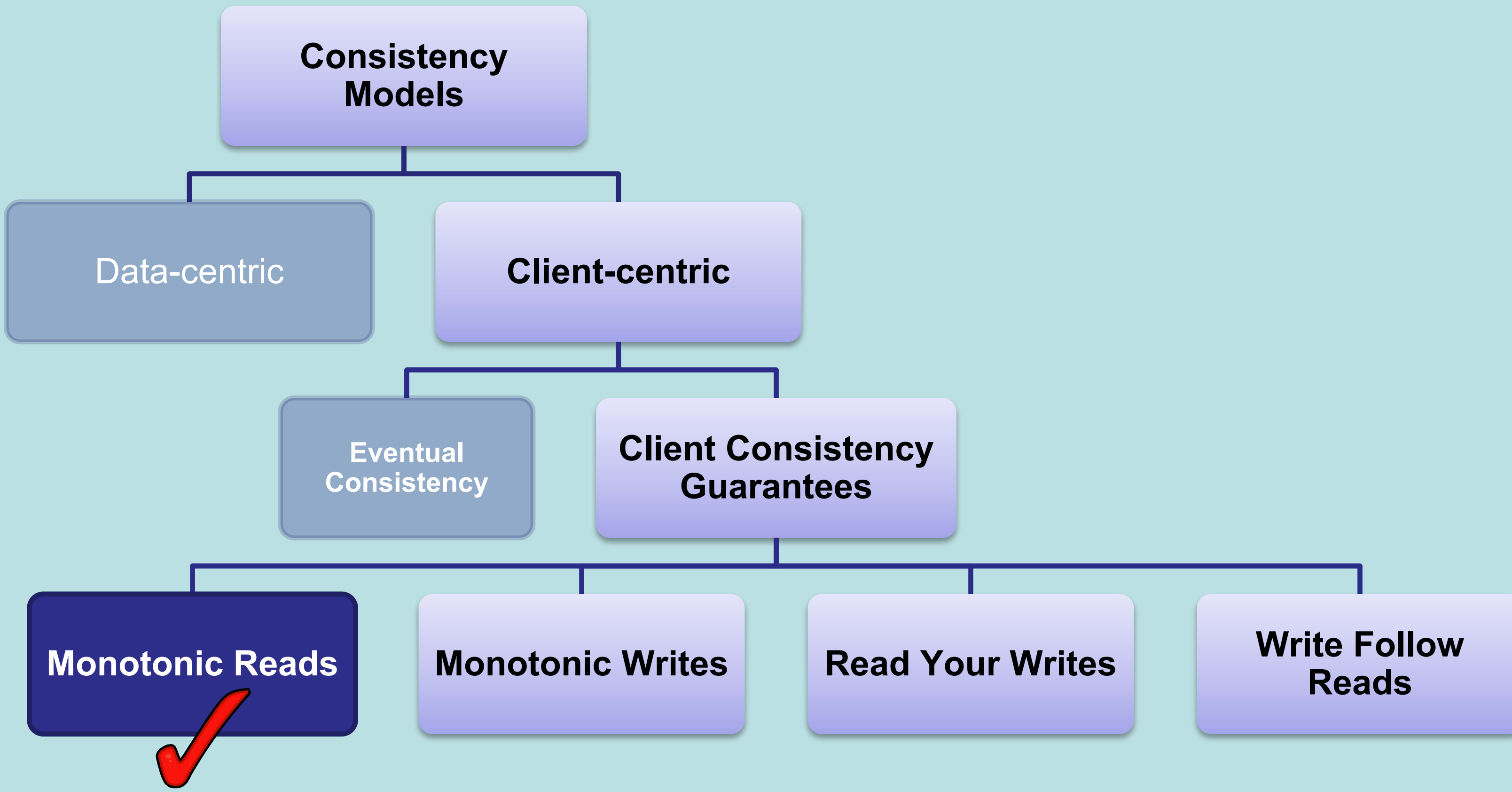
L_i	= Replica i	R(x_i)b	= Read variable x at replica i; Result is b	W(x)b	= Write variable x at replica i; Result is b	WS(x_i)	= Write Set

Client Consistency Models

- 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

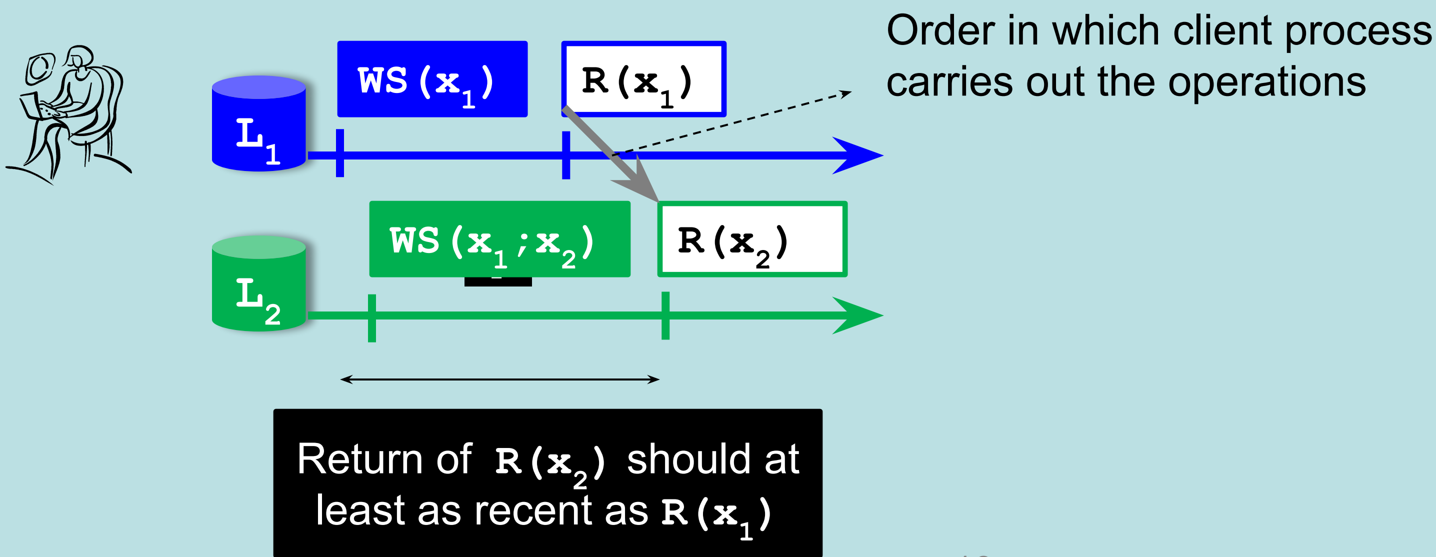
1. The work is based on the distributed database system built by Terry et al. [1]

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 <u>same</u> or a <u>more recent value</u> for **x**



Monotonic Reads – Puzzle

Recognize data-stores that provide monotonic read guarantees

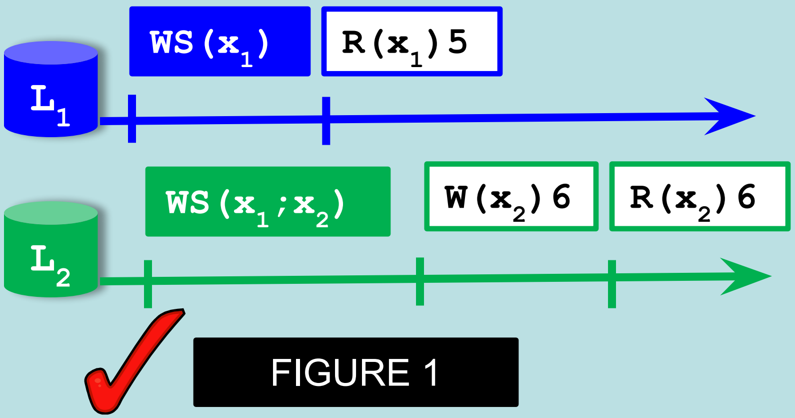


FIGURE 1

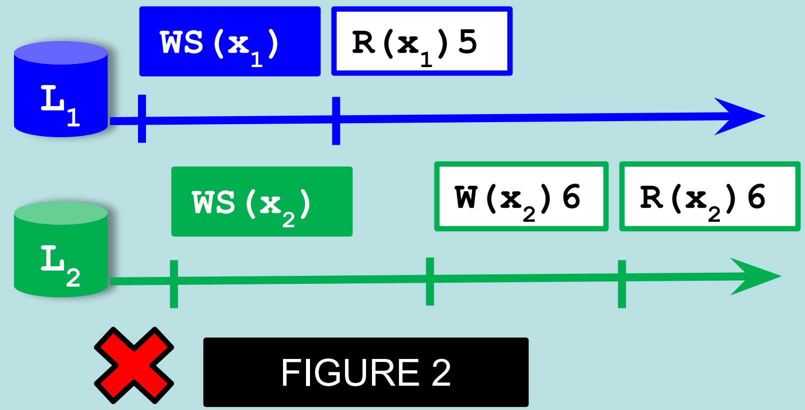


FIGURE 2

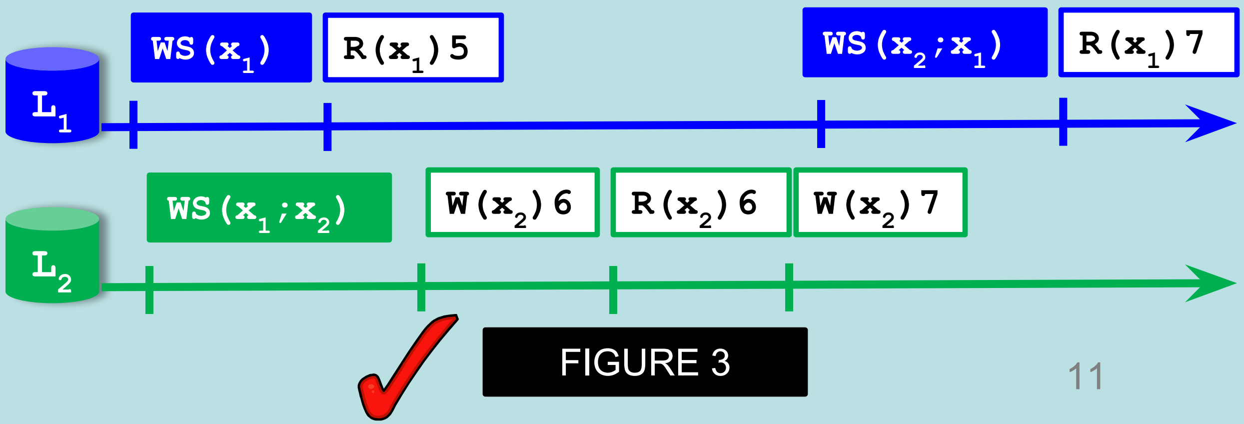
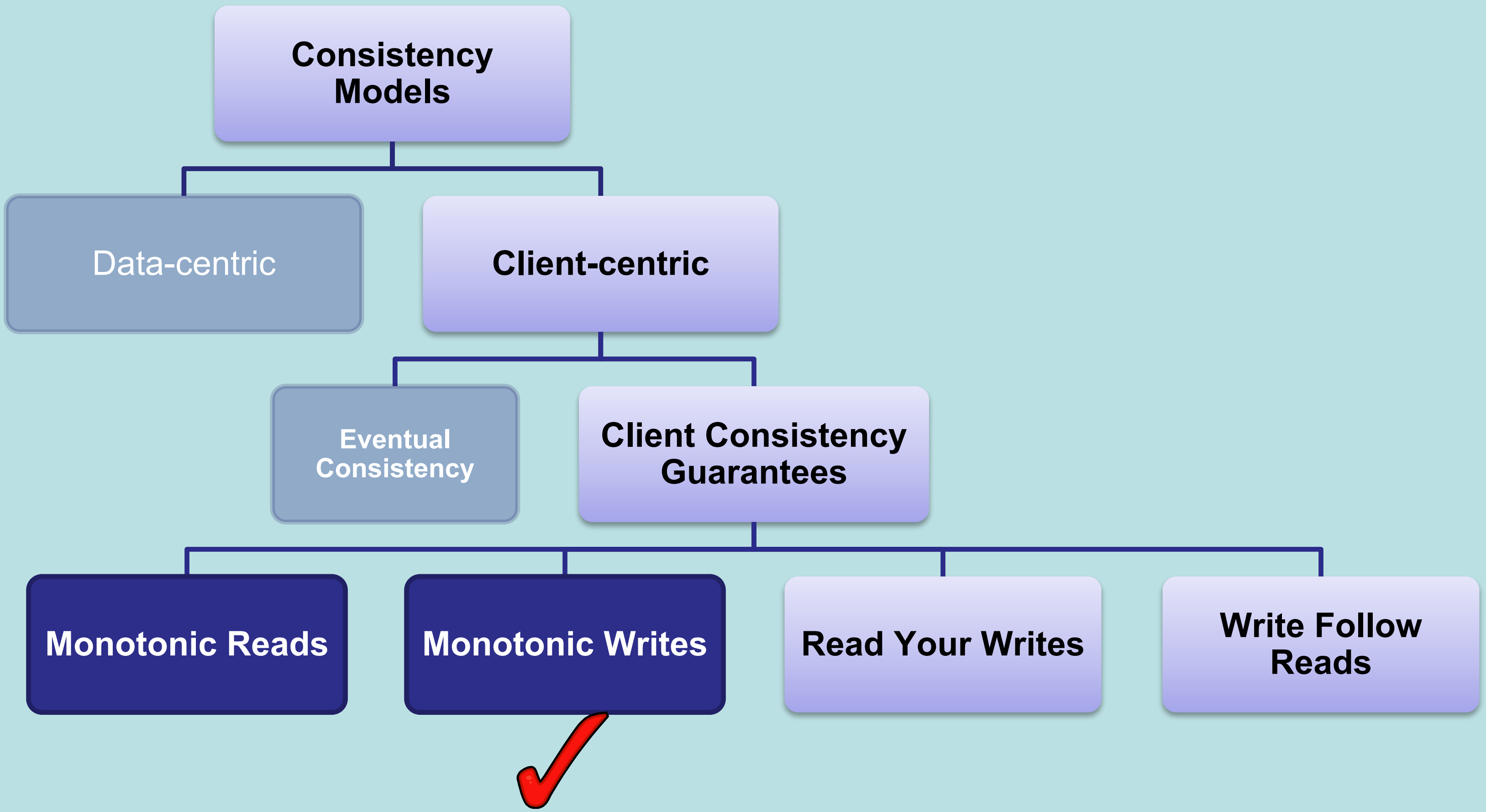


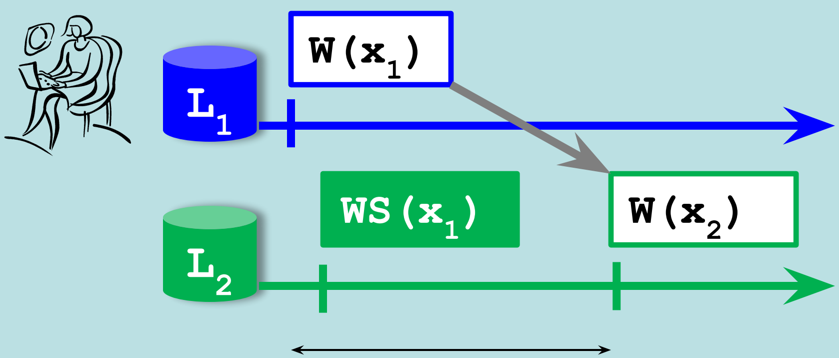
FIGURE 3

Overview

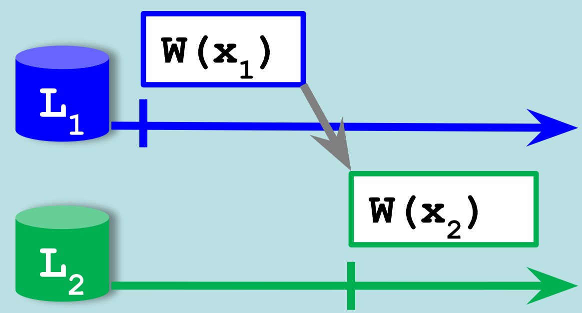

Consistency Models
Data-centric
Client-centric
Eventual Consistency
Client Consistency Guarantees
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

Monotonic Writes

- This consistency model assures that writes are monotonic
- A write operation by a client process on a data item **x** is completed before any successive write operation on **x** by the same process
  - 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



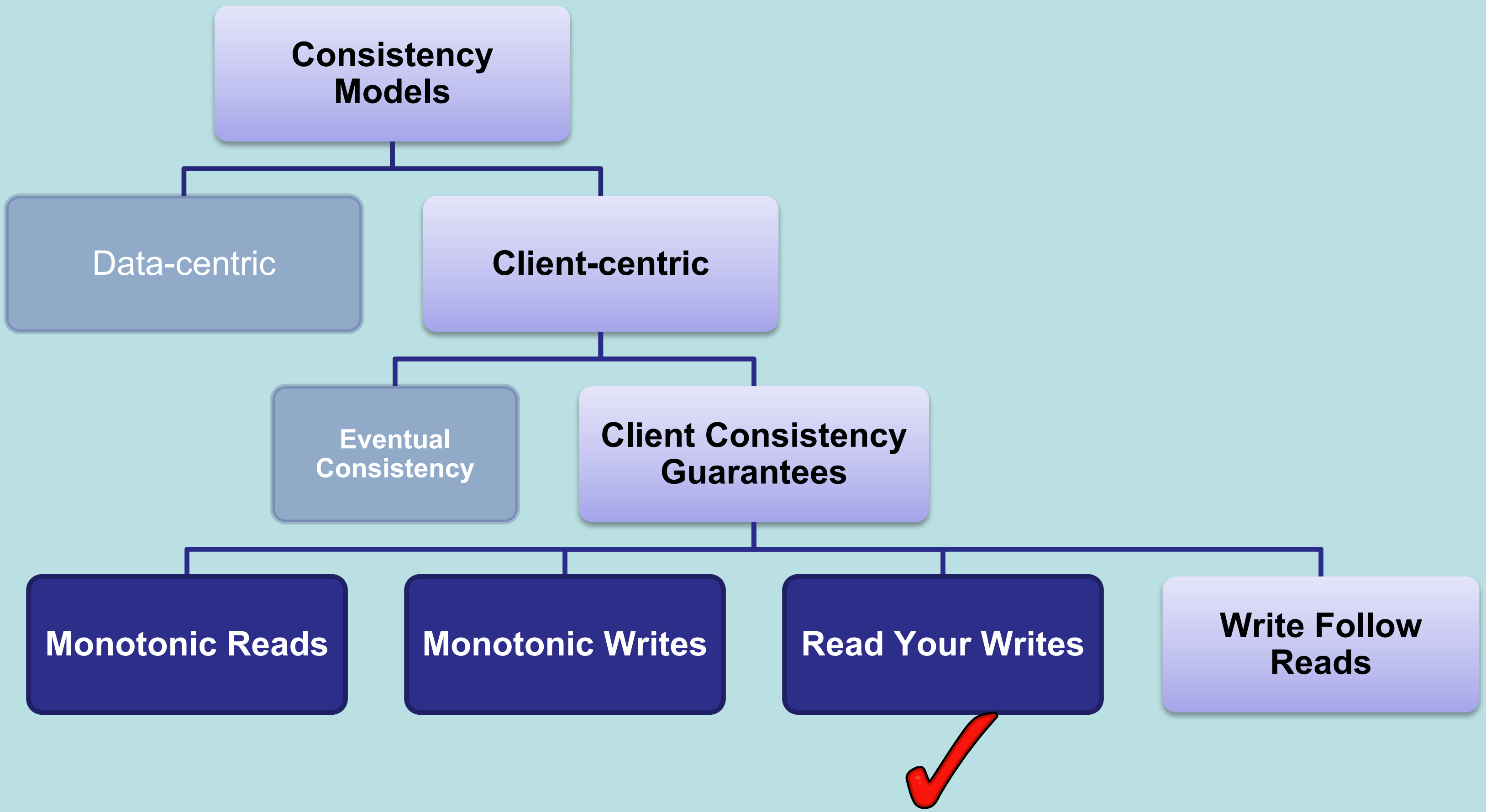
The data-store does not provide monotonic write consistency

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

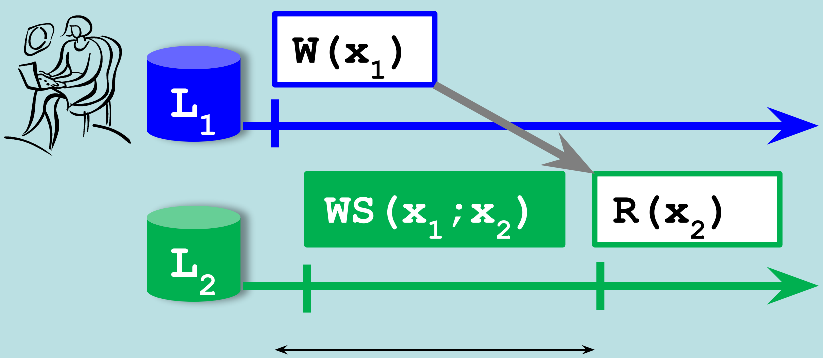
- Question: If the update overwrites the complete software source code, is it necessary to update all the previous updates?

Overview

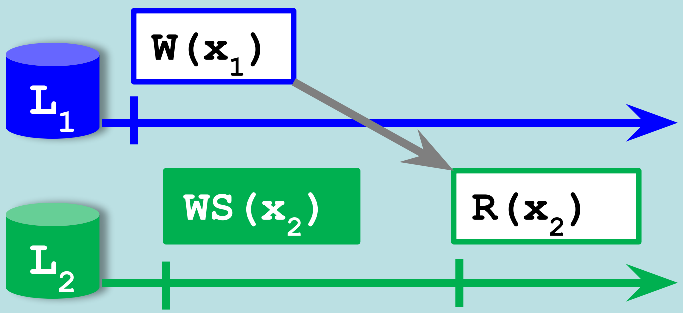

Consistency Models
Data-centric
Client-centric
Eventual Consistency
Client Consistency Guarantees
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

Read Your Writes

- The <u>effect of a write</u> operation on a data item **x** by a process will <u>always be seen by a successive read</u> 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

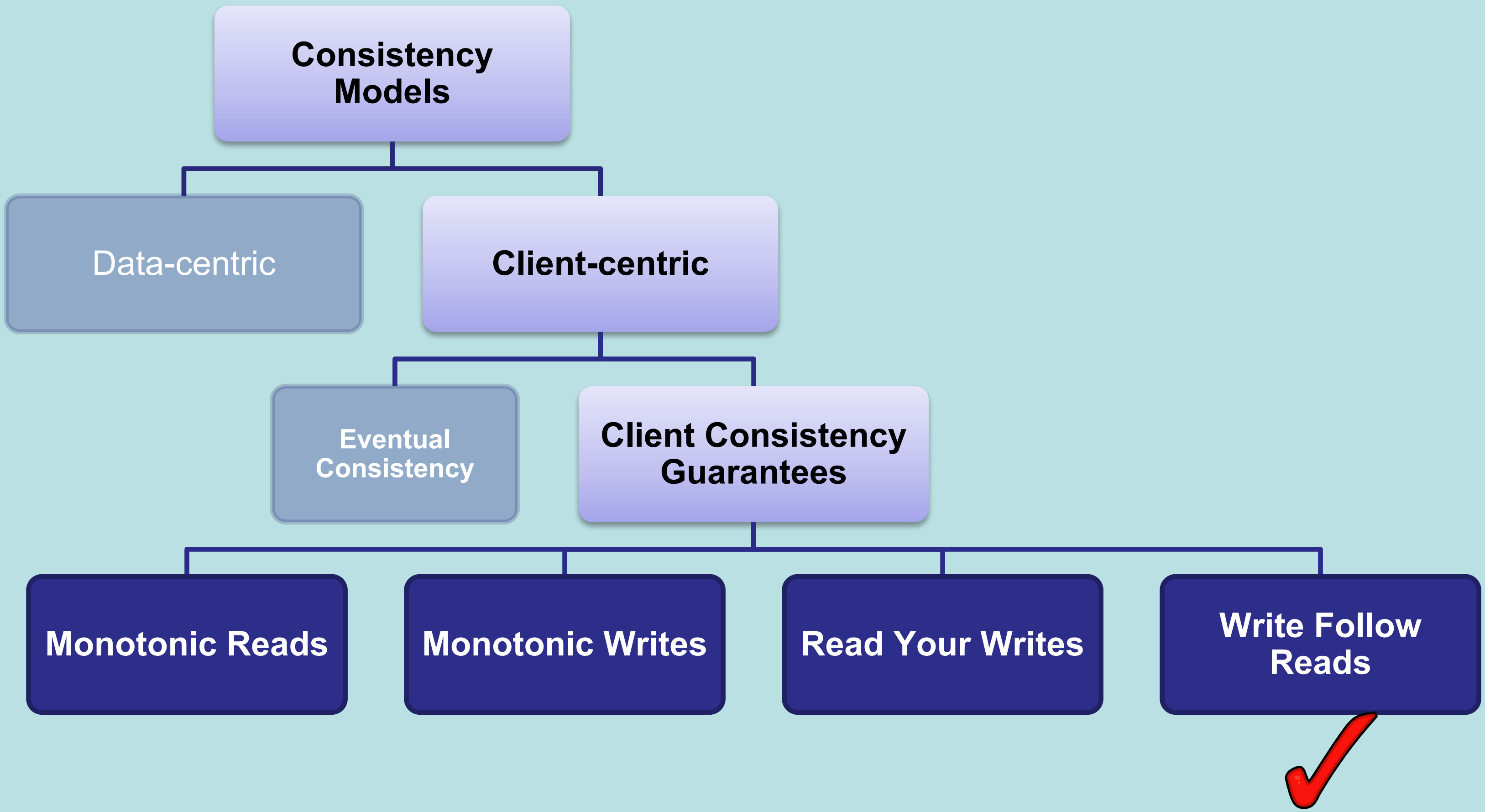


$R(x_2)$ operation should be performed only after the updating the Write Set $WS(x_1)$ at L_2



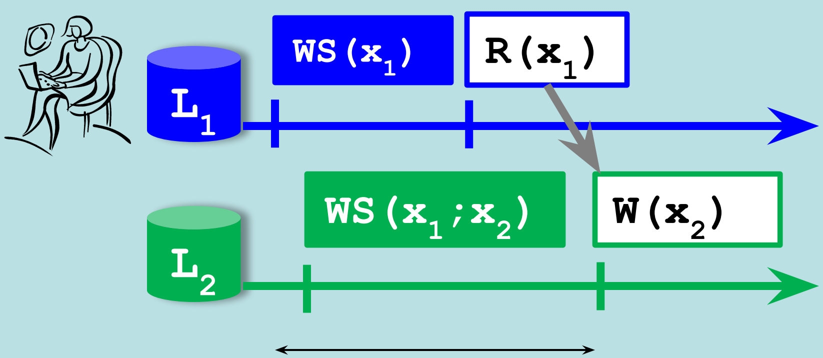
A data-store that does not provide *Read Your Write* consistency

Overview

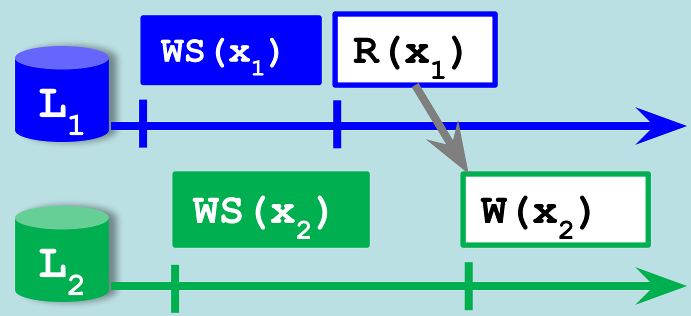

Consistency Models
Data-centric
Client-centric
Eventual Consistency
Client Consistency Guarantees
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

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 the all previous writes have been seen

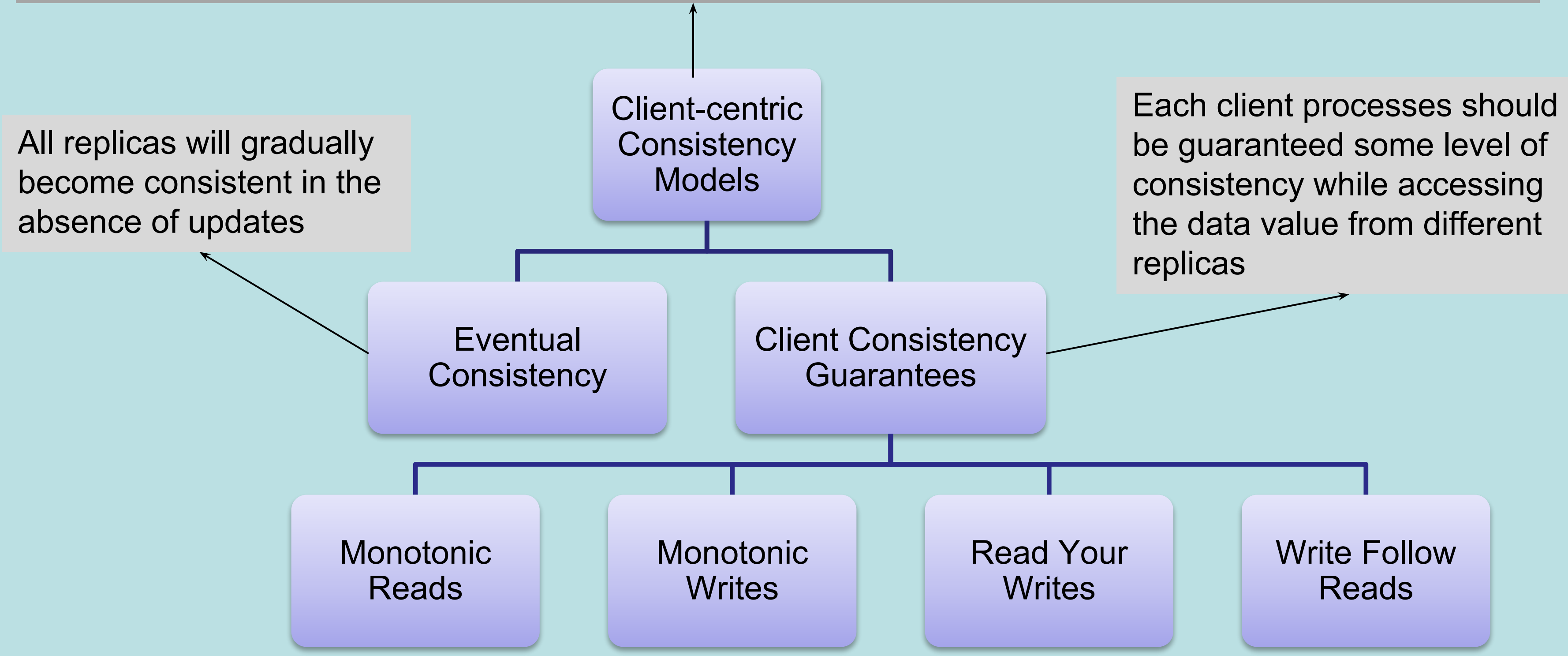


A data-store that does not guarantee Write Follow Read Consistency Model

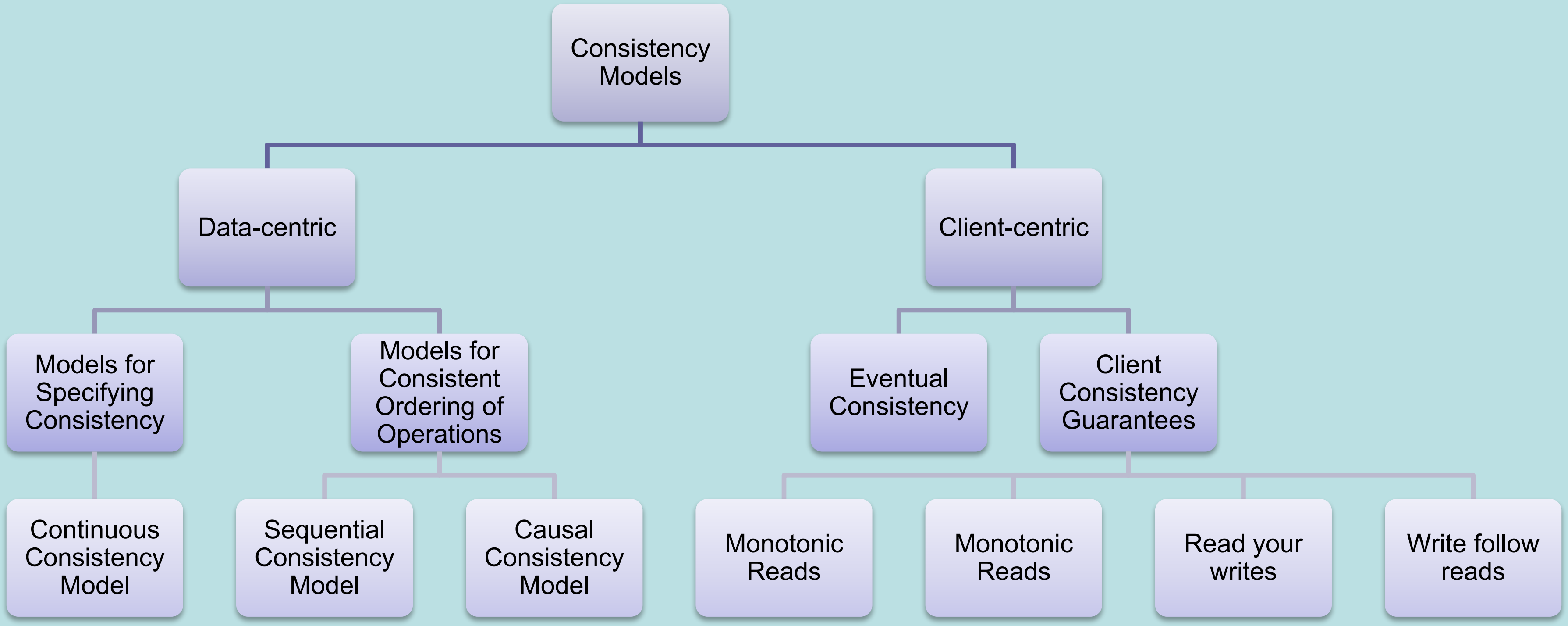
Summary of Client-centric Consistency Models

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 when
    - Here, 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