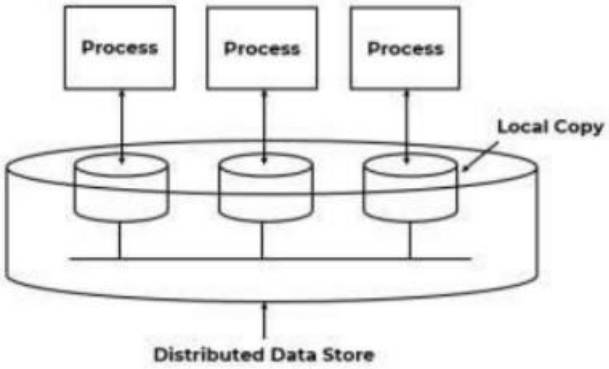
Q. Discuss different data-centric consistency models in detail.

Data-centric consistency models aim to ensure that all processes in a distributed system view the data store consistently, even though the data may be distributed across multiple machines. Here's a simplified explanation of different data-centric consistency models:

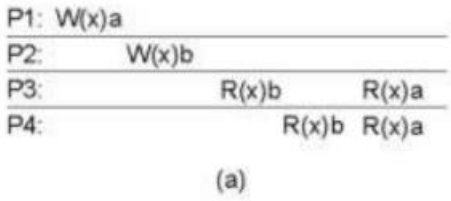


1. Strict Consistency:

- **Definition**: Strict consistency means that any read on a data item will return the value of the most recent write on that data item.
- **Strength**: This is the strongest consistency model, with the strictest requirement for data coherence.
- **Example**: If two processes are working on the same data item, they must always get the latest value after any update.

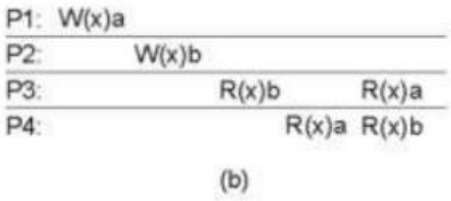
2. Sequential Consistency:

- **Definition**: A system is sequentially consistent if all operations (read and write) by all processes on the data store appear in a sequential order.
- **Requirement**: Each process's operations appear in order, and the execution of all operations should look like they happened one by one in some order.
- **Difference from Strict Consistency**: Sequential consistency is weaker than strict consistency because it doesn't guarantee the most recent write, just an overall sequence of operations.



(a)

A sequentially consistent data store.



(b)

A data store that is not sequentially consistent.

3. Linearizability:

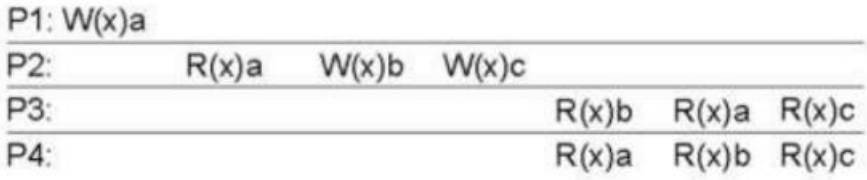
- **Definition**: Linearizability is similar to strict consistency but allows operations to be ordered by timestamps.
- **Requirement**: Operations are timestamped, and the result of any execution is as if all read/write operations happened in a sequential order.
- **Difference**: Linearizability is stronger than sequential consistency, but still weaker than strict consistency.

4. Causal Consistency:

- **Definition**: In causal consistency, only operations that are causally related are seen by all processes in the same order.
- **Requirement**: If one operation influences another, they must be seen in the correct order by all processes.
- **Difference**: Operations that are not causally related can appear in different orders to different processes.

5. FIFO Consistency:

- **Definition**: FIFO consistency ensures that all write operations from a single process are seen by all other processes in the exact order they were written.
- **Strength**: This is simpler to implement than other models and ensures good performance because it treats processes like a pipeline.
- **Difference**: This is weaker than causal consistency, as it doesn't care about causality, just the order of writes from the same process.



6. Weak Consistency:

- **Definition**: The weak consistency model does not enforce consistency on every operation but instead focuses on synchronizing memory at certain points.

- **Requirement**: It uses special variables called synchronization variables, which ensure that memory changes are visible across processes when needed.
- **Difference**: Unlike stronger models, weak consistency doesn't require operations to be strictly ordered.

7. Release Consistency:

- **Definition**: In this model, consistency is maintained by knowing when a process enters or exits a critical section.
- **Requirement**: Synchronization variables (acquire and release) are used to indicate when a process is working in a critical section and when it is done.
- **Difference**: This model is more focused on synchronization events rather than strict memory consistency.

8. Entry Consistency:

- **Definition**: Entry consistency is a variation of release consistency where each shared data item has its own synchronization variable.
- **Requirement**: Before accessing shared data, a process must acquire the synchronization variable associated with that data item.
- **Difference**: While release consistency affects all shared data, entry consistency only affects the shared data linked with specific synchronization variables.

Summary of Data-Centric Consistency Models:

1. **Strict Consistency**: Most rigid, always returns the most recent write.
2. **Sequential Consistency**: All operations are in a sequential order.
3. **Linearizability**: A bit weaker than strict consistency but maintains timestamped order.
4. **Causal Consistency**: Only causally related operations are seen in order.
5. **FIFO Consistency**: Ensures writes from one process are seen in order by all others.
6. **Weak Consistency**: Synchronization is done only when needed using special variables.
7. **Release Consistency**: Focuses on synchronization when entering/exiting critical sections.
8. **Entry Consistency**: Only data linked to synchronization variables is considered consistent.

Each of these models provides a trade-off between performance and consistency, depending on the requirements of the system.

Q. Discuss various client centric consistency models.

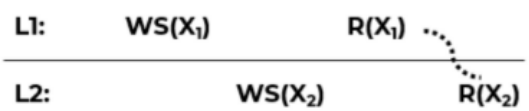
Client-centric consistency models focus on ensuring that each individual client has a consistent view of data, even in distributed systems where data is replicated across multiple machines. These models are easier to implement than system-wide consistency models and allow for some flexibility in dealing with inconsistencies. Here's a simplified explanation of the different client-centric consistency models:

1. Eventual Consistency:

- **Definition**: Eventual consistency is a weak consistency model where, if no updates occur for a long time, all copies of the data will eventually become consistent.
- **Key Points**:
    - There are no guarantees of immediate consistency after updates.
    - This model allows temporary inconsistencies but ensures that, over time, the system will become consistent.
    - **Example**: Websites (like social media platforms) where updates to posts or information might not immediately appear for all users but will eventually sync.

2. Monotonic Reads:

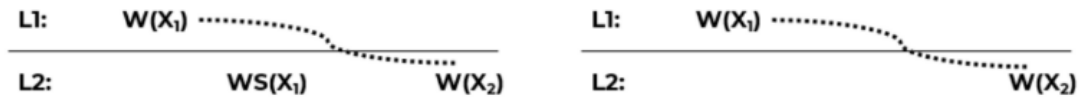
- **Definition**: Monotonic reads guarantee that once a client reads a data item, any future reads of that same item will return the same value or a more recent value.
- **Key Points**:
    - If a client reads data at one point, it will not see an older version of that data in future reads.
    - **Example**: If you check your email in one location and then check it again in another location (e.g., from Mumbai to Pune), you will always see the same or updated messages.



3. Monotonic Writes:

- **Definition**: Monotonic writes ensure that if a process writes to a data item, any future writes by the same process will be applied in the order they were made.

- **Key Points**:
    - This model guarantees that updates happen in the order they were initiated by the process.
    - **Example**: If you are updating a software library by replacing certain functions, the system ensures that earlier updates are applied before later ones.

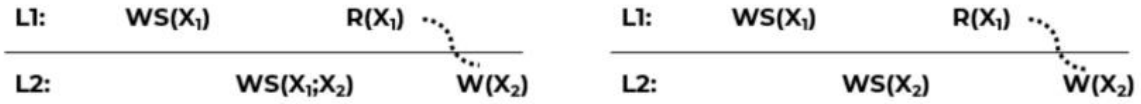


4. Read Your Writes:

- **Definition**: Read Your Writes ensures that after a client writes data, any subsequent read by that client will reflect the most recent write.
- **Key Points**:
    - If you write something, you can immediately read it back and see the changes.
    - **Example**: When you update a webpage, your browser will immediately show the newest version, not an older cached copy.

5. Writes Follow Reads:

- **Definition**: Writes Follow Reads guarantees that a write operation will always happen on a version of the data that is the same or more recent than what was read by the client.
- **Key Points**:
    - This model ensures that any changes made after reading data will be based on the most recent version of the data.
    - **Example**: If you read an article (A) and then post a comment (B), the system will ensure that your comment (B) is written only after the article (A) is updated.



Summary of Client-Centric Consistency Models:

1. **Eventual Consistency**: Allows temporary inconsistencies but ensures data becomes consistent over time.
2. **Monotonic Reads**: Guarantees that once a client reads data, future reads will return the same or updated value.
3. **Monotonic Writes**: Ensures that writes happen in the correct order for a given client.
4. **Read Your Writes**: Guarantees that a client will see its own write immediately.
5. **Writes Follow Reads**: Ensures that any write follows the most recent read by the same client.

These models offer flexible ways to handle inconsistencies in a distributed system, providing a balance between consistency and performance.

Q. Explain the difference between Data Centric and Client Centric Consistency Models.

Data-Centric Consistency	Client-Centric Consistency
Applied to **all users** in the system.	Applied to **each user (client)** individually.
Ensures **same data view for all clients** at the same time.	Ensures **consistent data view for a specific client**.
Handles **simultaneous updates** properly.	May not handle **conflicting updates** from different clients.
Designed to keep **all data copies in sync globally**.	Designed to give a **smooth experience to a single user**.
Harder to implement, but gives stronger consistency.	**Easier to implement**, but gives weaker consistency.
Mainly used where **accuracy and global consistency** are critical.	Used where **user experience and speed** are more important.
Example: **Banking systems**, where all users must see the same balance.	Example: **Social media**, where a user sees their own updates quickly.
Focuses on the **entire system's behavior**.	Focuses on the **individual client's behavior**.
Considers **causal, sequential, strict consistency models**.	Considers **monotonic reads/writes, read-your-writes** etc.

Q. Explain the difference between strict consistency and sequential consistency model

Strict Consistency Model	Sequential Consistency Model
Proposed by **R. Denvik**.	Proposed by **Leslie Lamport**.
A **read returns the most recent write**, instantly.	A **read returns values in a correct order**, but not necessarily the latest one.
Stronger model — very strict rules.	**Weaker model** — more flexible.
Needs an **absolute global time** to know exact operation order.	**No global time** needed; just maintains a consistent order.
Hard to implement in **real-world distributed systems**.	Easier to implement than strict consistency.
Guarantees that all clients always see **exact same and most recent** data.	Guarantees that all clients see the **same order**, but data may be **slightly older**.
Example: Like reading the latest post on a blog immediately after it's published.	Example: Everyone sees the posts in the same order, but not necessarily the latest post.
Performance is lower because of strict timing rules.	**Better performance**, as it allows more flexibility.
Used where **real-time accuracy** is very important.	Used where **operation order matters** more than real-time accuracy.
Not practical in large distributed systems.	**Commonly used** in distributed systems.

Q. Write a short note on fault tolerance.

Fault tolerance refers to the ability of a system to keep working even when some parts of it fail. Here's a simple explanation of the concept:

1. **System Failure**: A system is said to fail when it does not meet the expectations or promises it made. For example, if a file system stops working or provides incorrect data, it has failed.

2. **Error**: An error happens when something goes wrong inside the system, which may lead to failure.
3. **Fault**: The reason behind an error is called a fault. Faults can be:
    - **Transient**: Temporary faults that go away after some time.
    - **Intermittent**: Faults that appear occasionally and unpredictably.
    - **Permanent**: Faults that don't go away until fixed.
4. **Fault Tolerance**: It's the ability of a system to continue operating even when some parts of it fail. In **distributed systems** (where components are spread across different machines), fault tolerance is very important to ensure the system remains reliable.
5. **Importance in Distributed File Systems**: Fault tolerance is critical in designing systems like distributed file systems, where data is spread across multiple computers. If one part fails, the system must still work.

Types of Faults That Affect Systems:

1. **Memory Loss**: If a processor loses its memory due to a crash, it can make the data inconsistent or incomplete.
2. **Disk Storage Problems**: If storage devices decay (break down), data can be lost or corrupted. For example, if a part of a hard disk becomes damaged, the data in that area may become unreadable.

Key File Properties That Help Systems Handle Faults:

1. **Availability**:
    - Refers to how often a file is accessible.
    - If there is a communication problem, some clients might not be able to access the file while others can.
    - **Replication** (having copies of files in different locations) is a way to increase availability.
2. **Robustness**:
    - Refers to the system's ability to survive crashes or storage failures.
    - Using **redundancy** (extra copies of data) helps store data in a way that can survive hardware failures.
    - Sometimes, a file may not be available until the faulty part is fixed, but the system can recover.
3. **Recoverability**:

- Refers to the system's ability to go back to a safe and consistent state if something goes wrong.
- **Atomic updates** (like transaction mechanisms) are used to ensure that if an operation is interrupted, the system can safely undo it and return to a stable state.

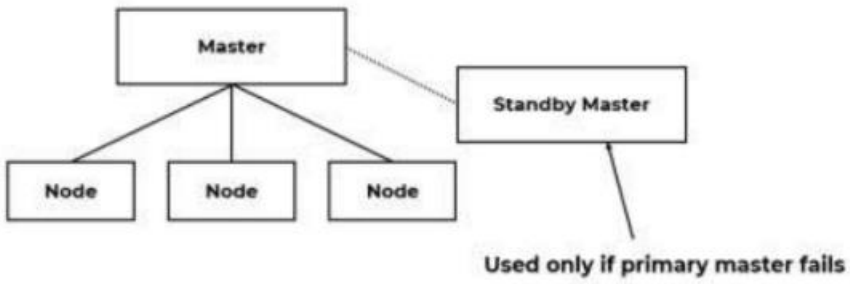
Q. Explain replication models.

Replication models help in managing multiple copies of data to keep them consistent across different systems. Below are the main models with simple explanations:

I) Master–Slave Model:

1. One copy is called the **master**, and all the other copies are called **slaves**.
2. **Slaves** can **only read data**, but they **cannot make changes**.
3. The **master** is the only one that can make updates or changes to the data.
4. If any change is made on the slave, it is usually **ignored or undone** during synchronization with the master.
5. This model is simple and **easy to manage** because everything is controlled by the master.

Analogy: Think of the **master** as the "main" book, and **slaves** are like "photocopies" that you can read, but not write on. Any changes made on the photocopies are not allowed.

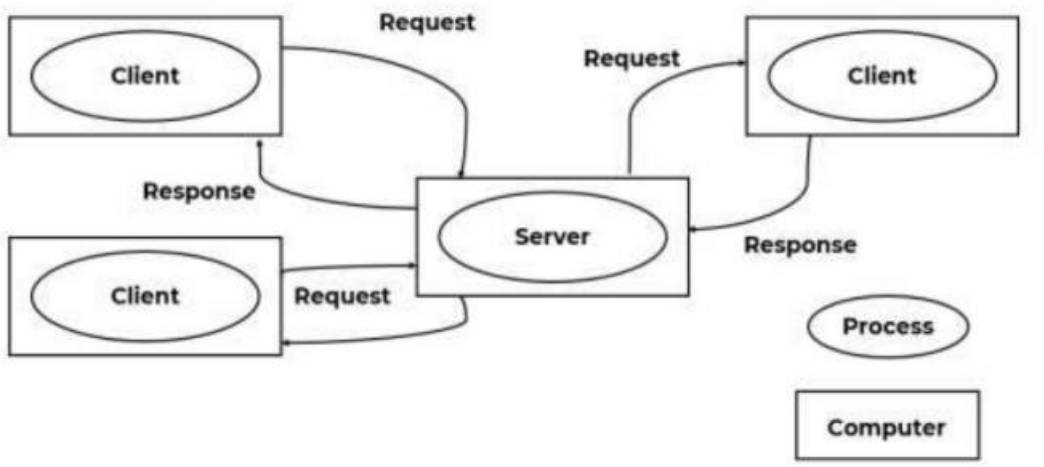


II) Client–Server Model:

1. This model is similar to master-slave, but here, the **server** handles all updates and serves multiple **clients**.
2. **Clients** send updates to the **server**, which then updates other clients.
3. The **server acts as the central point** where all updates are gathered and distributed.
4. Any **conflicts** are handled **only by the server**.

5. This setup helps keep everything **organized and consistent**, as the server handles all changes.

Analogy: Imagine a **teacher (server)** who collects homework from all the students (clients), grades it, and then hands back the corrected work to everyone. The teacher ensures everything is correct and consistent.



III) Peer-to-Peer (P2P) Model:

1. In this model, all copies are **equal** and there is no master or server.
2. **Any copy (peer)** can be updated and can **sync with any other copy**.
3. Updates are shared more quickly because **any peer can connect with any other**.
4. This model provides a **rich communication framework** but is **more complex** to manage.
5. It works well in smaller systems but can face challenges in **scaling up**.

Analogy: Think of a group of **friends** who share notes with each other. No one is in charge, and everyone can share their notes or get updates from anyone else. However, if the group grows too big, it might become harder to keep everything organized.