Unit-7

Consistency and Replication

- **Replication:**

⇒An important issue in distributed system is the replication of data. Data are generally replicated to enhance **reliability,** or improve **performance, high availability** and **fault tolerance.**

⇒Replication is widely used for example, the caching of resources from web servers in browser and web proxy servers is a form of replication, since the data held in caches and at server are replicas of one another. That is replication is the mechanism of maintaining multiple copies of data at multiple nodes.

⇒From the client's viewpoint , there is a single logical copy of data. Update at one copy of replica by a client should be reflected to all other replicas.

- **Reason for replication:**

The main aim of replication is to provide backup on the scenario of failure. There are three reasons for replication. They are:

1. **Performance Enhancement:** The copy of data is placed in multiple location. So, client can get data from nearby location. This decreases the time take to access the data. It enhance performance of the distributed system. Multiple servers located at different locations provided the same services to the client. It allows parallel processing of the client request to the resources or computation.
2. **Increased availability:** Replication is a technique for automatically maintaining the availability of data despite server failures. If data are replicated at two or more failure-independent servers, then client software may be able to access data at an alternative server.
3. **Fault Tolerance:** Replication ensures the correctness of data in addition to availability. If a server fails, the data can be accessed from other servers. If a server of a group of n server provides faulty information, the other servers can outvote the faulty server and provide correct data to this client.

“A fault-tolerance service , by contrast , always guarantees strictly correct behavior despite a certain number and type of fault.”

- **Challenges in Replication:**

1. Placement of replicas

The major challenges in replication is where to put the replicas. There are three places to put replicas.

- **Permanent Replicas:** permanent replicas consist of cluster of servers that may be geographically dispersed.
- **Server initiated replicas:** Server initiated caches include placing replicas in the hosting servers and server caches.
- **Client initiated replicas:** Client initiated replicas include web browsers cache.

2. Propagation of updates among replicas

The net challenges is to how to propagate the updates in one replica among all the replicas efficiently and faster as possible.

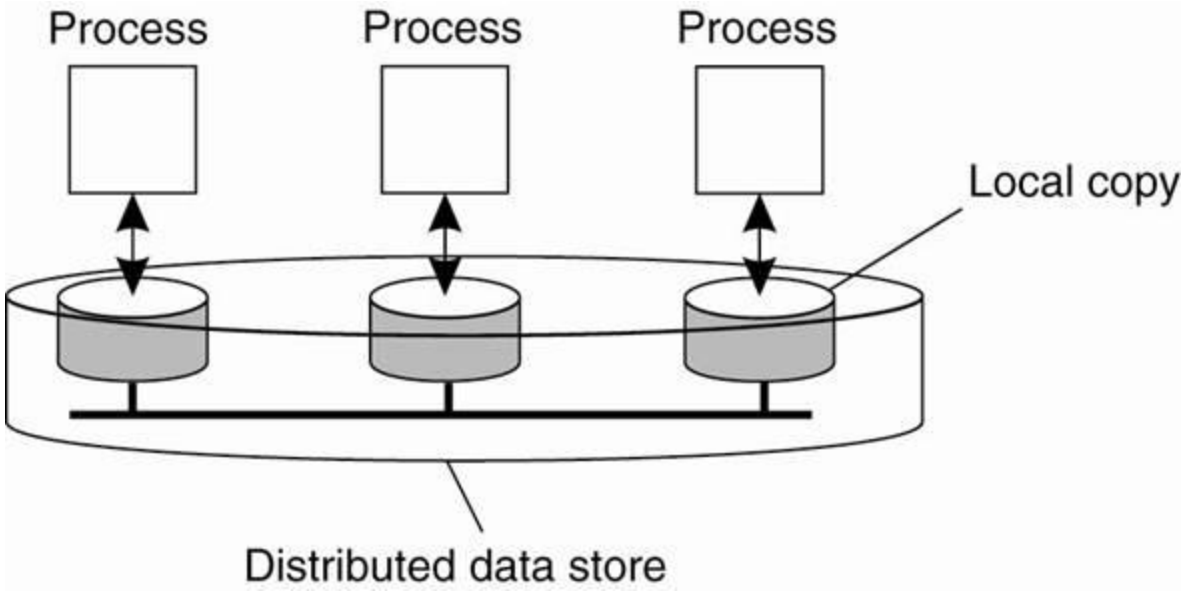
- **Push based propagation:** A replica in which update occurs pushes the update to all other replica.
- **Pull based propagation:** A replicas requests another replica to send the newest data it has.

3. Lack of consistency: if a copy if modified, the copy becomes inconsistent from the rest of copies. It takes some time for all the copies to be consistent.

- **DATA-CENTRIC MODELS:**

A data store may be physically distributed across multiple machines. Each process that can access data from the store is assumed to have a local or nearby copy available of the entire store.

Specifically, it determines how data is accessed and updated across multiple nodes in a distributed system, and how these updates are made available to clients.



- There are various types of consistency models available in distributed systems. Each consistency model has its own strengths and weaknesses, and the choice of model depends on the specific requirements of the system.
- **Types of Consistency Models**

1. Strict Consistency:

Any read on a data item ‘x’ returns a value corresponding to the result of the most recent write on ‘x’ (regardless of where the write occurred).

With *Strict Consistency*, all writes are *instantaneously visible* to all processes and *absolute global time order* is maintained throughout the distributed system.

Example:

- **Begin with:**

Wi(x)a –write by process Pi to data item x with the value a

Ri(x)b - read by process Pi from data item x returning the value b

- **Assume:**

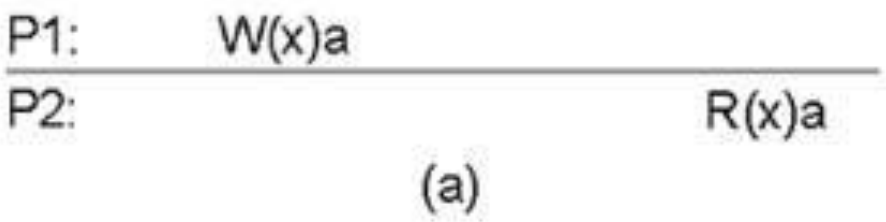
Time axis is drawn horizontally, with time increasing from left to right

Each data item is initially NIL.

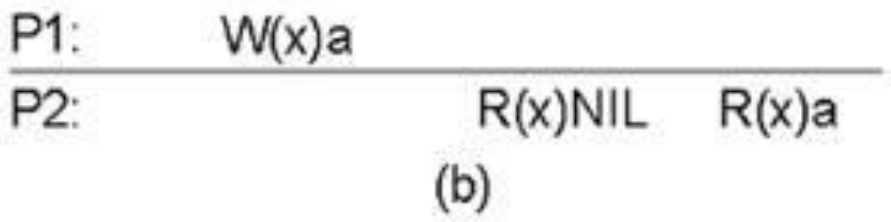
P1 does a write to a data item x, modifying its value to a.

- Operation W1(x)a is first performed on a copy of the data store that is local to P1, and is then propagated to the other local copies.
- P2 later reads the value NIL, and some time after that reads a (from its local copy of the store).

NOTE: it took some time to propagate the update of x to P2, which is perfectly acceptable.



(a)



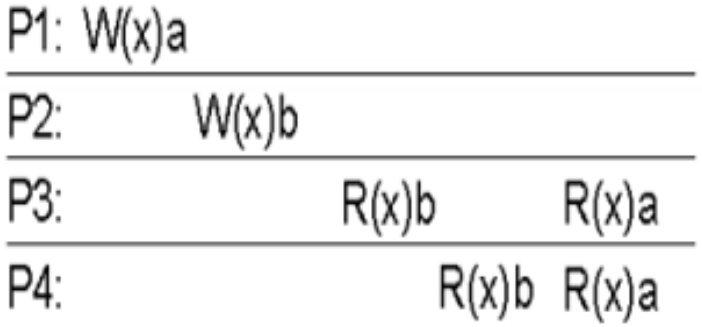
(b)

2. **Sequential Consistency:**

Sequential consistency is an important data-centric consistency model , which was first defined by lamport in the context of shared memory for multiprocessor system. In general, a data store is said to be sequential consistent when it satisfies the following condition.

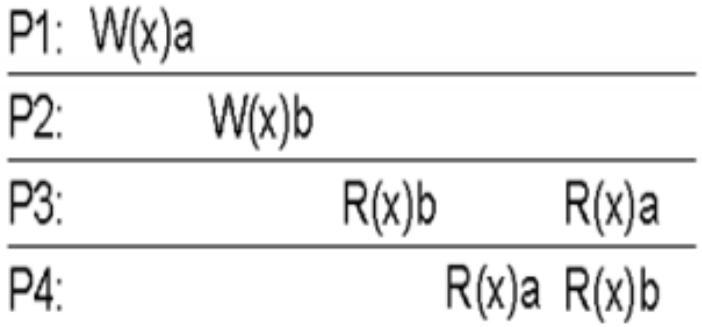
The result of any execution is the same as of the (read and write) operation by all process on the data store were executed in some sequential order and the operation if each individual process appear in this sequence in the order specified by its program.

- Example:



(a)

A sequentially consistent data store.



(b)

A data store that is not sequentially consistent.

- **Linearizability:**
- Weaker than strict but stronger than sequential.
- Operation are assumed to receive a timestamp using a global available clock, but with finite precision.(i.e. with the assumption that process use loosely synch. Clocks.)
- Assumes sequential consistency and

- If TS(x)<TS(y) then OP(x) should precede OP(y) in the sequence

<u>Causal Consistency:</u>

The causal consistency model represents a weakening of sequential consistency in that it requires a total order of causal related write operation only.

- A read is causally related to the write that provided the data the read got.
- A write is causally related to a read that happened before this write in the same process.
- If $write_1$ -> read, and read -> $write_2$, then $write_1$ -> $write_2$.

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

Client Centric Consistency model:

- 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.
- The goal of client-centric consistency models is to show how we can perhaps avoid system wide consistency, by concentrating on what specific clients want, instead of what should be maintained by servers.

- **Eventually consistency**

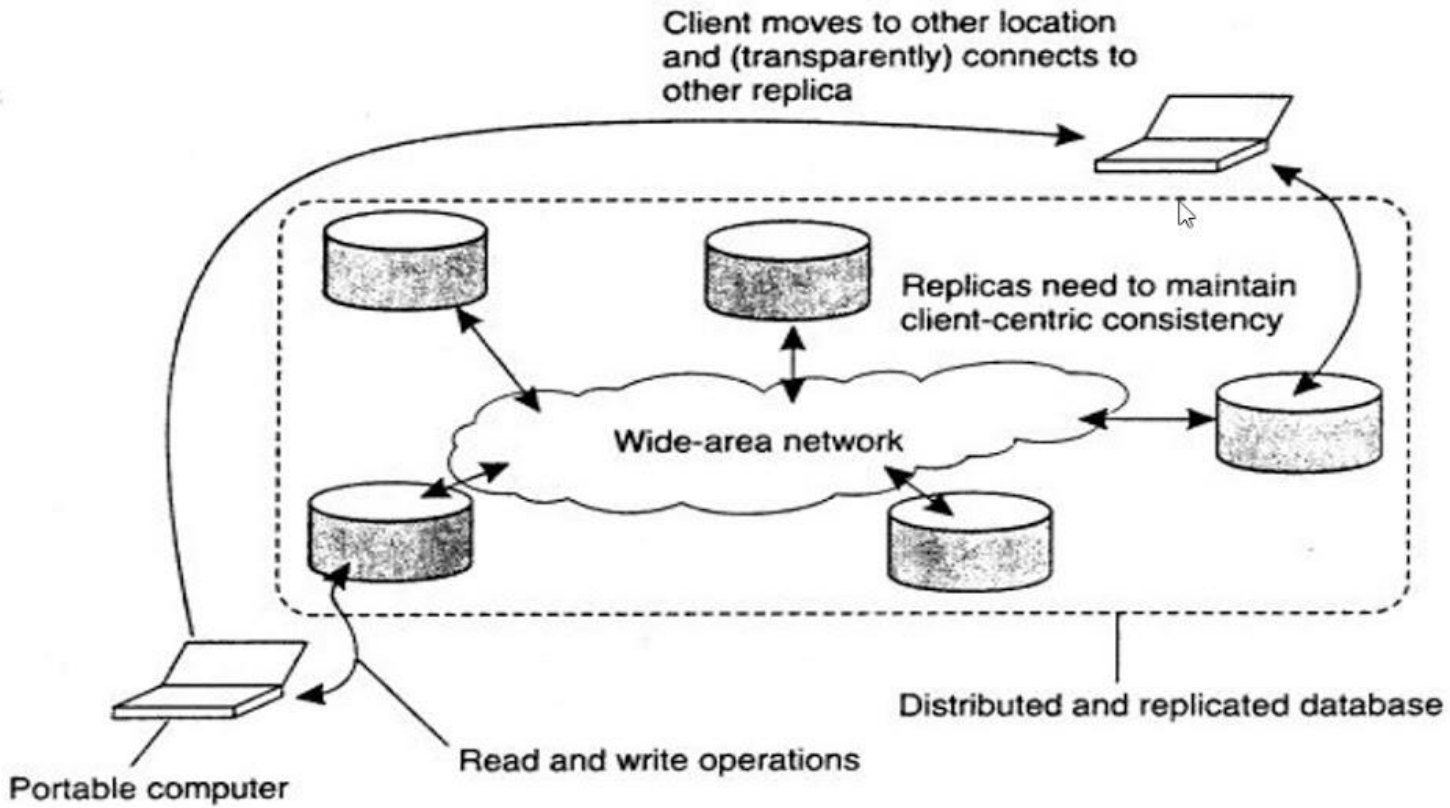


Figure '-11. The principle of a mobile user accessing different replicas of a distributed database.

- This example is typically eventually consistent data stores and is caused by the fact that users may sometimes operates on different replicas while updates have not been fully propagated. The problem can be alleviated by introducing **client-centric consistency.** In essence, they ensure that whenever a client connects to an new replica , that replica is up to date according to the previous access of the client to the same data in the other replicas on different **sites.**

- **Types of client centric consistency model.**

1. **Monotonic reads**
2. **Monotonic writes**
3. **Read your writes**
4. **Write follow reads.**

- **Monotonic reads:**

A data store is said to provide *monotonic-read consistency* if the following condition holds:

If a process reads the value of a data item x, any successive read operation on x by the process will always return that same value or a more recent value.

For example: automatically reading your personal calendar updates from different servers. Monotonic reads guarantees that the user sees all updates, no matter from which server that automatic reading takes place. Another example is reading (no modifying) incoming mail while you are on the move. Each time you connect to a different email server, that server fetches all the updates from the server you previously visited.

- **Monotonic Writes:**

A data store is said to monotonic writes consistency if the following condition holds:

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

For examples: updating a program at server S2, and ensuring that all components on which compilation and linking depends, are also placed at s2.

- **Read your writes:**

A data store is said to provide read-your-write consistency if the following condition holds:

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

- For example: updating your web page and guaranteeing that your web browser shows the newest version instead of its cached copy.

Writes Follow Reads:

A data store is said to provide write-follow-reads consistency if the following condition holds:

> A write operation by a process on 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.

For example: see reaction to posted articles only if you have the original posting (a read "pulls in" the corresponding write operation).

REPLICA MANAGEMENT:

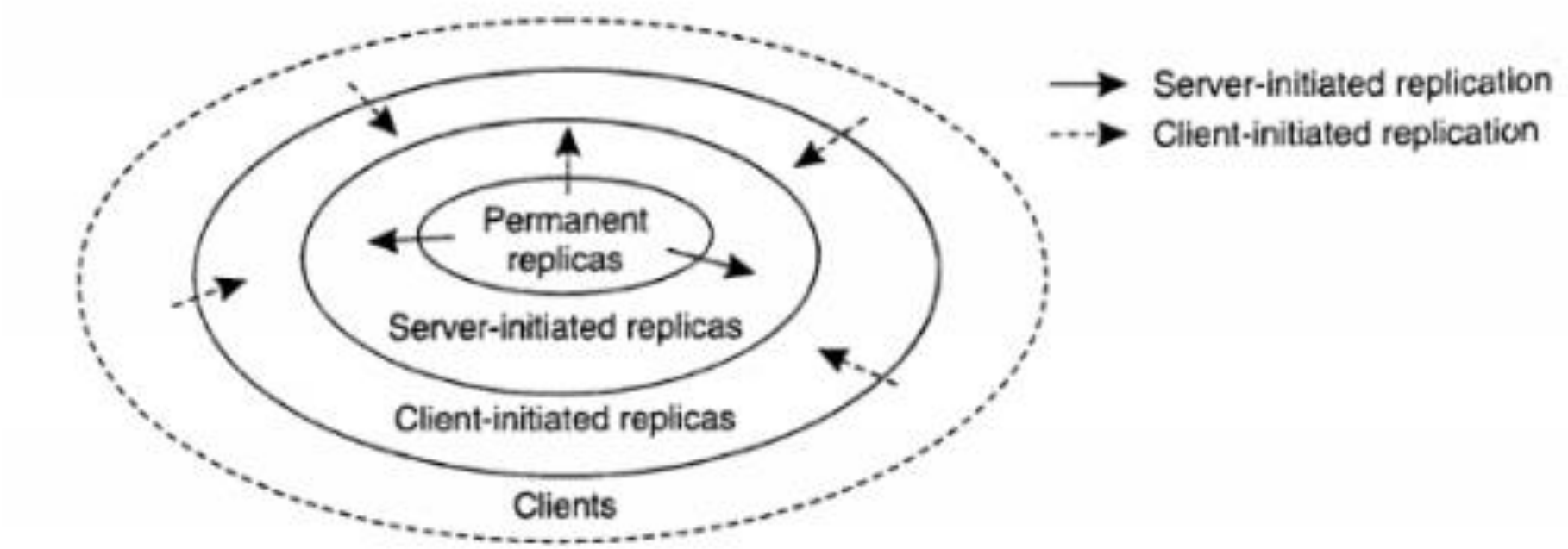
Replica management describes where,when and by whom replicas should be placed.

Two problems under replica management;

- Replica-server Placement: Decides the best locations to place the replica server that can host data-stores.
- Content replication and Placement; Finds the best server for placing the contents.

Finding the best server location:

- **Content Replication:**
- When it comes to content replication and placement, three types of replicas can be distinguished logically organized as shown in figure:



- **Permanent Replicas:**Process / Machine always having a replica. In many cases, the number of permanent replicas is small. Example website.
- **Server-Intended Replicas:**Process that can dynamically host a replica on request of another server in the data store. For example: a webserver placed in Kathmandu. Normally this server can handle incoming request quite, but it may happen that over a couple of days a sudden burst if request quite easily, but it may happen that from the server . In that case , it may be worthwhile to install a number of temporary replicas in regions where request are coming from.
- **Client intended replicas:** Process that can dynamically host a replica on request of a client . In essence , a cache is a local storage facility that is used by client to temporarily a copy of the data it has just requested. In principle , managing the cache is left entirely to the client.

- **Consistency Protocol:**

The main approaches are as follows:

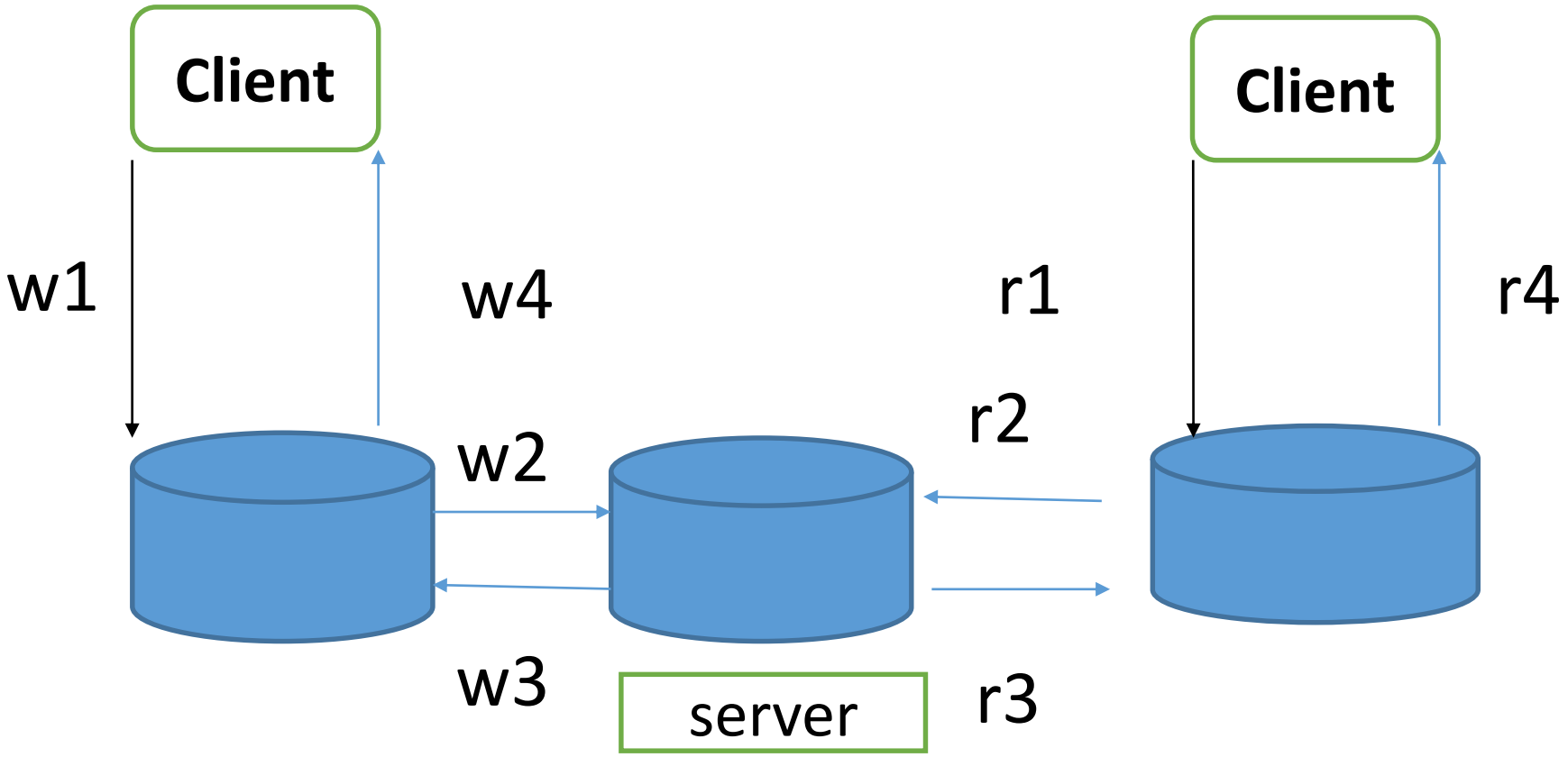
1. **Primary-based protocol**
2. **Replicated-write protocol**
3. **Cache-coherence protocol**

Primary-based protocol:

Primary-based consistency models employ one node in the system to ensure that this consistency is sustained across the distributed system.

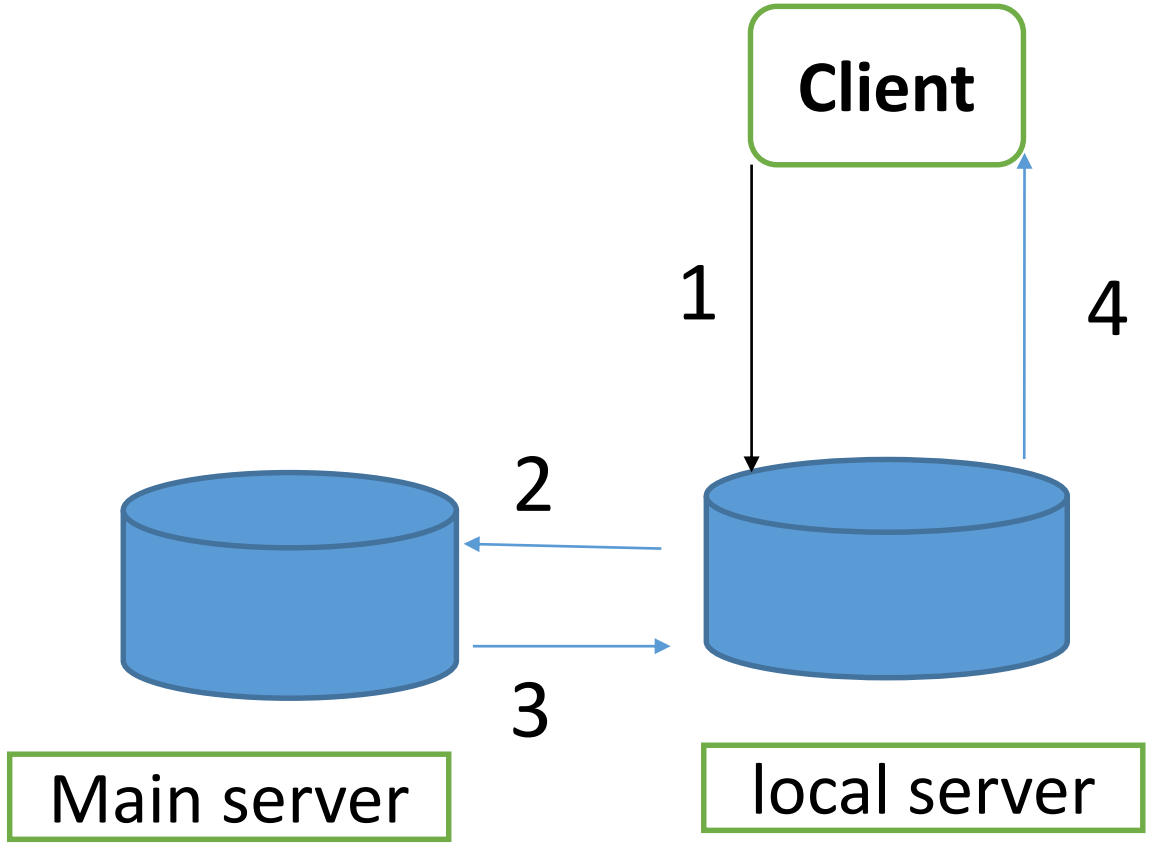
- **Remote-write protocols:**

The simplest primary-based protocol that supports replication is the one in which all write operation need to be forwarded to a fixed single server. Read operation can be carried out locally. Such schemes are also known as **primary-backup protocols.** A primary –backup protocol works as shown in figure.


Client
Client
w1
w4
r1
r4
r2
w2
w3
server
r3

Local write protocol:

A variant of primary backup protocols is one in which the primary copy migrates between processes that wish to perform a write operation. As before, whenever a process wants to update data item x, it locates the primary copy of x, and subsequently moves it to its own location.



- **Replicated write protocol:**

1. **Active replication:** Updates are forwarded to multiple replicas, where they are carried out.
2. **Replicated invocation:** Assign a coordinator on each side(client and server) which ensure that only one invocation and only one replay is send.
3. **Quorum-based protocols:** Ensure that each operation is carried out in such a way that majority vote is established . Majority is checked based in the request and replica is shifted to that db.

Cache-coherence protocol:

Caches from a special case of replication, in sense that they are generally controlled by clients instead of servers. However, cache coherence protocols, which ensure that aa cache is consistent with the-initiated replicas.

It works when inconsistency are detected. Which determine how caches are kept consistent with copies stored at server.

The simplest solution is that disallow shared data to be cached at all.