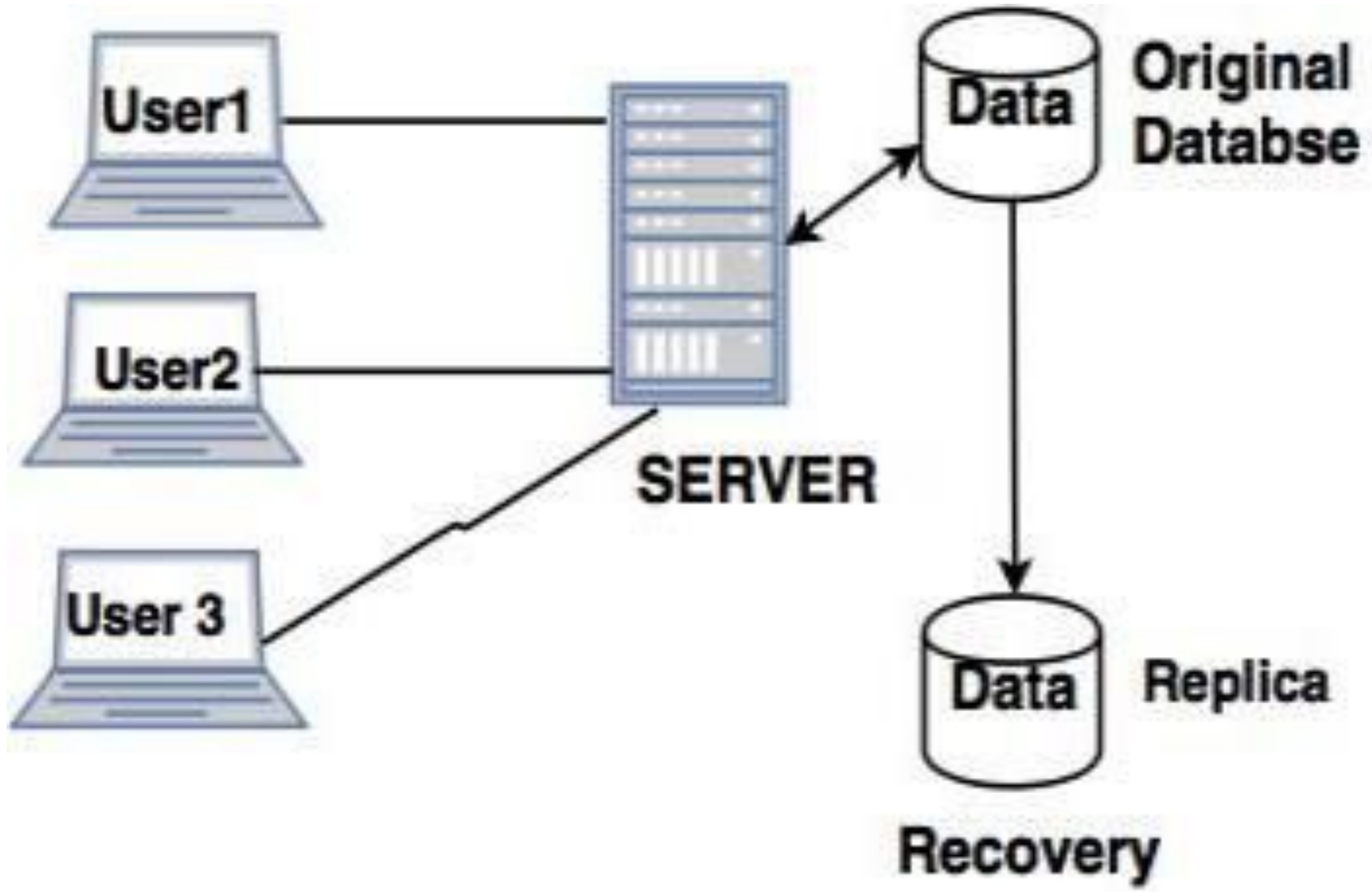
CHAPTER 7

CONSISTENCY AND REPLICATION

Replication in Distributed System

- In a distributed system data is stored is over different computers in a network.
- Therefore, we need to make sure that data is readily available for the users.
- **Availability** of the data is an important factor often accomplished by data replication.
- ***Replication is the practice of keeping several copies of data in different places.***



Full Replication Process In Distributed System

Why do we require replication?

- The first and foremost thing is that it makes our **system more stable because of node replication.**

It is good to have replicas of a node in a network due to following reasons:

- If a **node stops working, the distributed network will still work fine** due to its replicas which will be there.
- Thus it increases the **fault tolerance** of the system.
- It also helps in **load sharing** where loads on a server are shared among different replicas.
- It enhances the **availability of the data.** If the replicas are created and data is stored near to the consumers, it would be easier and faster to fetch data.

- **DISADVANTAGES OF DATA REPLICATION –**
  - **More storage space is needed** as storing the replicas of same data at different sites consumes more space.
  - Data Replication becomes **expensive** when the replicas at all different sites need to be updated.
  - **Maintaining Data consistency** at all different sites involves **complex measures.**

Types of Replication

- Active Replication
- Passive Replication

Active Replication:

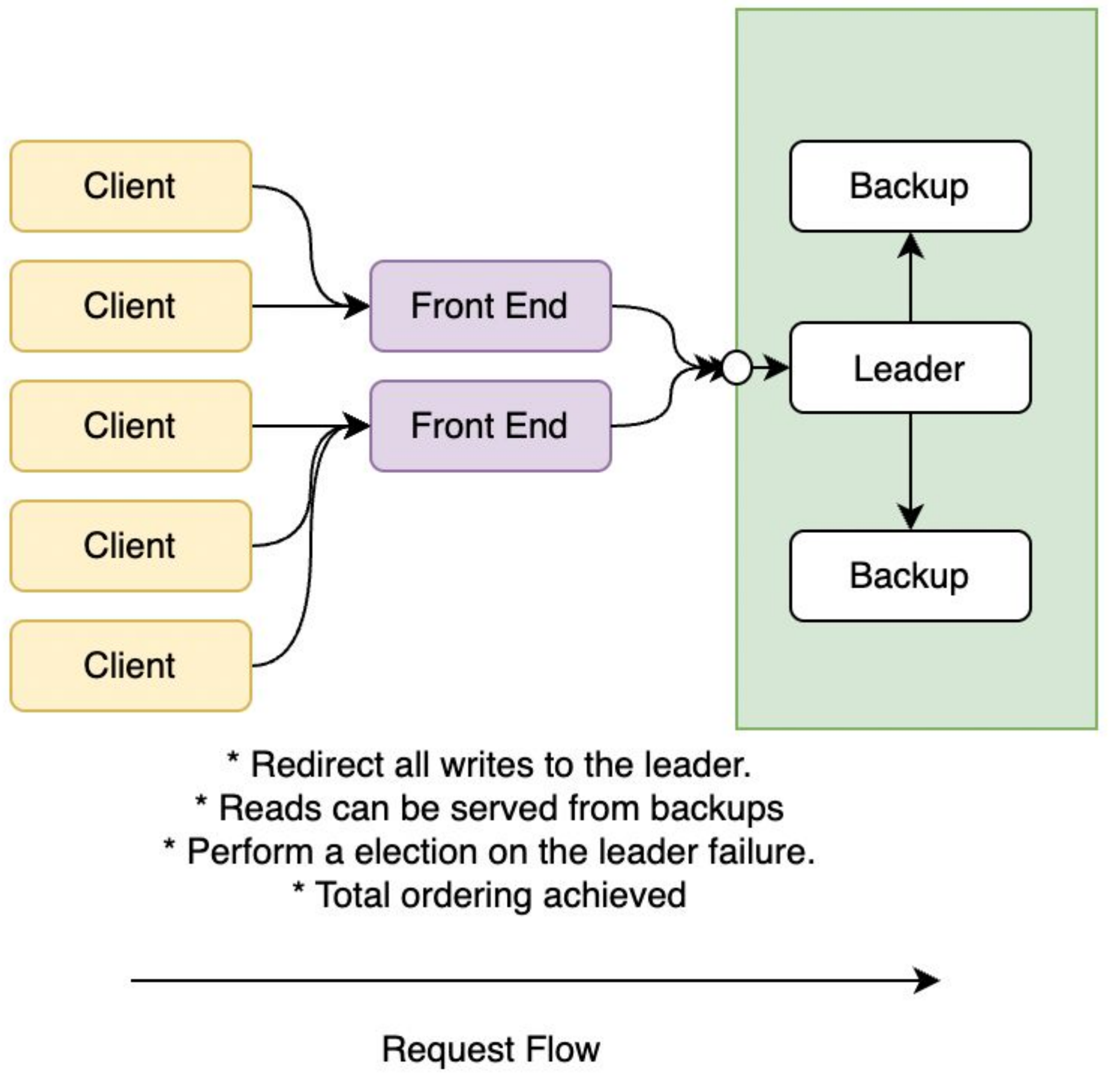
- The request of the client goes to **all the replicas.**
- It is to be made sure that every replica receives the client request in the same order else the system will get inconsistent.
- There is no need for coordination because each copy processes the same request in the same sequence.
- All replicas respond to the client’s request.

Passive Replication:

- The client request goes to the **primary replica, also called the main replica.**
- There are more replicas that act as **backup for the primary replica.**
- Primary replica informs all other backup replicas about any modification done.
- The response is returned to the client by a primary replica.
- Periodically primary replica sends some signal to backup replicas to let them know that it is working perfectly fine.
- In case of failure of a primary replica, a backup replica becomes the primary replica.

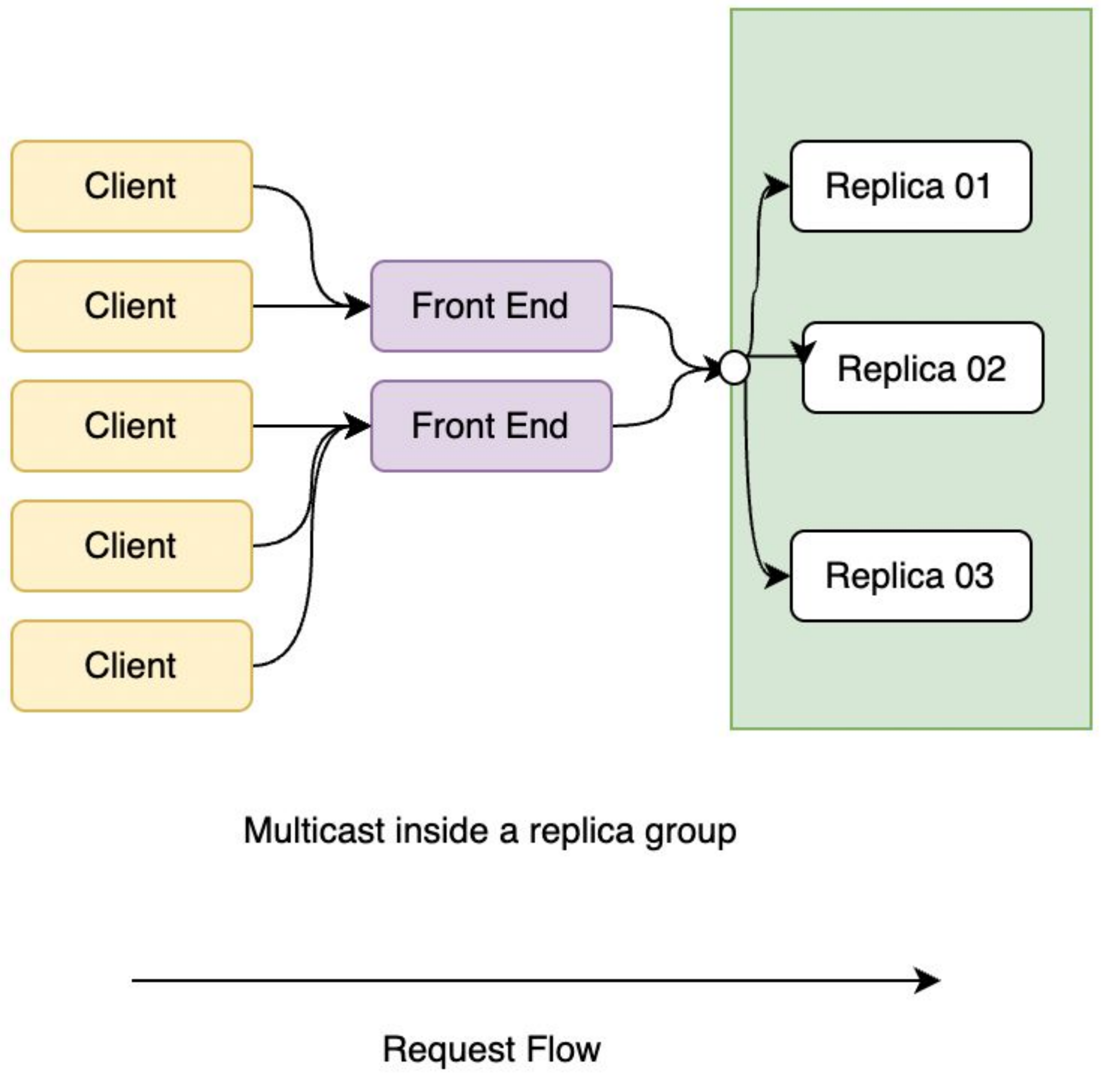
Active Replicas (or primary replicas) handle all writes and reads, while Passive Replicas (or backup replicas) passively synchronize with the primary, serving as backups and potentially for read-only operations.

Passive Replication



* Redirect all writes to the leader.
* Reads can be served from backups
* Perform a election on the leader failure.
* Total ordering achieved

Active Replication



Multicast inside a replica group

An important issue in distributed systems is the replication of data.

- Data are generally replicated **to enhance reliability or improve performance**
- One of the major **problems is keeping replicas consistent.**
- Informally, this means that **when one copy is updated** we need to ensure that the **other copies are updated as well**; otherwise the replicas will no longer be the Same.

First of all, we start with concentrating on managing replicas: -which takes into account not only the placement of replica servers, but also how content is distributed to these servers.

The second issue is how replicas are kept consistent.

-In most cases, applications require a strong form of consistency.

-This means that updates are to be propagated more or less immediately between replicas.

REASON FOR REPLICATION

- **Two primary reasons for replication: reliability and performance.**

Increasing reliability: – If a replica crashes, system can continue working by switching to other replicas.

Improving performance: – Important for distributed systems over large geographical areas.

Fault Tolerance: if the server fails data can be accessed from other servers .

Challenges in Replication

There is a price to be paid when data are replicated:

- The problems with replication are: Having multiple copies may lead to consistency problems.
- Whenever a copy is modified, that copy becomes different from the rest.

- Consequently, modifications have to be carried out on all copies to ensure consistency.
- Exactly when and how those modifications need to be carried out determines the price of replication.
- Cost of increased bandwidth for maintaining replication

Real World Example of Replication

A real-world example of replicas in a distributed system can be found in the context of large-scale web applications or databases.

One such example is the use of replica sets in distributed databases like MongoDB or clusters in distributed file systems like Hadoop's HDFS.

Example: MongoDB Replica Sets

MongoDB, a popular NoSQL database, uses replica sets to provide high availability and fault tolerance.

- **Primary and Secondary Nodes**: A MongoDB replica set typically consists of multiple nodes:
    - **Primary Node**: Handles all write operations and is the main node that clients interact with.
    - **Secondary Nodes**: Maintain copies of the data from the primary node and serve read operations.

- **Automatic Failover**: If the primary node fails (due to hardware failure, network issues, etc.), one of the secondary nodes is automatically elected as the new primary. This ensures that the database remains available even if one node goes down.
- **Data Replication**: MongoDB uses a mechanism called replication to synchronize data between nodes. The primary node logs all write operations (in the form of an oplog) and sends these operations to its secondary nodes. This way, each secondary node maintains an up-to-date copy of the data.

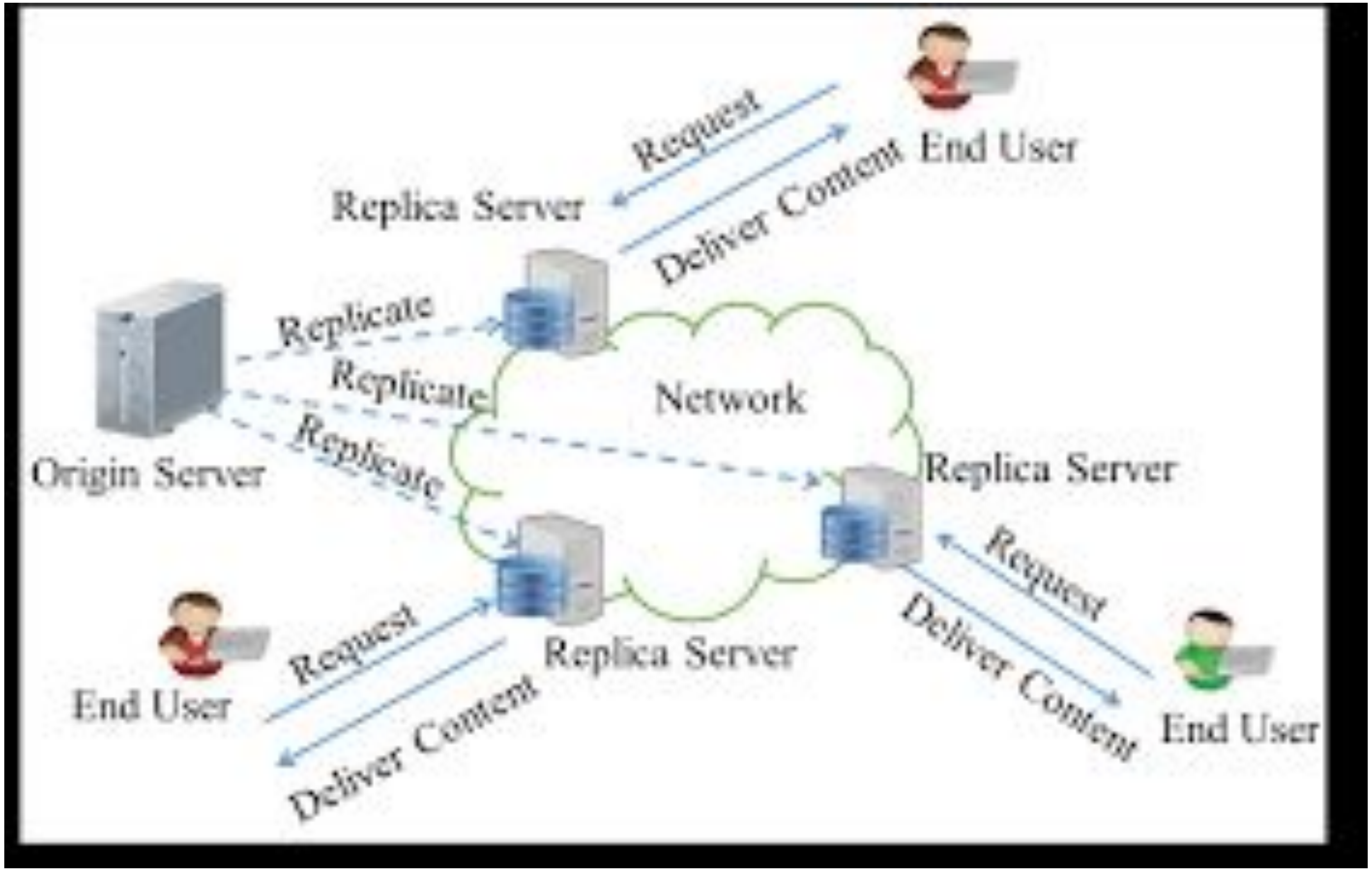
Benefits and Use Cases

- **High Availability**: Replica sets ensure that if one node fails, another node can take over, minimizing downtime.
- **Fault Tolerance**: By maintaining multiple copies of the data across different nodes, replica sets protect against data loss.
- **Read Scalability**: Secondary nodes can serve read operations, distributing the read workload and improving overall system performance.

Placement of replicas

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 server and server caches.
- Client initiated replicas: Client initiated replicas include web browser cache.


End User
Request
Deliver Content
Replica Server
Replicate
Replicate
Replicate
Origin Server
Network
Replica Server
Request
Deliver Content
End User
Replica Server
End User
Request
Deliver Content

1. Permanent Replica

- These replicas are always available and do not change dynamically.
- Typically found in **databases or distributed storage systems** where a fixed number of copies are maintained.
- Used to ensure **high availability and fault tolerance**.
- Example: A database system that maintains three fixed copies of data across different servers for redundancy.

2. Server-Initiated Replica

The server dynamically creates replicas based on demand, workload, or fault tolerance requirements.

Often used in **content delivery networks (CDNs)** and cloud computing environments.

Example: A cloud storage system like AWS S3 automatically creating new replicas when a data center is overloaded or fails.

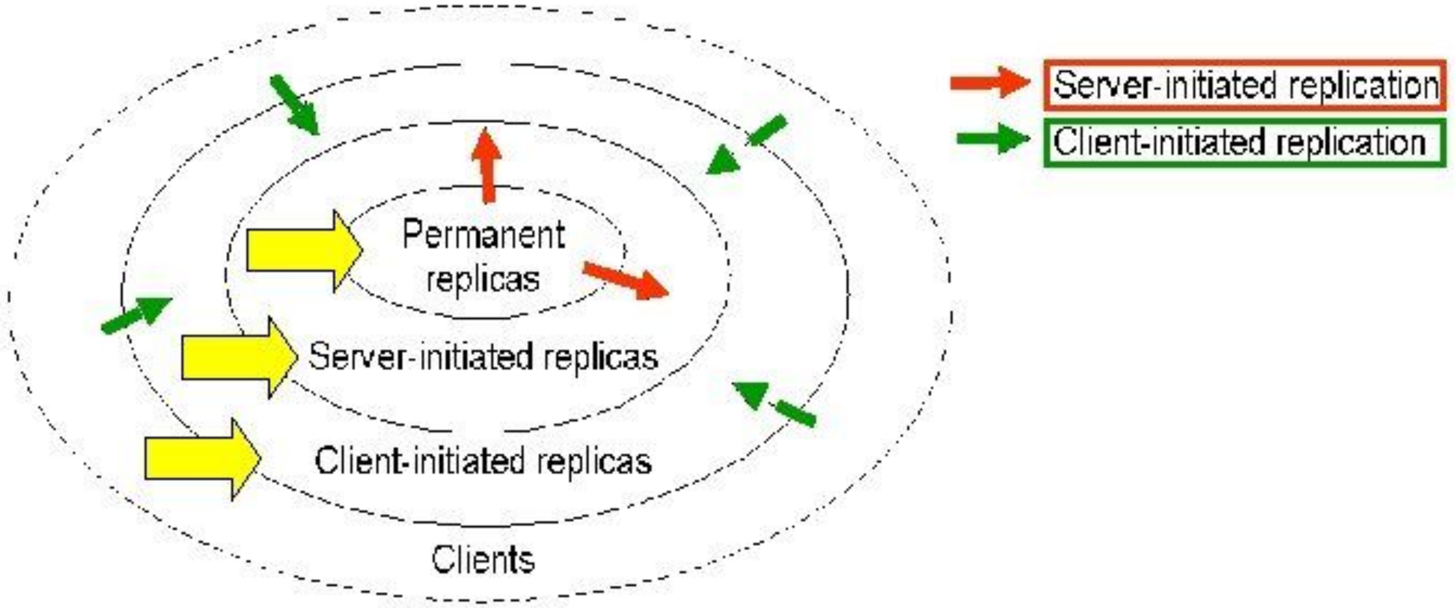
Client-Initiated Replica

- The client requests the creation of a replica, usually to **improve access speed or offline availability**.
- Typically used in **mobile applications, edge computing, and peer-to-peer networks**.
- Example: A user downloading a movie for offline viewing in a video streaming service, effectively creating a temporary replica.

Distribution Protocols

Replica Placement

Where, when, by whom copies of data are to be placed?


Server-initiated replication
Client-initiated replication
Permanent replicas
Server-initiated replicas
Client-initiated replicas
Clients

The logical organization of different kinds of copies of a data store into three concentric rings.

Propagation of updates among replicas

- Push based propagation: a replica in which updates occurs pushes the updates to all other replicas.

- Pull based propagation: a replicas requests another replica to send the newest data it has.

Lack of consistency

- If a copy is modified , the copy becomes inconsistent from the rest of copies .
- It takes some time for all the copies to be consistent.

Replication as a scaling technique

- Scalability issues generally appear in the form of performance problems.
- Placing copies of data close to the processes can improve performance through reduction of access time and thus solve scalability problems.
- A possible trade-off that needs to be made is that keeping copies up to date may require more network bandwidth.

Consistency Model in Distributed System

In distributed systems, consistency models establish criteria for data synchronization and **specify how users and applications should interpret data changes across several nodes.**

In a distributed system, it specifically controls **how data is accessed and changed across numerous nodes** and **how clients are informed of these updates.**

These models range from strict to relaxed approaches.

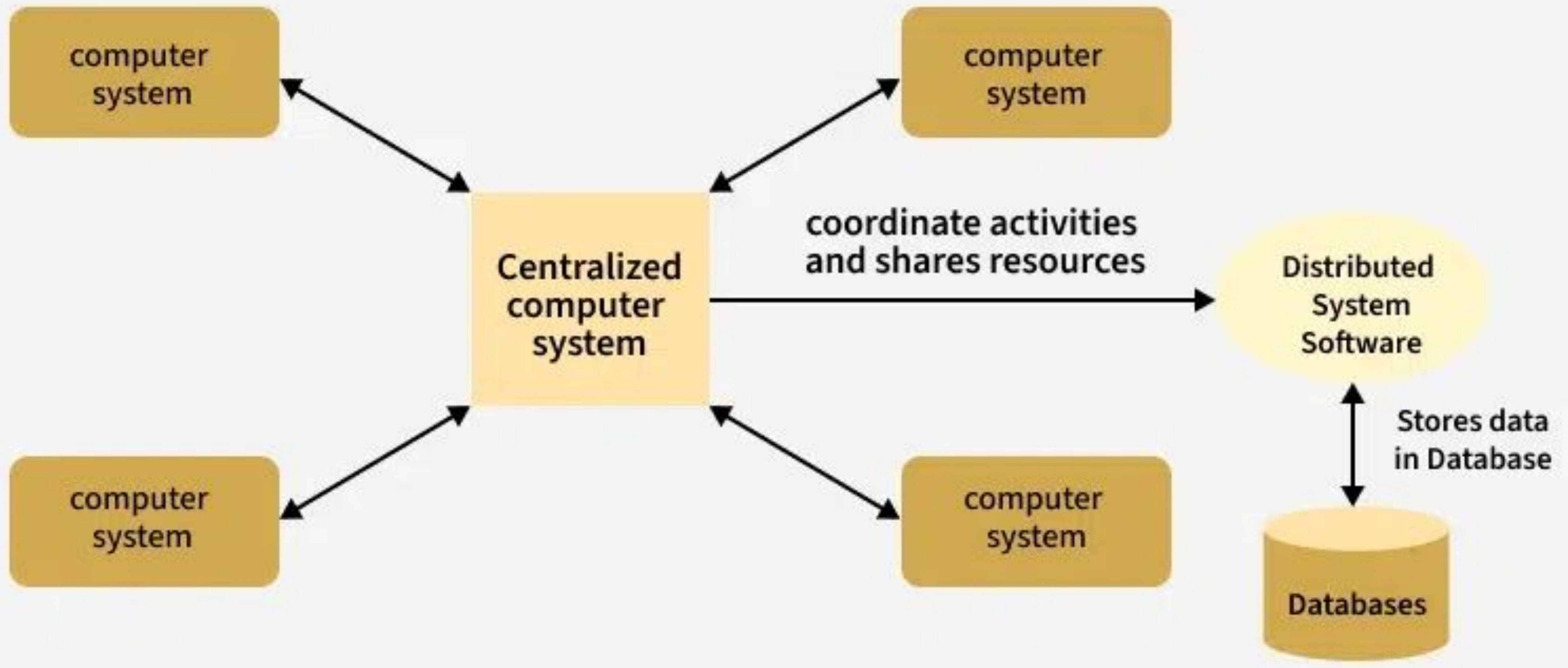
- ***A collection of copies is consistent when the copies are always the same.***
- This means that a read operation performed at any copy will always return the same result.
- Consequently, when an update operation is performed on one copy, the update should be propagated to all copies before a subsequent operation takes place.

- This type of consistency is sometimes informally referred to as **tight consistency or synchronous replication.**

- The key idea is that an update is performed at all copies as a single atomic operation, or transaction.

- we need to synchronize all replicas.

What are Consistency models in distributed systems?


computer system
computer system
Centralized computer system
coordinate activities and shares resources
Distributed System Software
Stores data in Database
Databases
computer system
computer system

Data Consistency

- **Data consistency** is the process of keeping information **uniform** as it moves across a network and between various applications on a computer.

Consistency Model

Consistency models define the rules and guarantees regarding the order and visibility of updates in distributed systems.

They ensure that clients or users perceive a coherent state of the system despite its distributed nature.

Data centric consistency model

- Traditionally, **consistency has been discussed in the context of read and write operations on shared data,** available by means of (distributed) shared memory.

- 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.

- Write operations are propagated to the other copies.

- A data operation is classified as a write operation when it changes the data, and is otherwise classified as a read operation.

1. Strict(Strong) Consistency Model

In a strongly consistent system, all nodes in the system agree on the order in which operations occurred.

Reads will always return the **most recent version of the data**.

When an update occurs on one server, this model makes sure every other server in the system reflects this change immediately.

This model provides the highest level of consistency, but it can be slower and require more resources in a distributed environment since all servers must stay perfectly in sync.

Example :

When operations rely on the most recent data and consistency is crucial, like in banking systems or inventory management, it is important.

ii.Sequential Consistency

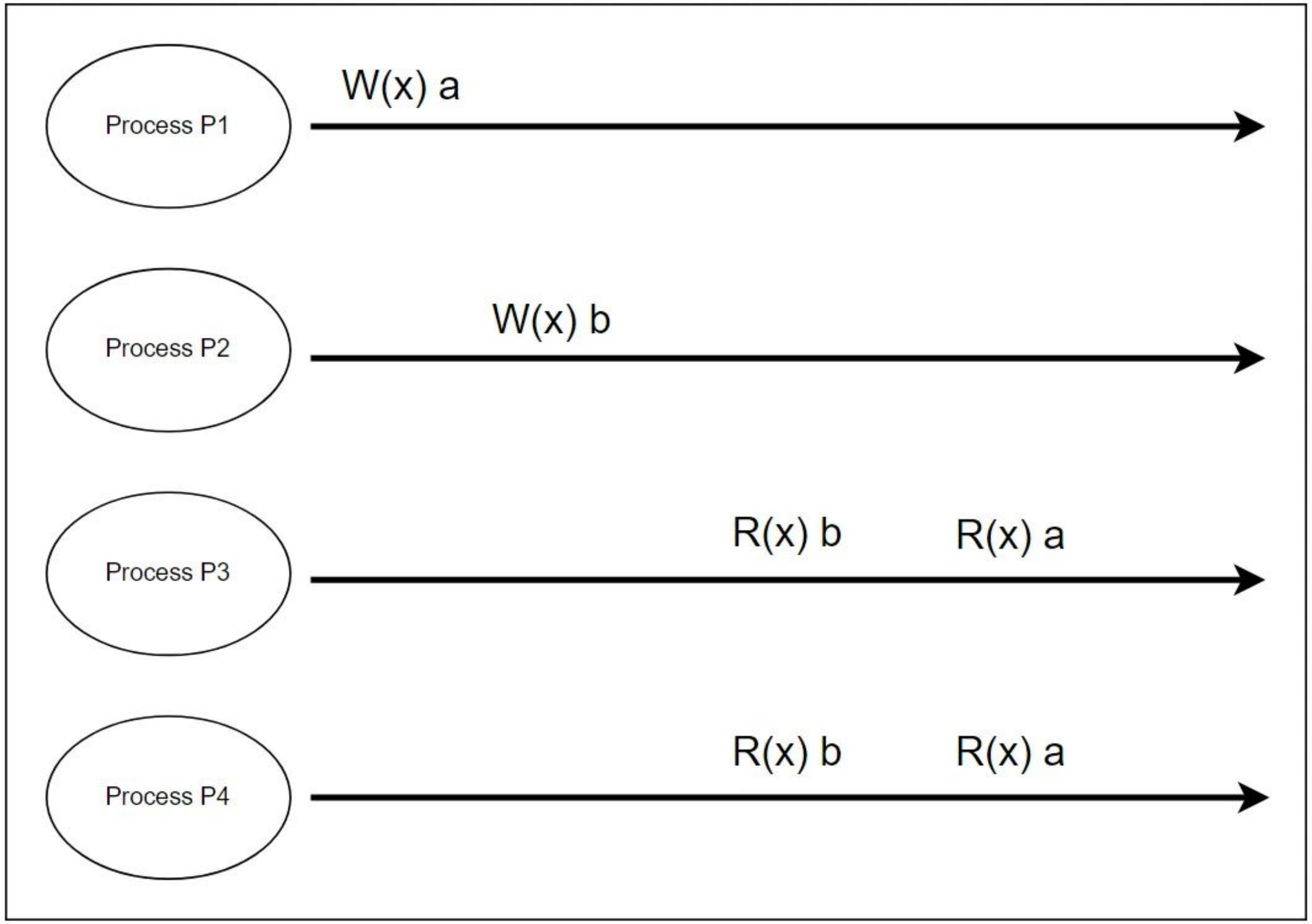
- It is a consistency model in distributed systems that ensures all operations across processes appear in a single, unified order.
- In this model, every read and write operation from any process appears to happen in sequence, regardless of where it occurs in the system.
- Importantly, all processes observe this same sequence of operations, maintaining a sense of consistency and order across the system.

Opt for sequential consistency when the order of operations matters but a perfect global order isn't necessary.

Real-World Example: Distributed Database

- Imagine a distributed database with multiple replicas (copies) of the data.
- If client A writes a value to the database, and then client B reads the value, sequential consistency ensures that client B will see the write from client A in the same order as it happened.
    -

sequential consistency



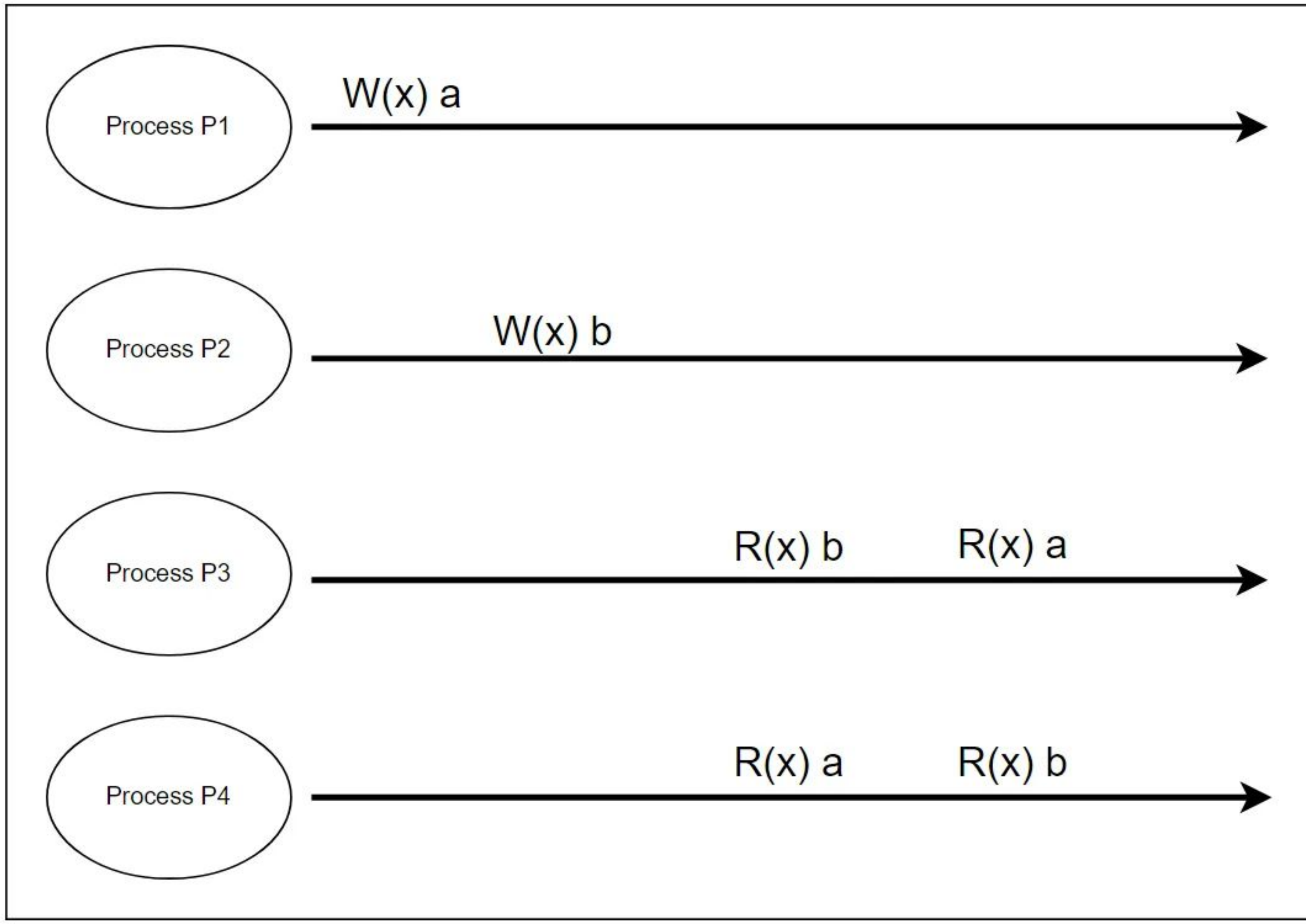
Let us consider them data nodes acting on some clients' requests.

This interleaving is sequentially consistent and both $P3$ and $P4$ see the same ordering of events.

$W(x)a$ means that a process is writing the value a to a variable x.

$R(x)a$ means that a process did a read operation on x and got a value a back.

non sequential consistency



let us see an example that is not sequentially consistent using the following illustration.

Note that process $P3$ reads as x=b and then x=a as a final value. Process $P4$ reads x as a and then b as the final value.

That means if no more updates are coming, $P3$ and $P4$ have divergent values of x.

As explained earlier, each process should have seen the same total ordering to be sequentially consistent.

iii. Causal Consistency

The Causal Consistency Model is a type of consistency in distributed systems that ensures that related events happen in a logical order.

In simpler terms, if two operations are causally related (like one action causing another), the system will make sure they are seen in that order by all users.

However, if there's no clear relationship between two operations, the system doesn't enforce an order, meaning different users might see the operations in different sequences.

- It is a weaker model than sequential consistency.
- (one process follow other)
- causal consistency means , if the first operation is influenced by the second operation.
- If a write(w2) operation is causally related to another write (w1) the acceptable order is (w1, w2).

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**.

i.

Eventual Consistency

- In Systems that tolerate high degree of inconsistency, if no updates take place for a long time all replicas will gradually and eventually become consistent.

- This form of consistency is called eventual consistency.

- Eventual consistent data stores work fine as long as clients always access the same replica.

- Write conflicts are often relatively easy to solve when assuming that only a small group of processes can perform updates.

- Eventual consistency is therefore often cheap to implement.

ii. Monotonic Reads and Writes

Once a piece of data has been read or written, monotonic reads and writes guarantee that the data will always be viewed in a predictable and consistent order in subsequent reads or writes.

- **Monotonic Reads**: If you read a value from a system, the next time you read it, you will either get the same value or a more recent one. You won't get an older value after seeing a newer one.
- **Monotonic Writes**: This ensures that once a write happens, all future writes will follow in the correct order. If you update a record or send a message, the system guarantees that it won't reverse the order of your updates.

iii. Read Your Writes

- states that the system guarantees that, once an item has been updated, any attempt to read the record by the same client will return the updated value.

- A write operation is always completed before a successive read operation by the same process no matter where that read operation takes place.
- Example: Updating a Web page and guaranteeing that the Web browser shows the newest version instead of its cached copy.

iv. Writes Follow Reads

- A data store is said to provide writes-follow-reads consistency:

  – if a process has write operation on a data item x following a previous read operation on x

  –then it is guaranteed to take place on the same or a more recent value of x that was read.

-

- Example: Suppose a user first reads an article A then posts a response B. By requiring writes-follow-reads consistency, B will be written to any copy only after A has been written.

Replica Management

- A key issue for any distributed system that supports replication is to decide **where , when and by whom replicas should be placed and subsequently which mechanism to use for keeping the replicas consistent.**
- The placement problem itself should be split into two subproblems: -that of placing replicas servers and -

  -that of placing content.

Where to place replica ?

- When it comes to content and placement , three different types of replicas can be distinguished logically organized as shown :

Content Replication and Placement

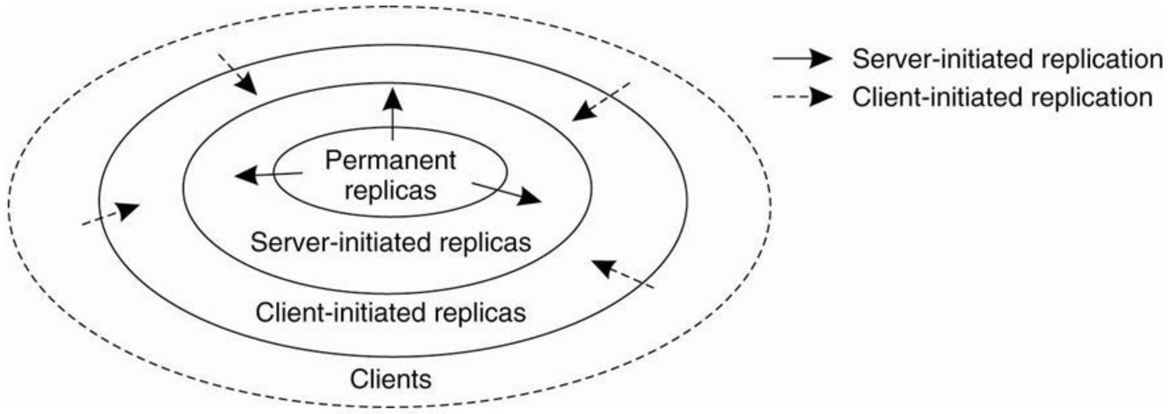

Server-initiated replication
Client-initiated replication
Permanent replicas
Server-initiated replicas
Client-initiated replicas
Clients

Figure 7-17. The logical organization of different kinds of copies of a data store into three concentric rings.

1. Permanent Replicas: Process/Machine always have a replica. In many cases , the number of permanent replicas is small.

Eg: a website.

2. Server-Initiated Replicas: Process that can dynamically host a replica on request of another server in the data store. Eg: a web server placed in kathmandu .

- Normally this server can handle incoming request quite easily, but it may happen that over a couple of days a sudden burst of request come in from an unexpected location for 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 Initiated 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 store a copy of the data it has just requested, In principle , managing the cache is left entirely to the client.

When Should Replicas Be Created or Updated?

At Deployment (Initial Replica Creation):

On Demand (Dynamic Replication):

Real-time Updates (Replication of Writes):

On Failure or Recovery:

By Whom Should Replicas Be Placed?

System Administrators (Manual Placement):

Distributed System Algorithms (Automated Placement):

Distributed Consensus Protocols (e.g., Paxos, Raft):

Cloud Providers and Managed Services:

Content Distribution

- Replica management also deals with propagation of (updated) content to the relevant replica server.

- content is duplicated and stored on multiple servers (replicas) :
    - to improve availability,
    - reduce load on the primary server, and
    - enhance performance by allowing content to be served from the closest replica to the user.

Any important design issue concerns what is actually to be propagated . Basically there are three possibilities:

1. Propagate only a notification of an update (often used for caches)

2. Transfer data from one copy to another(Distributed Database)

3. Propagate the update operation to other copies(Also called active replication)

Pull Vs Push Protocols

- Another design issue relates to whether or not the updates are pushed or pulled?
- Push-based/Server-based Approach: sent “automatically” by server, the client does not request the update.
- This approach is useful when a high degree of consistency is needed. Often used between permanent and server-initiated replicas.
- Pull-based/Client-based Approach: used by client caches (e.g., browsers), updates are requested by the client from the server. No request, no update!

Consistency protocols

- A consistency protocol describes an implementation of a specific consistency model.
- Consistency protocols can be divided into two types according to whether data disagreement is allowed or not:

Single master protocol (data disagreement is not allowed)

The whole distributed system is like a monolithic system.

All write operations are handled by the master node and synchronized to other replicas.

Multi-master protocol (allowing data divergence)

All write operations can be initiated by different nodes and synchronized to other replicas.

- Main approaches are as follows:
- 1. Primary- based protocol
- 2. Replicated- write protocol
- 3. Cache-coherence protocol

1. Primary based protocols

- This protocol is typically used in distributed systems, where a "primary" or "leader" node handles all write requests, and the other nodes (often called "replicas" or "followers") only serve read requests.
- In the case of a write request, the primary node updates the data, and the changes are propagated to the secondary nodes to ensure consistency across the system.
- This is common in distributed databases, such as in **master-slave replication** setups.

Characteristics:

- The primary node is the point of control for write operations.
- Replicas receive updates asynchronously or synchronously depending on the configuration.
- Offers simplicity and scalability for read-heavy systems but can become a bottleneck if writes are frequent.

2. Replicated –write protocols

- This protocol is commonly used in systems with multiple replicas, where all replicas must agree on the order of writes to maintain consistency.

- In systems that use this protocol, the data is written to multiple replicas simultaneously or in a specific order to ensure the write is reflected consistently across all nodes.

Common examples include **Quorum-based systems** or **Paxos-based protocols**:

- Write operations are replicated across multiple nodes.
- The system ensures that multiple nodes have the same data after a write, often requiring some form of consensus.
- These protocols provide fault tolerance and high availability because the system can continue operating as long as a majority of replicas are available.

Quorum-Based Protocols

- ✓A different approach to supporting replicated writes is to use voting
- ✓Clients to request and acquire the permission of multiple servers before either reading or writing a replicated data item

How the algorithm works?

1. A file is replicated on *N servers*
2. To update a file: A client must first contact at least half the servers plus one (a majority) and get them to agree to do the update.
3. Once they have agreed, the file is changed and a new version number is associated with the new file.

3. Cache-coherence protocols

- In a multi-core or multi-processor system, a **cache-coherence protocol** ensures that when multiple processors (each with its own local cache) read or write to the same memory location, all caches are kept in sync with each other.
- The goal is to avoid situations where processors operate on stale or inconsistent data, leading to errors or unexpected behavior.

Popular cache-coherence protocols include:

- **MESI (Modified, Exclusive, Shared, Invalid)**: One of the most well-known cache-coherence protocols that defines states for cache lines to ensure consistency across different processors' caches.

- **MOESI** (Modified, Owner, Exclusive, Shared, Invalid): An extension of MESI that adds an "Owner" state for better efficiency in systems with high write contention.

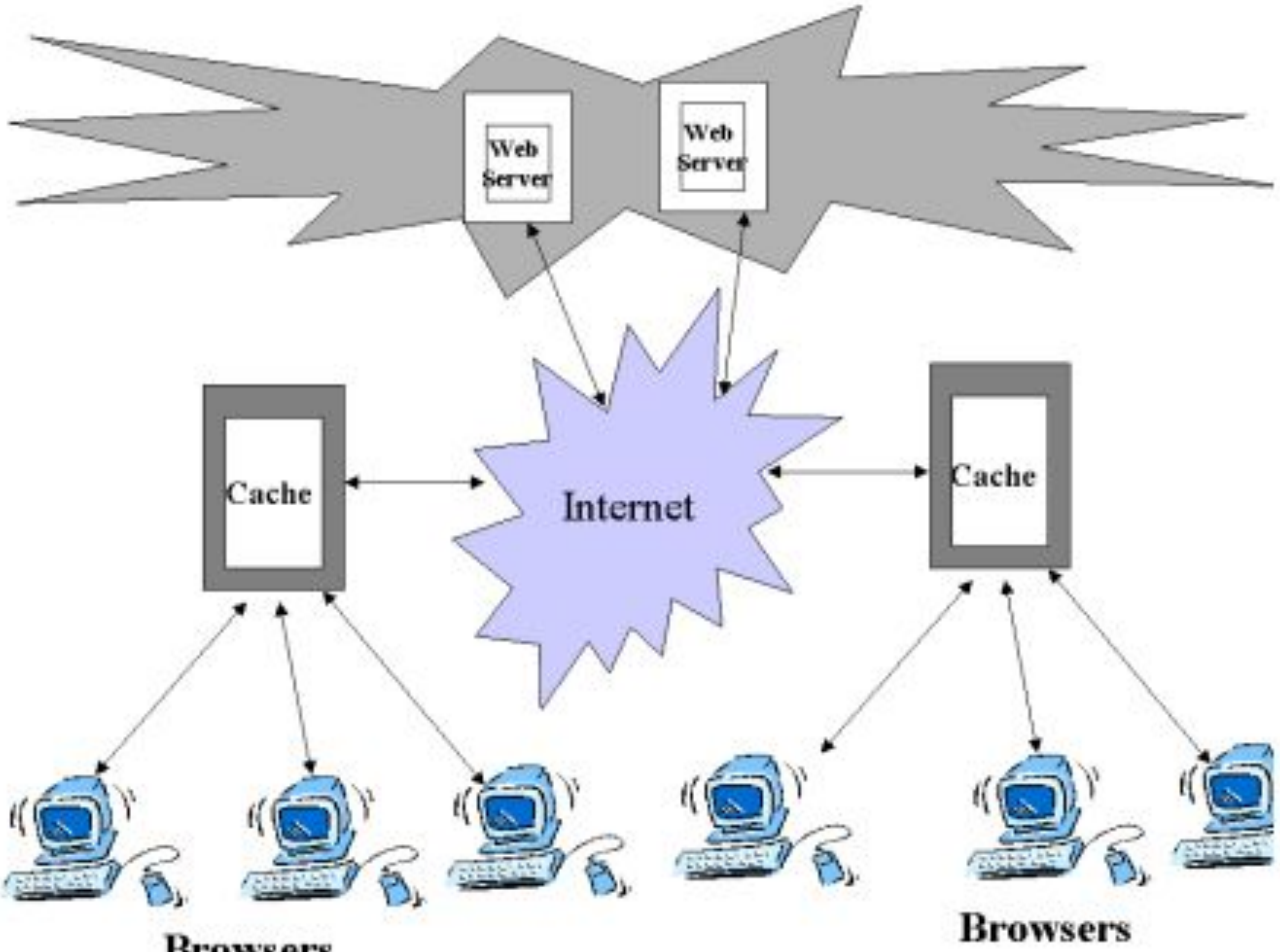
Characteristics:

- Ensures that all caches have a consistent view of the memory.
- Reduces the chance of race conditions and errors in multi-core systems.
- Can have an impact on performance, especially when there are frequent memory writes across processors.

Web Cache

- A cache is a temporary storage location for copied information . There are over a billion pages(or objects) on the internet.
- Many users request the same popular objects .
- An example of that would be the top logo image of Yahoo.com which appears in almost all Yahoo pages .
- The image must be delivered to the browser each time the browser accesses any of Yahoo's pages an these pages are requested a number of times each day by different users .

- A Web cache is a dedicated computer system which will monitor the object requests and stores objects as it retrieves them from the server .
- On subsequent requests **the cache will deliver objects from its storage rather than passing the request to the origin server .**
- Every Web object changes over time and therefore has a useful life or "freshness" .
- If the freshness of an object expires it is the responsibility of the Web cache to get the new version of the object .
- The more the number of requests for the same object the more effective will the Web cache be in reducing upstream traffic and will also help reducing server load, resulting in less latency.


Web Server
Web Server
Cache
Internet
Cache
Browsers
Browsers