Unit 7 : Consistency and replication

7.1 Introduction

- In the distributed system, data is **duplicated** mainly for “reliability” and “performance”.
- Replication is required, especially if the distributed system needs to grow in quantity and geographically.
- Replication is such one of the scaling techniques. Also, it can cope with data corruption and replica crash.
- At this time, it is necessary to maintain the consistency of the state of the data and the replica. However, this directly leads to **scalability** problems.
- The ideal when you think about consistency is “To Keep All Replicas In Exactly The Same State And Operation”, **that is, atomicity is implemented**, but it’s quite a challenge.

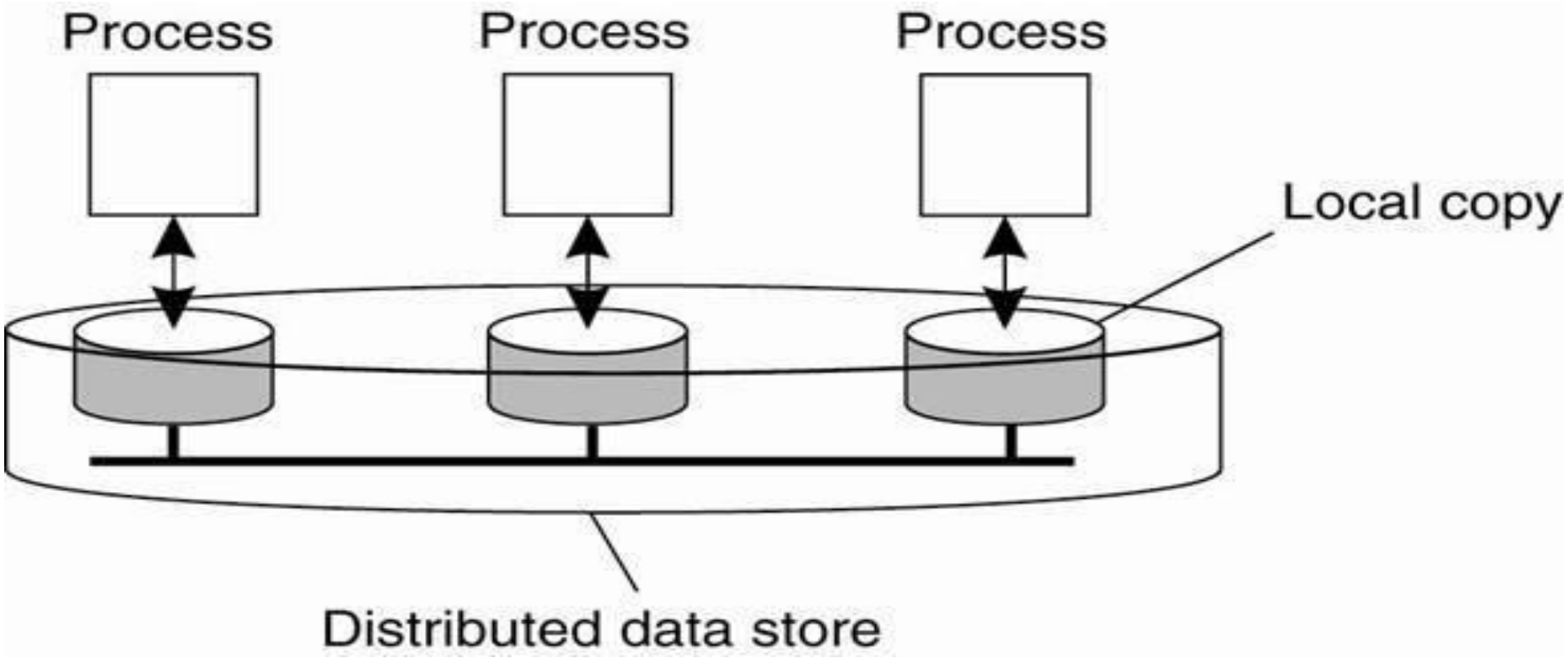
- In reality, we can not solve the problem of scalability unless we give up some consistency constraints.
- Thus, it is necessary to understand **how much consistency is required in the system**, and **how to realize its extent of consistency**.

7.2 Data-centric consistency model

Traditionally, consistency has been discussed mainly from data (data stores).

In a distributed system, each process holds its own local copy. At this time, a collection of stored data including each local copy is called a distributed data store.

When the process reads from a data store, it expects the result of the last write operation to be returned as data. In order to clarify which is the last write, we call **ordering relation** as consistency.


Process
Process
Process
Local copy
Distributed data store

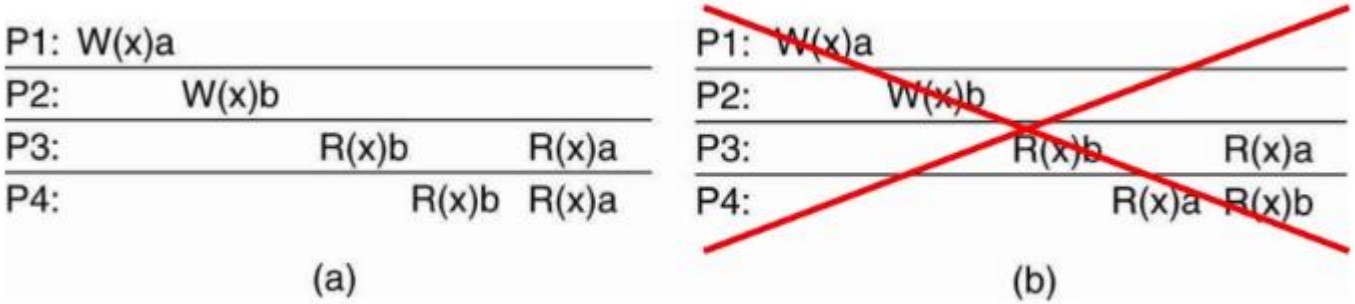
- It is nearly impossible to maintain complete consistency.
- Allowing some extent of inconsistency leads compatibility with performance, but it depends on your application that what you can tolerate and how much you can tolerate.
- You should choose a level of consistency according to your application, system.
- Therefore, it is necessary to understand first what kind of consistencies there are.

Sequential consistency is the most popular and important consistency model. There is **Casual consistency** as its weak variant.

Sequential Consistency

When the following conditions are satisfied, the data store is Sequentially consistent.

- The result of any execution sequence is that the read / write operations by all processes to the data store are the same as the results executed in a certain sequential specific order, and the operations of the individual processes are executed in the order specified by the process.
- The point is that **they are in the exactly same order** when comparing the operation of each process.
- In the following figures, (a) is sequentially consistent, but (b) is not because the order of reading in Process3 and Process4 is different.
- Specifically, in Process3, it is supposed to change from the value R(x)b of the write result by Process2 to the value R(x)a by P1, but it is the reverse in P4.



W(x)a represents the writing of the value “a” for the data “x” and R(x)b represents the reading of the value “b” from the data “x”.

Casual Consistency

When the following conditions are satisfied, the data store is Casually consistent.

- Potentially causally related writes must be observed in the same order by all processes. For different machines, concurrent writes may be observed in different orders.
- In other words, it reduces the constraints of sequential consistency, and **processing sequence without causality can be in different orders**.
- At this time, it is important each operation depends on which operations (whether two of the operations have causal relationship or not).

In the following figure, in the above one, since writing of the value “b” and the value “c” is a parallel operation, it satisfies **Causal consistency**.

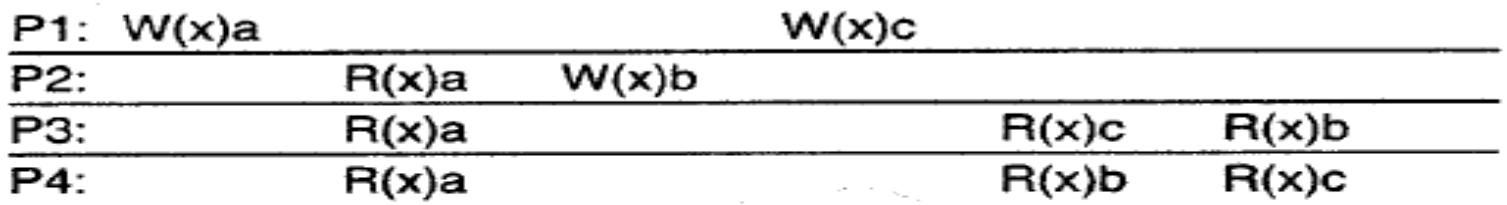
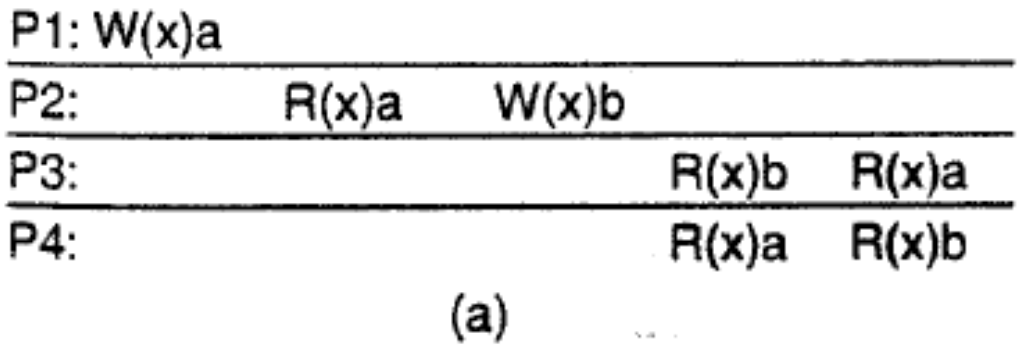
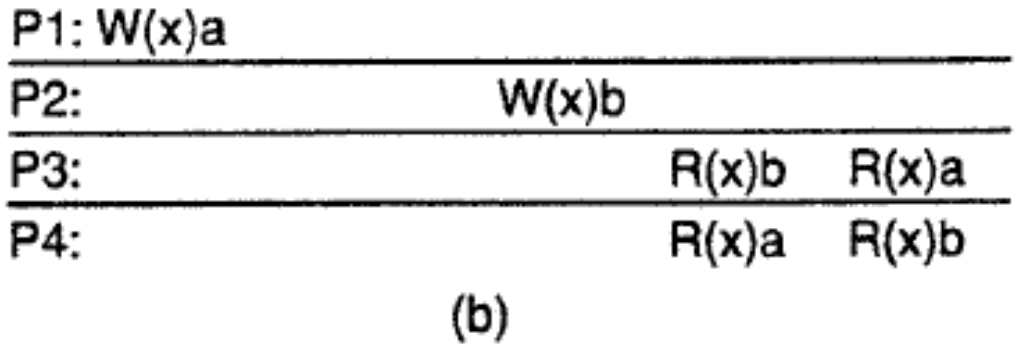


Figure 7-8. This sequence is allowed with a causally-consistent store, but not with a sequentially consistent store.



(a)



(b)

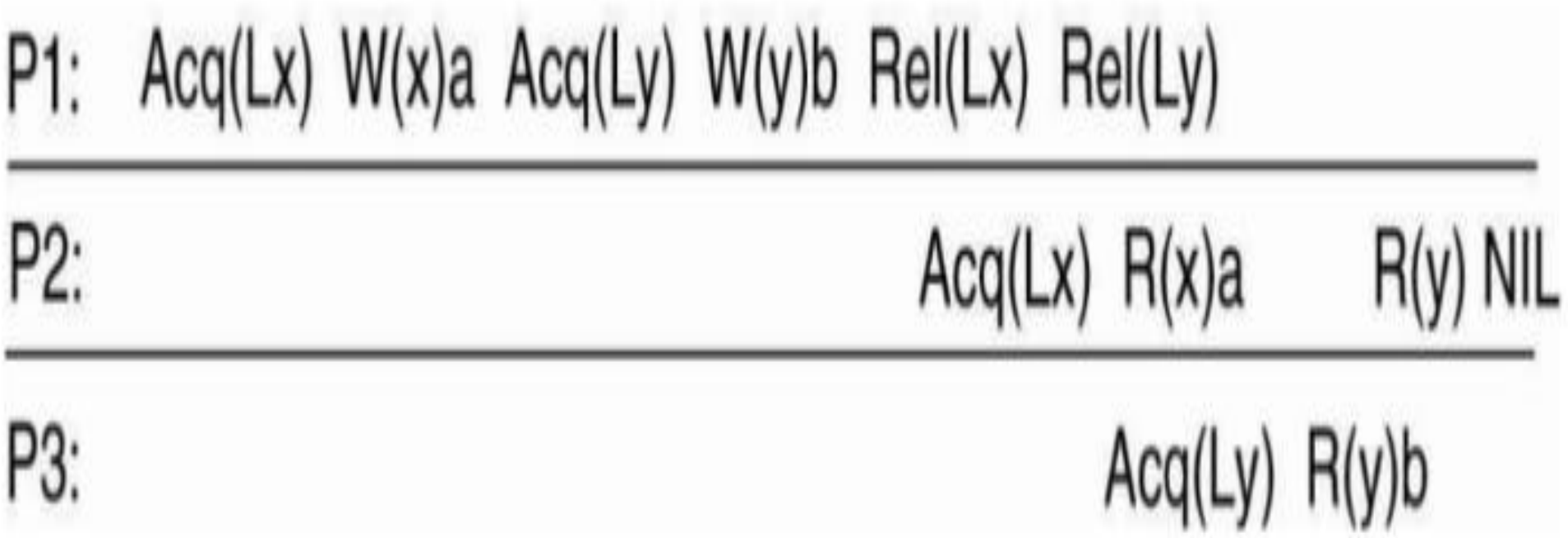
Figure 7-9. (a) A violation of a causally-consistent store. (b) A correct sequence of events in a causally-consistent store.

Entry Consistency

- First, prepare synchronization variables. It is also possible to maintain consistency by acquiring synchronization variables and allowing only the process owing its synchronization variable to update the data. This is called “Entry Consistency”.
- For entry consistency to be preserved, there should not be two owners of synchronous variables at exclusive mode access such as writing.
- In a non-exclusive mode that can be read but not written, it is possible for multiple processes to own synchronous variables at the same time.

In the following figures, since Process2 does not hold the access right (= synchronous variable) to the data item “y”, the reading result becomes NIL.

- Fig. below shows an example of what is known as entry consistency.
- Instead of operating on the entire shared data, in this example we associate locks with each data item.
- In this case, *P* I does an acquire for *x,* changes *x* once, after which it also does an acquire for *y.*
- Process *P2* does an acquire for *x* but not for *Y'.so* that it will read value *a* for *x,* but may read *NIL* for *y.*
- Because process *P3* first does an acquire for *y,* it will read the value *b* when *y* is released by *Pl'*



At this time, how to properly correlate data and synchronization variables becomes a problem. In the case of object-oriented, it can be realized by implicitly associating a synchronization variable with each object.

7.3 Client-centric consistency model

Eventual consistency and client-centric consistency model

- In general, inconsistencies between processes are acceptable to some extent.
- For example, in a web cache, cached pages responded to a client may be older than the actual version of the web server actually exists.
- However, for many users, such inconsistencies are acceptable to some extent.

- Even if there is inconsistency as described above, all replicas will gradually become consistent as time goes by. Consistency of this form is called **eventual consistency**.
- Eventually consistent data store operate normally only if the client always accesses the same replica.
- However, what if there are mobile users who move and access different replicas in a short time?
- For example, if you are on express, the client will move to another location in a short period of time and access different replicas, but here it seems inconsistent unless previously done updates have already been propagated.

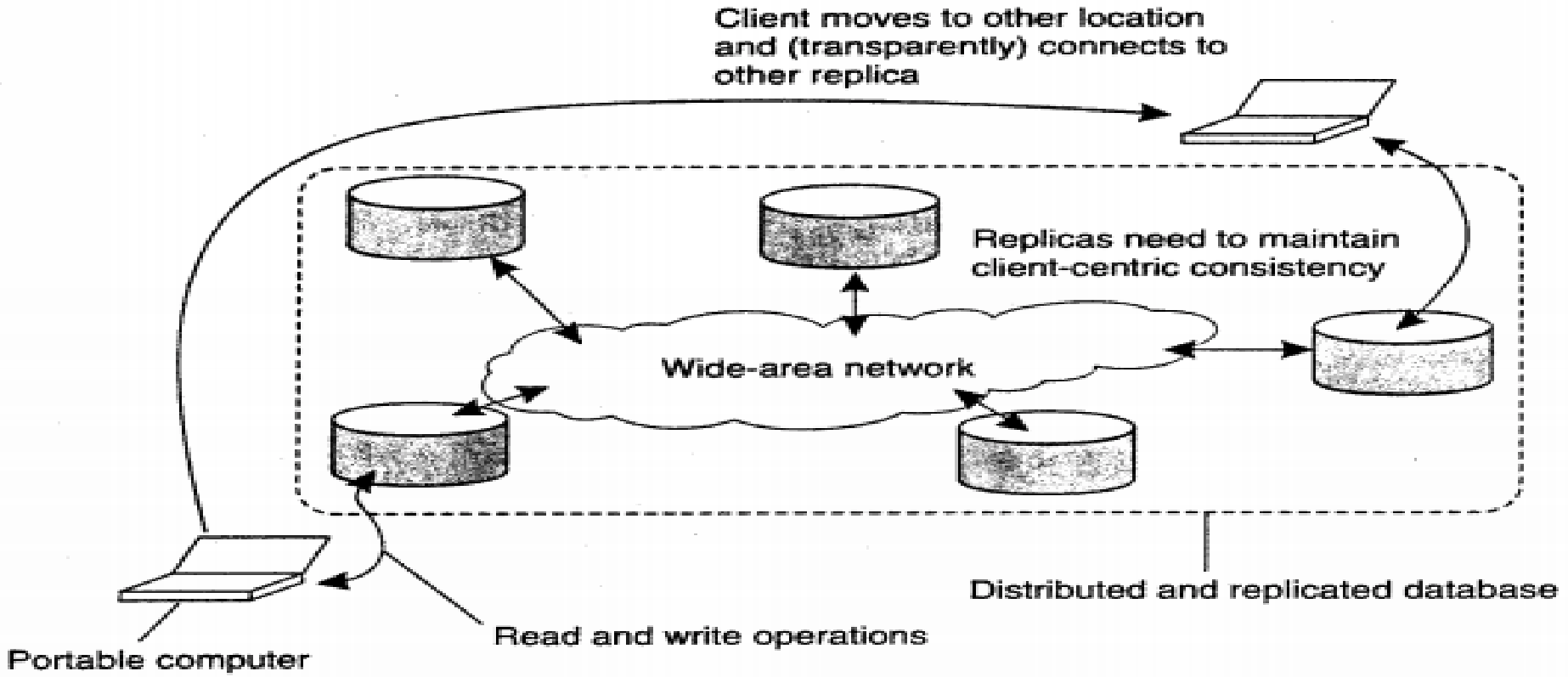


Figure '-11. The principle of a mobile user accessing different replicas of a distributed database.

As for the above problem, introduction of **client centric consistency** is a solution. Unlike the data-centric consistency model in the previous section, this model aims to guarantee consistency for a single client.

Client-centric consistency has several patterns as follows.

- Monotonic reading
- Monotonic writing
- Read after writing
- Write after reading

Monotonic reading

The following sentences are the conditions that satisfy monotonic reading consistency.

- *If a process reads data item x, any subsequent reads on x by that process will either reply with the same value or reply with a newer value.*

In the following figures, in (b) there is no guarantee that the contents of the write operation WS(x1) is included in the reading result of L2, so there is no monotonic read consistency

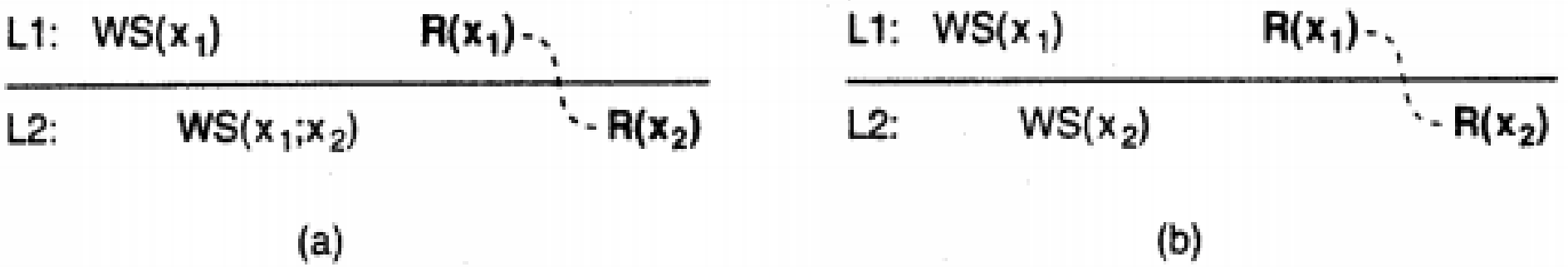


Figure 7-12. The read operations performed by a single process P at two different local copies of the same data store. (a) A monotonic-read consistent datu store. (b) A data store that does not provide monotonic reads.

•Whenever you connect to a mail server anywhere, it is guaranteed that all mails that you can read when you access the server up to the last time can be read at the destination.

Monotonic writes

The following sentences are the conditions for satisfying monotonic writing consistency.

A write operation by one process to a data item x is completed before any subsequent write to x by the same process.

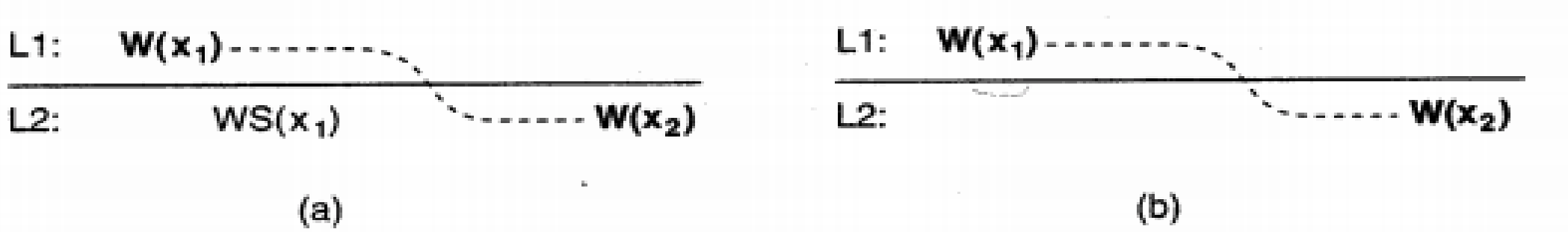


Figure 7-13. The write operations performed by a single process P at two different local copies of the same data store. (a) A monotonic-write consistent data store. (b) A data store that does not provide monotonic-write consistency.

- Monotonic write consistency is similar to data-centered FIFO consistency.
- The essence of FIFO consistency is that write operations by the same process are done in the correct order everywhere.
- In monotonic writing consistency, it is the same order constraint, but only a single process, not a concurrent process collection, is targeted.

Read Your Writes

The following sentences are the conditions that satisfy read consistency after writing.

The result of a write operation by a process to data item x is always observed by subsequent read operations by the same process.

That is, a write operation is always completed before a subsequent read operation performed by the same process, no matter where it is done.

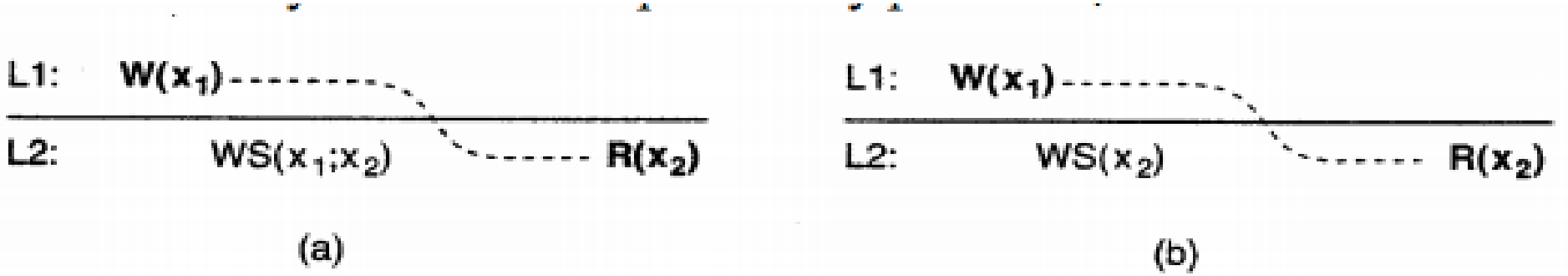


Figure 7-14. (a) A data store that provides read-your-writes consistency. (b) A data store that does not.

Example 1: After updating a web page, new browser content will always be displayed in your browser.

Example 2: When password is updated, guarantee that updated password can be used at the place you moved to.

Writes Follow Reads

The following sentences are the conditions for satisfying write consistency after reading.

The result of a write operation by a process to data item x is always observed by subsequent read operations by the same process.

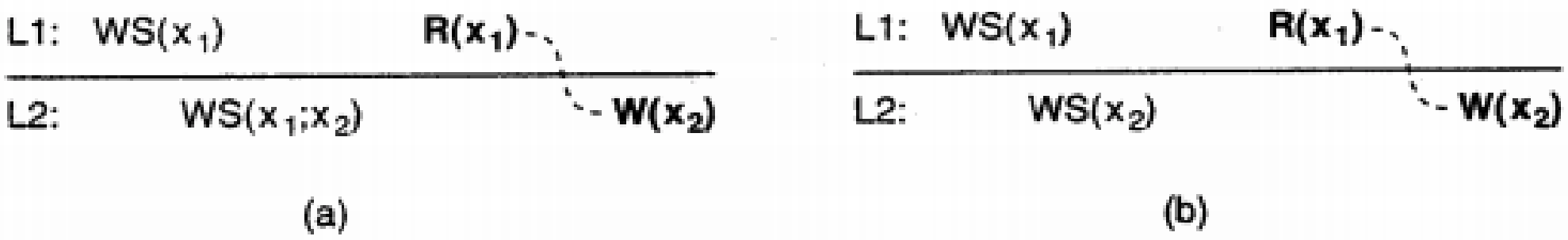


Figure 7-15. (a) A writes-follow-reads consistent data store. (b) A data store that does not provide writes-follow-reads consistency.

7.4 Replica Management

Up to now, we have explained the types of consistency in detail. In this section, we will explain **replication arrangement**, where, when, when and how the replica is placed, and how to achieve consistency.

There are two types of replication placement:

- Replica server placement problem
- Contents placement problem.

1. **Replica Server Placement Problem**

- As for the server placement problem, there is one way to arrange it based on the distance between the client and the server.
- Distance can be measured by transmission delay and bandwidth. It is good to select one server so that the average distance between server and client is minimized.
- As an alternative, there is a way of considering the network topology. It is good to place the server on a router with the maximum number of network interfaces (i.e. links).

Problems of them are large quantity of computational complexity. It can not tolerate the calculation at the time of a “flash cloud” (a sudden explosive request for a certain site). At this time, there are some solutions to increase efficiency such as finding areas with the most access by delimiting areas.

2. **Content duplication and placement**

Regarding the contents, three different types of replicas are distinguished and organized as shown in the figure below.

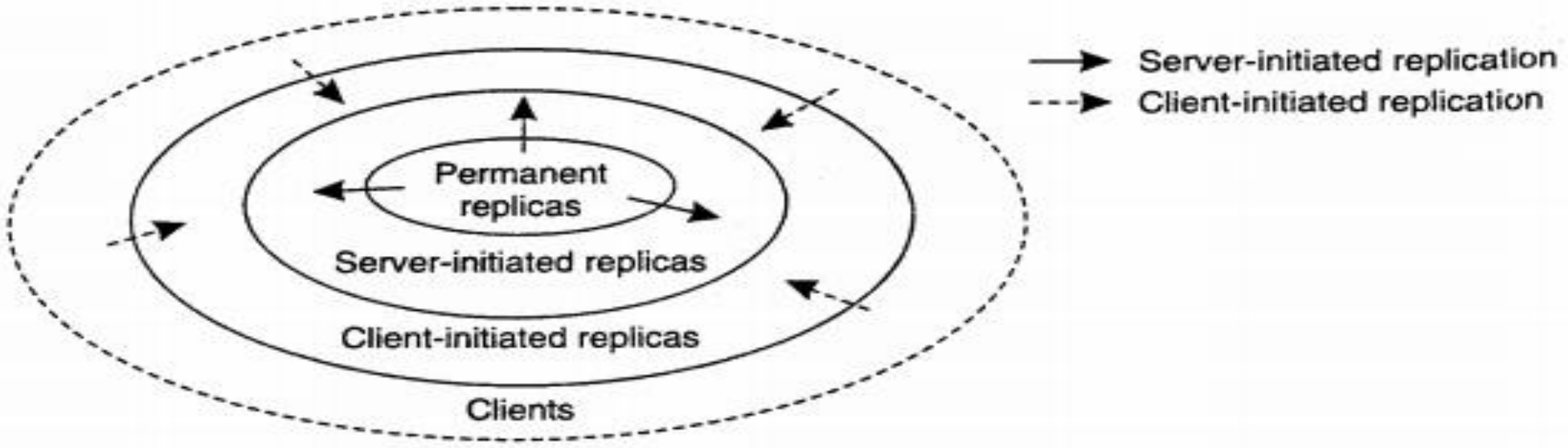


Figure 7-17. The logical organization of different kinds of copies of a data store into three concentric rings.

Permanent replica

This is the initial set of duplicates that make up the distributed data store. In many cases, the number of permanent replicas is small.

Server Startup Replica

- A copy of the data store that is generated by the launch of the data store owner and used to increase performance. Dynamic placement replication means to dynamically copy files that need to be improved in performance to the server in the web hosting service.

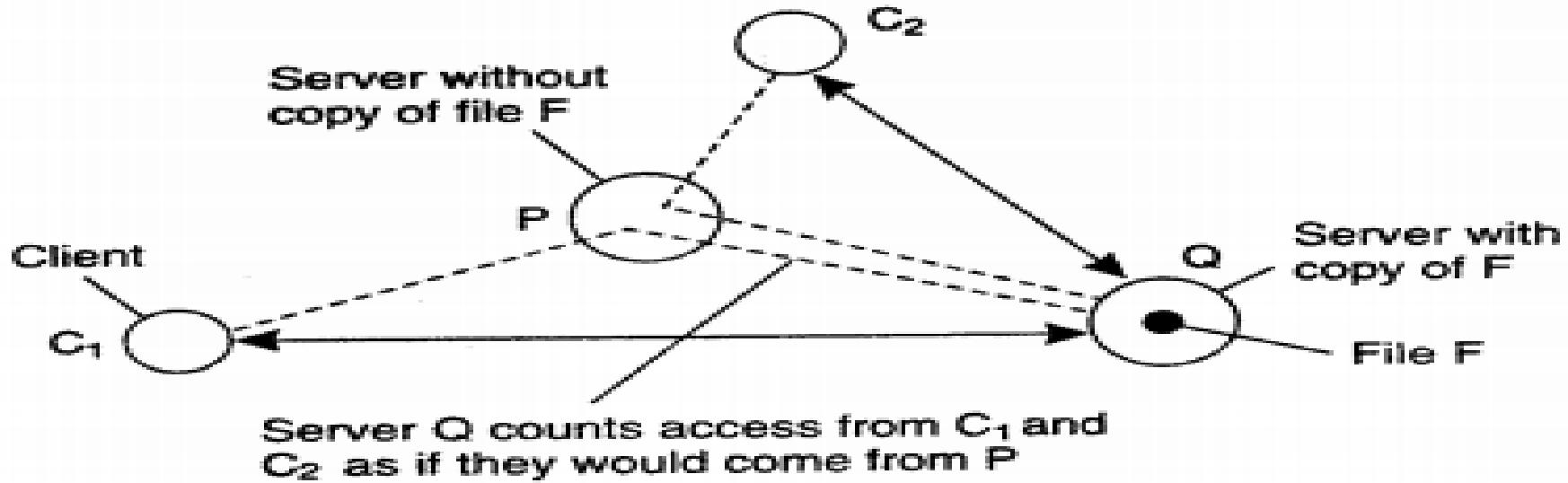


Figure 7-18. Counting access requests from different elients.

For example, to the web server in NewYork, if the number of explosive requests increases over several days by a client at an unexpected location away from the server, it is effective to place duplicates in that area temporarily.

Client Startup Replica

This is commonly referred to as client cache. This is only used to improve the access time to the data. When it is necessary to read the same data, the cache function that holds the replica on the client side only for a limited time is effective.

3. **Content distribution**

Replication management also includes propagation of updated content delivery to the replica server (how to update).

Information type to be propagated

There are three possibilities for information to be actually propagated.

1.Propagate only updates notifications

2.Transmit update data from one copy to another

3.Propagate update operations to other replicas

Propagation of notification of 1 is done by **invalidation notification protocol**. You can only notice that replica information is no longer valid. Advantageously, it uses little network bandwidth, and it is used when replica do not have to update frequently.

- 2 is to transmit the updated data between replicas. Updates tend to be more effective and practical.
- 3 notifies only what kind of update operation should be performed. At this time, it can be assumed that each replica always has the ability to keep the data up to date, so consistency is improved from 1.

Pull vs. Push

There are two methods of update propagation, pull and push.

When the read / write ratio is high, efficiency is better because consistency is maintained when updated by push from the server.

On the other hand, when the read / write ratio is low, high updating frequency is wasteful use of bandwidth etc, so pull is more suitable.

Push and pull have the following features.

Comparison item	Push base update	Pull base update
Server status	Client duplication and cache list	None
Message sent	update (and possibly fetch update later)	polling and updating
Client response time	immediate(or fetch-update time)	fetch-update time

Unicast vs. Multicast

The push-based method can be implemented efficiently by using multicast. In contrast, unicast is the most efficient solution for pull-based methods.

7.5 Consistency Protocol

1. **Primary base protocol**

Primary base protocol is often used for realization of sequential-consistency. In this protocol, the primary server associated with item x performs its write operations responsibly.

Primary base protocol has two ways.

- The way to affirm the primary server to a specific server
- The way to execute a write operation after moving the primary server to the server where the write operation was started

5–1–1. **Remote-write protocol**

All write operations are transferred to a fixed single primary server and executed. In the **Remote Write Protocol** (Primary Backup Protocol), it is a straightforward implementation of sequential-consistency since all writes can be ordered by the primary server in globally unique time order.

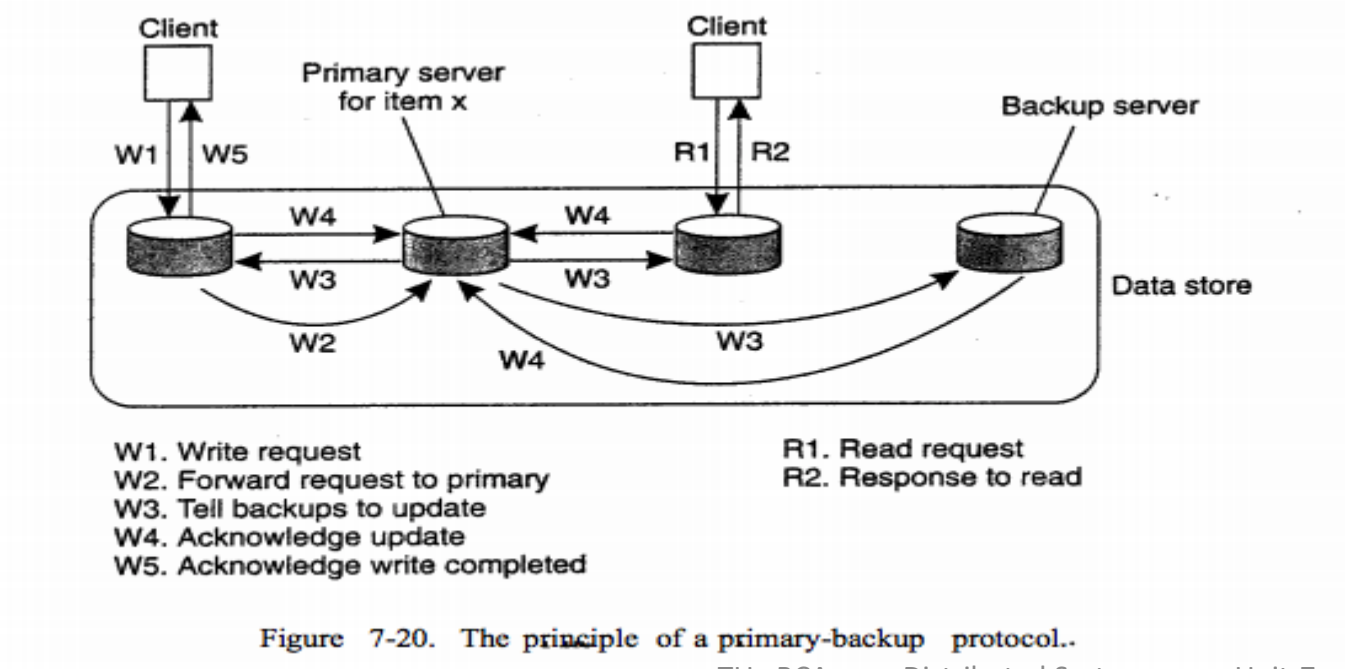


Figure 7-20. The principle of a primary-backup protocol..

However, this protocol has a performance problem of being kept waiting for a relatively long time, which is improved by adopting the nonblocking method, but it is a tradeoff with the reliability of updating.

5–1–2. **Local write protocol**

A protocol that moves the rights so that the server that operates the write operation can operate as a primary replica. It was applied to many distributed memory systems.

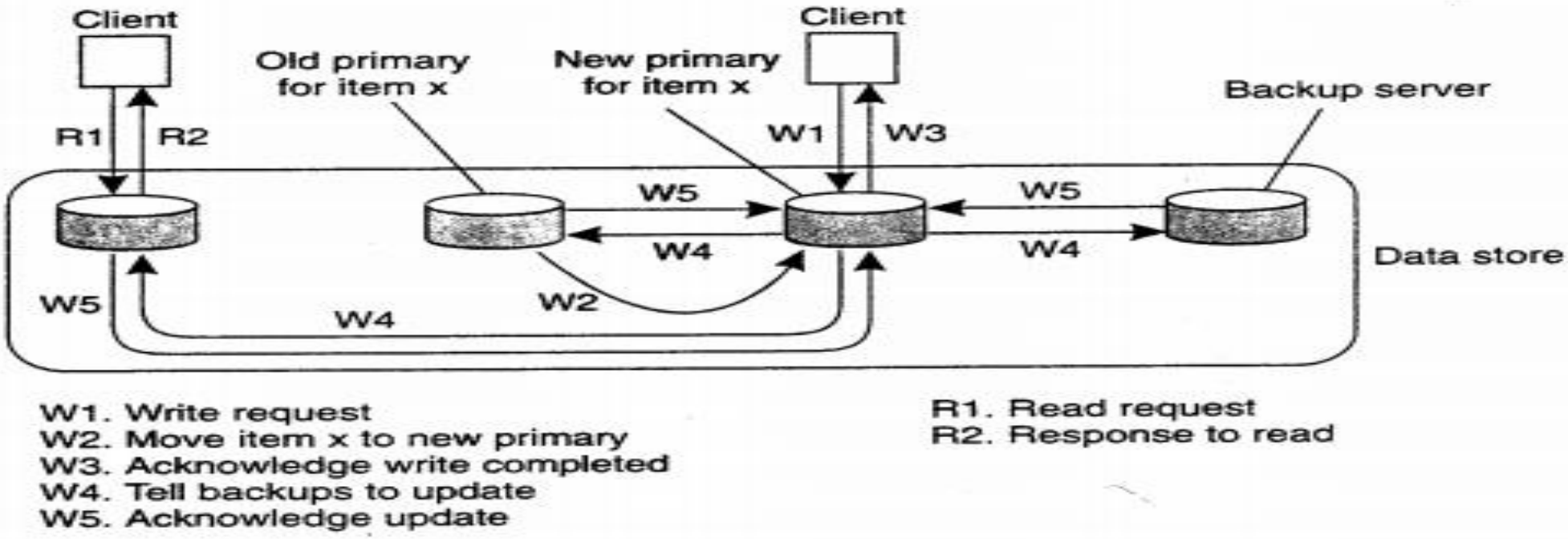


Figure 7-21. Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

2. Replicated-write protocol

In the primary base replication, the write operation is performed on one copy, but in the duplicate write protocol, the write operation is executed for plural copies.

There are two ways for the replicated-write protocol,

- Active replication protocol that transfer operational instructions to all replicas
- Consistency protocol based on majority voting

5–2–1. Active replication protocol

Each copy has a process of executing an update operation, and a write operation is executed. At this time, all replicas must execute operations in the same order.

Ordering can be realized by total order multicast using Lamport's clock, but it is difficult to expand the scale to a large distributed system and it is common to realize total ordering by using a central coordinator or sequencer.

5–2–2. Constant base protocol

This is to realize the write operation duplicated using voting. Basically, before execution of reading and writing of data items, request permission of operation to a plurality of replica servers is made, and execution is performed when permission is obtained.

At this time, let N be the number of servers, let NR be a reading constant and NW be a writing constant, then we have the following constraints:

1. NW + NR> N
2. NW> N / 2

When satisfying these two, it is possible to prevent reading and writing competition and writing competition. The two constants are optimized according to the frequency of each reading / writing.

5–3. Cache Consistency Protocol

Unlike the previous two, consistency is controlled by the client, not the server. Cache consistency protocols can be divided into several design methods.

First, it can be divided as follows depending on when inconsistency is detected.

- The case to Verify consistency when cached data items are accessed during transaction
- The case you want the transaction to proceed while validating the data item
- The case to Verify consistency only when the transaction commits

Next, it can be divided as follows according to the method of maintaining consistency among replicas.

- When updating, with the server always invalidating all caches
- When simply propagating updates

In many cases, the update operation is done only by the server, and updates are propagated by the pull base from the client.
There is also a method similar to the Primary Base Local Write Protocol, which allows the client to modify the cached data. This method is used in a distributed file system and is called write-through cache method. At this time, in order to guarantee sequential-consistency, the client needs to hold exclusive write permission. If you delay the electromagnetic wave of update and allow multiple writes before notifying the server, you can further improve performance, this is called write-back cache.

Name	Contents
Write-through cache	Give permission to write exclusive write to client
Write back cache	Exclusive to client **Give permission to write multiple** writes

7.6 Caching and replication in web

- As the Internet has become an essential part of everyday life, hundreds of millions of users now connect to the Internet. At the same time, more resource-hungry and performance-sensitive applications have emerged.
- Expectations of scalability and performance have made caching and replication common features of the infrastructure of the Web.
- By directing the workload away from possibly overloaded origin Web servers, Web caching and replication address Web performance and scalability from the client side and the server side, respectively.
- Caching stores a copy of data close to the data consumer (e.g., in a Web browser) to allow faster data access than if the content had to be retrieved from the origin server. R
- eplication, on the other hand, creates and maintains distributed copies of content under the control of content providers. This is helpful because client requests can then be sent to the nearest and least-busy server.

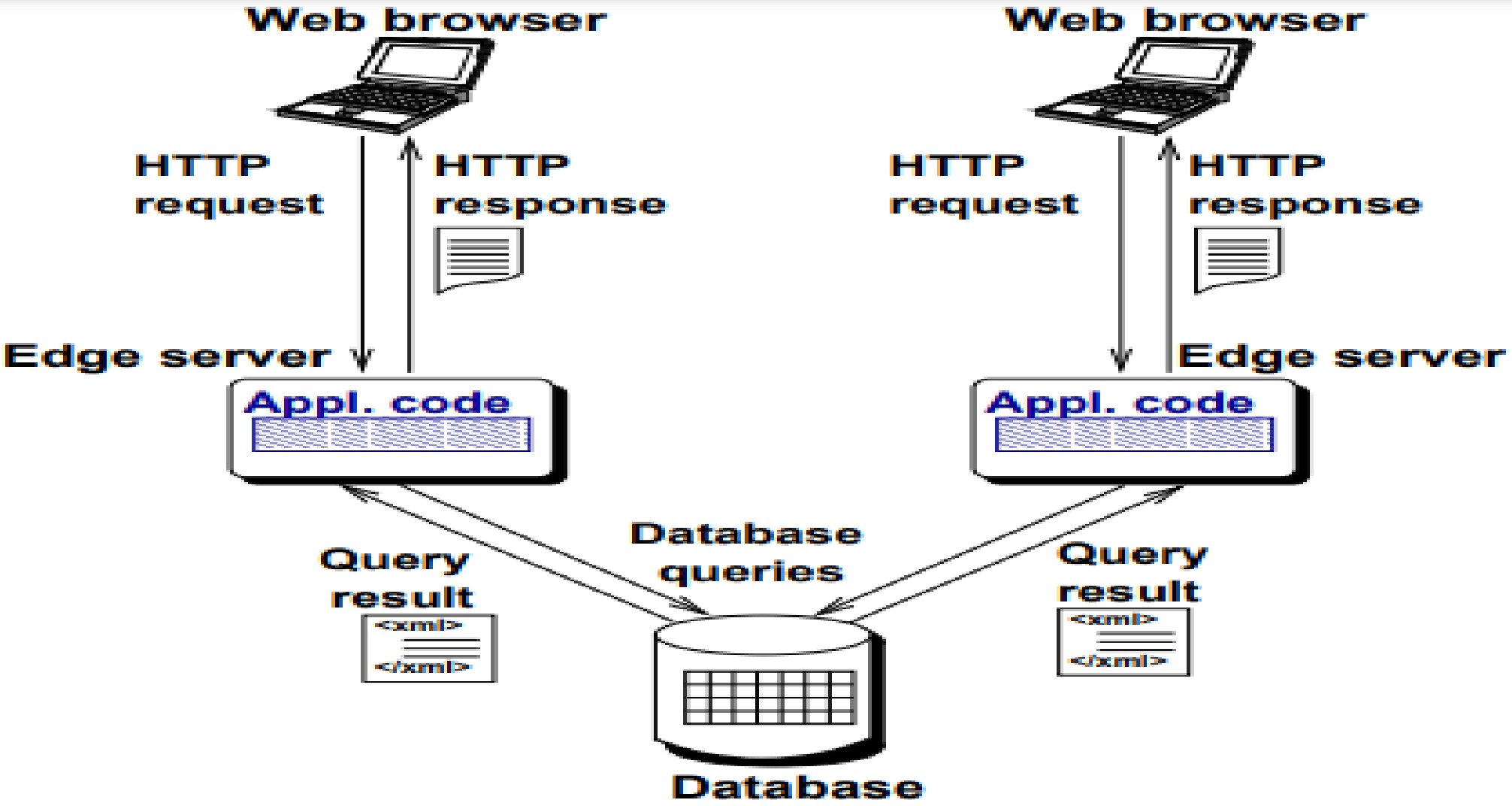
- There are many reasons why Web sites can be slow and an important one is dynamic generation of Web documents.
- Modern Web sites such as Amazon.com and Slashdot.org do not simply deliver static pages but generate content on the fly each time a request is received, so that the pages can be customized for each user.
- Clearly, generating a Web page in response to every request takes more time than simply fetching static HTML pages from disk.
- The main cause is that generating a dynamic Web page typically requires to issue one or more queries to a database.
- Access times to the database can easily get out of hand when the request load is high. A number of techniques have been developed in industry and academia to overcome this problem.
- The most straightforward one is Web page caching where (fragments of) the HTML pages generated by the application are cached for serving future requests.

- For example, Content Delivery Networks (CDNs) like Akamai1 deploy edge servers around the Internet, which locally cache Web pages and deliver them to the clients.
- By delivering pages from edge servers that are usually located close to the client, CDNs reduce the network latency for each request.
- Page caching techniques work well if many requests to the Web site can be answered with the same cached HTML page. These techniques have shown to be effective in hosting many Web sites.
- However, with growing drive towards personalization of Web content, generated pages tend to be unique for every user, thereby reducing the benefits of conventional page caching techniques.

- The limitations of page caching have triggered the CDN and database research community into investigating new approaches for scalable hosting of Web applications.
- These approaches can be broadly classified into four techniques: replicate application code , cache database records , cache query results and replicate the entire database.
- While numerous research efforts have been spent in developing each of these approaches, very few works have analyzed their pros and cons and examined their performance

Edge computing

- The simplest way to generate user-specific pages is to replicate the application code at multiple edge servers, and keep the data centralized (see Figure 1(a)).
- Edge computing (EC) allows each edge server to generate user-specific pages according to context, sessions, and information stored in the database, thereby spreading the computational load across multiple servers.
- However, the centralization of the data can also pose a number of problems.
- First, if edge servers are located worldwide, then each data access incurs wide-area network latency;
- second, the central database quickly becomes a performance bottleneck as it needs to serve all database requests from the whole system.
- These properties restrict the use of EC to Web applications that require relatively few database accesses to generate the content.



(a) Edge Computing