

Unit 7. Consistency and Replication

5 Hrs.

7.1 Introduction

7.2 Data-centric consistency models

7.3 Client-centric consistency models

7.4 Replica management

7.5 Consistency protocols

7.6 Caching and Replication in Web

5 Hrs.

Yuba Raj Devkota | NCCS

Distributed System

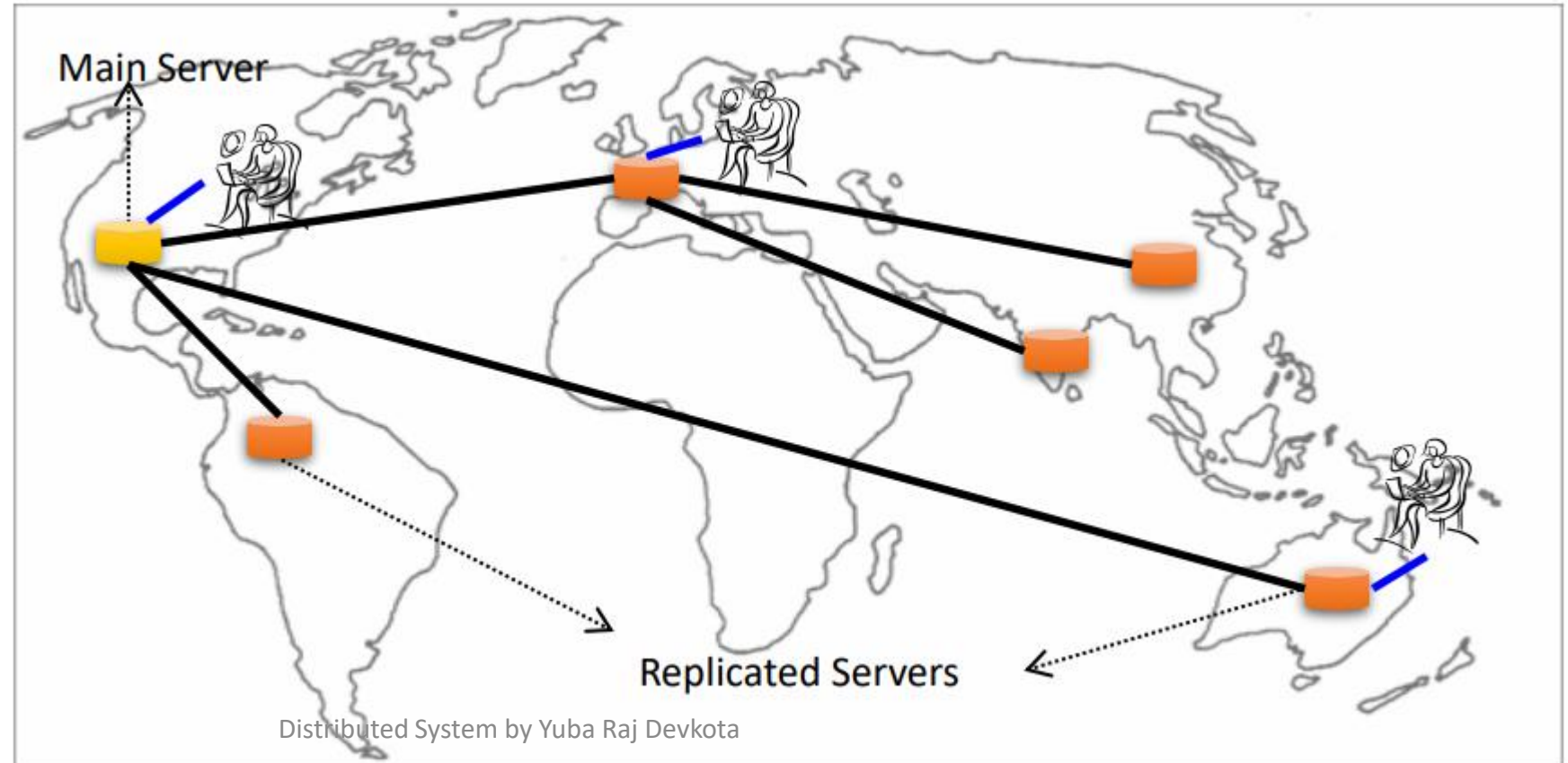
Why Replication?

- Replication is the process of maintaining the data at multiple computers
- Replication is necessary for:
 1. Improving performance
 - A client can access the replicated copy of the data that is near to its location
 2. Increasing the availability of services
 - Replication can mask failures such as server crashes and network disconnection
 3. Enhancing the scalability of the system
 - Requests to the data can be distributed to many servers which contain replicated copies of the data
 4. Securing against malicious attacks
 - Even if some replicas are malicious, secure data can be guaranteed to the client by relying on the replicated copies at the non-compromised servers

1. Replication for Improving Performance

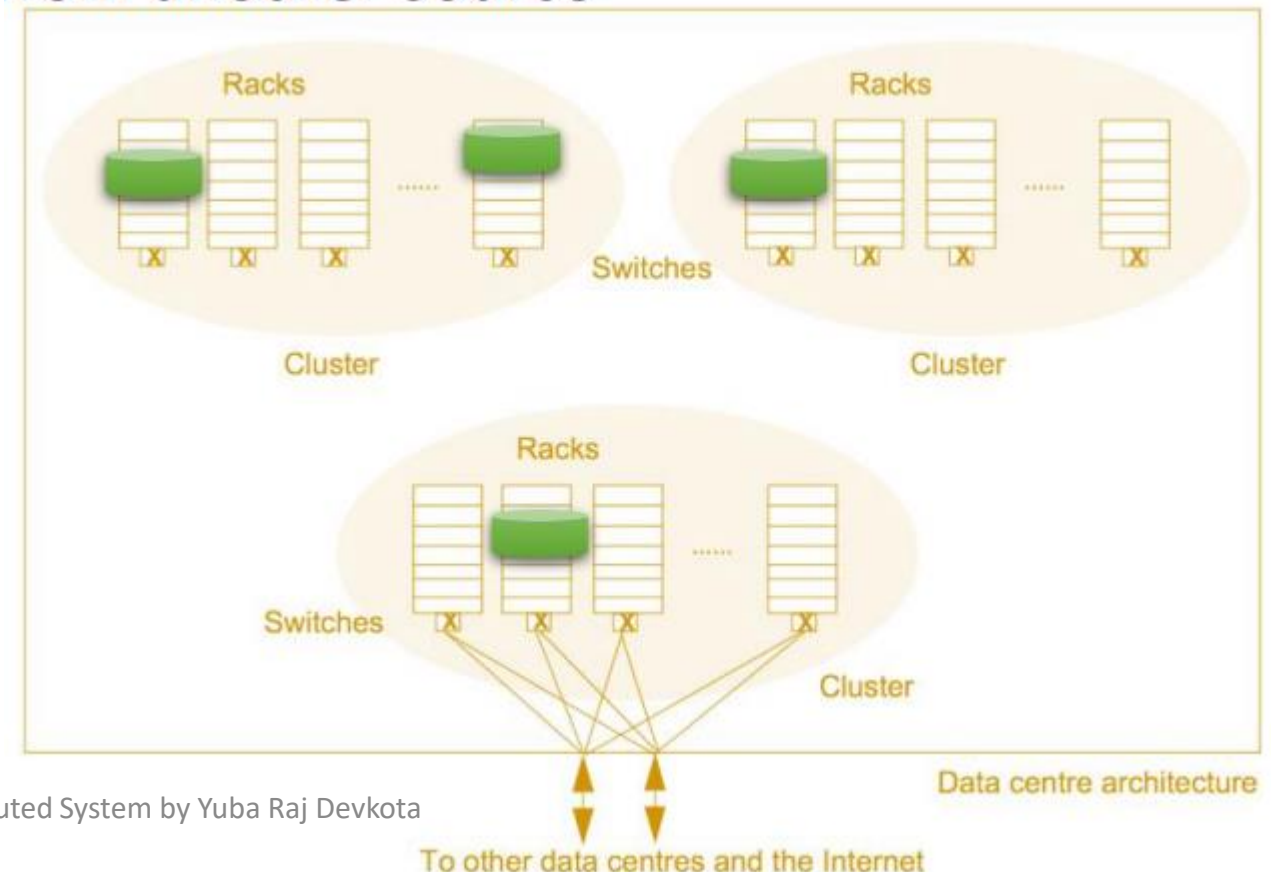
- Example Applications

- Caching webpages at the client browser
- Caching IP addresses at clients and DNS Name Servers
- Caching in Content Delivery Network (CDNs)
 - Commonly accessed contents, such as software and streaming media, are cached at various network locations



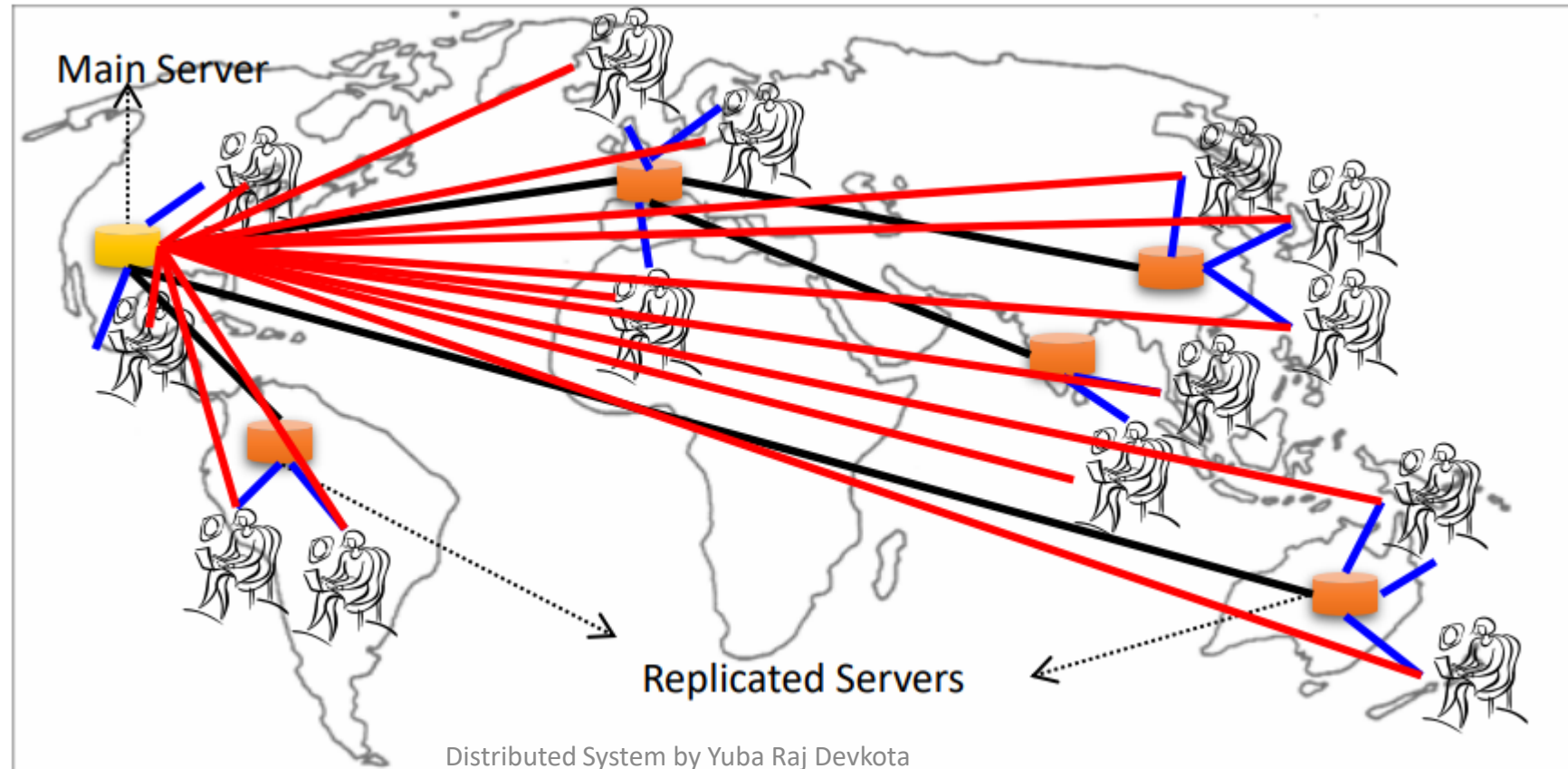
2. Replication for High-Availability

- Availability can be increased by storing the data at replicated locations (instead of storing one copy of the data at a server)
- Example: Google File-System and Chubby replicate the data at computers across different racks, clusters and data-centers
 - If one computer or a rack or a cluster crashes, then the data can still be accessed from another source



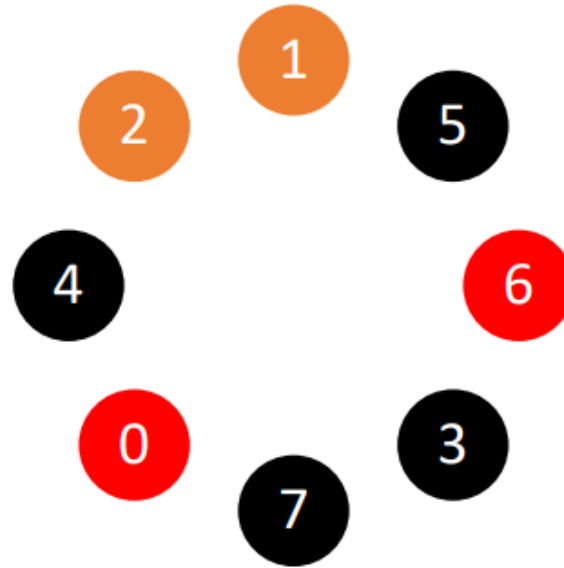
3. Replication for Enhancing Scalability

- Distributing the data across replicated servers helps in avoiding bottle-necks at the main server
 - It balances the load between the main and the replicated servers
- Example: Content Delivery Networks decrease the load on main servers of the website





4. Replication for Securing Against Malicious Attacks


- If a minority of the servers that hold the data are malicious, the non-malicious servers can outvote the malicious servers, thus providing security.
- The technique can also be used to provide fault-tolerance against non-malicious but faulty servers
- Example: In a peer-to-peer system, peers can coordinate to prevent delivering faulty data to the requester



Number of servers with correct data outvote the faulty servers

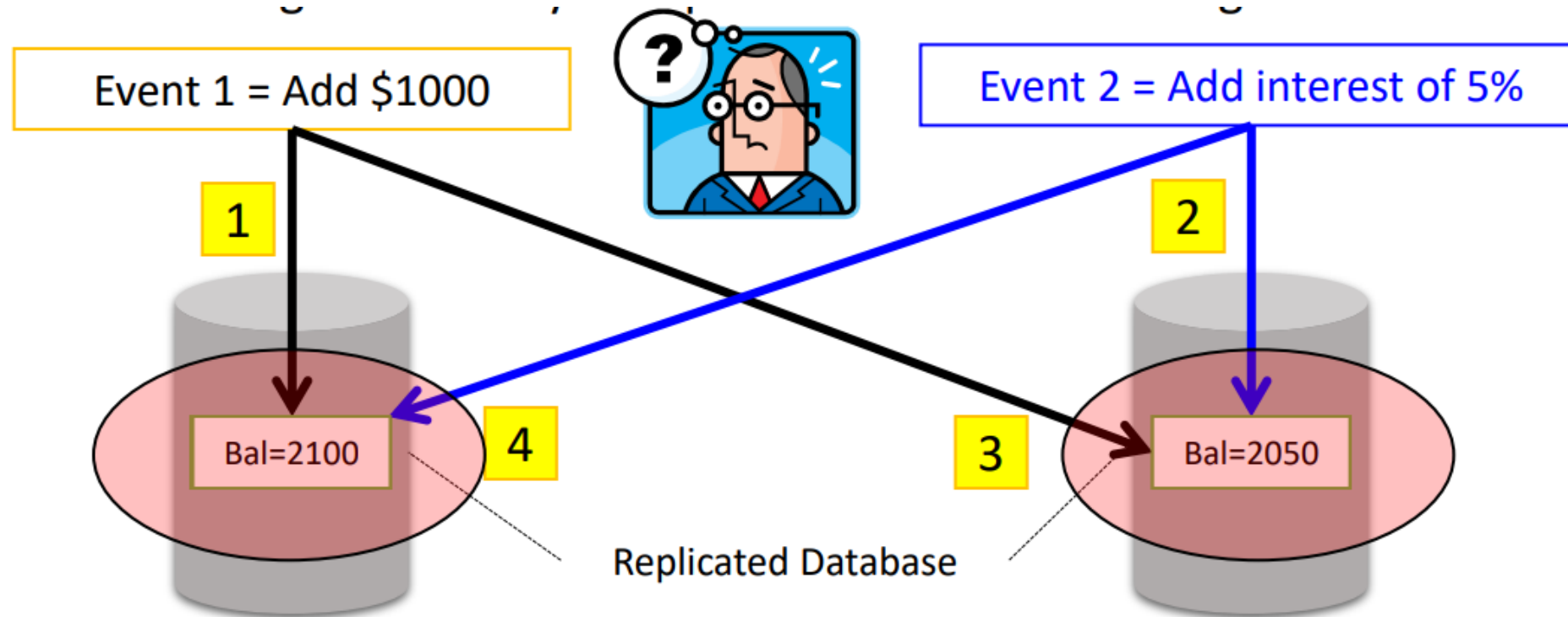
 = Servers that do not have the requested data

 = Servers with correct data

 = Servers with faulty data

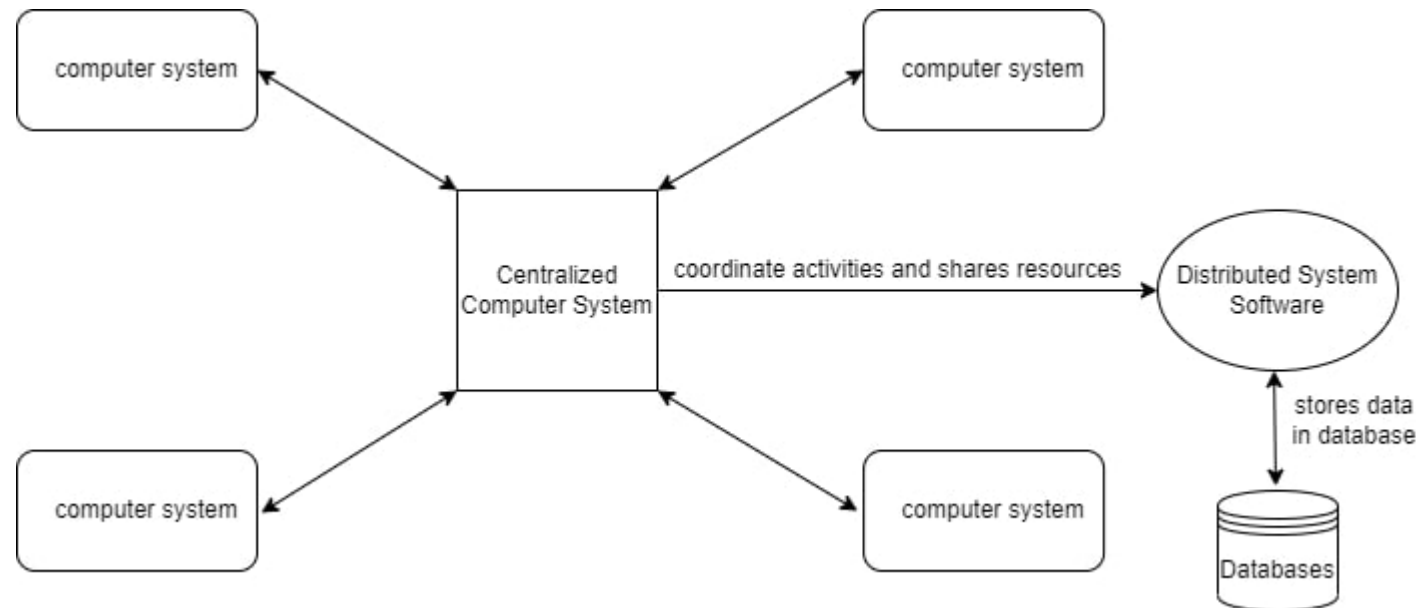
Why Consistency?

- In a DS with replicated data, one of the main problems is keeping the data consistent
- An example:
 - In an e-commerce application, the bank database has been replicated across two servers
 - Maintaining consistency of replicated data is a challenge

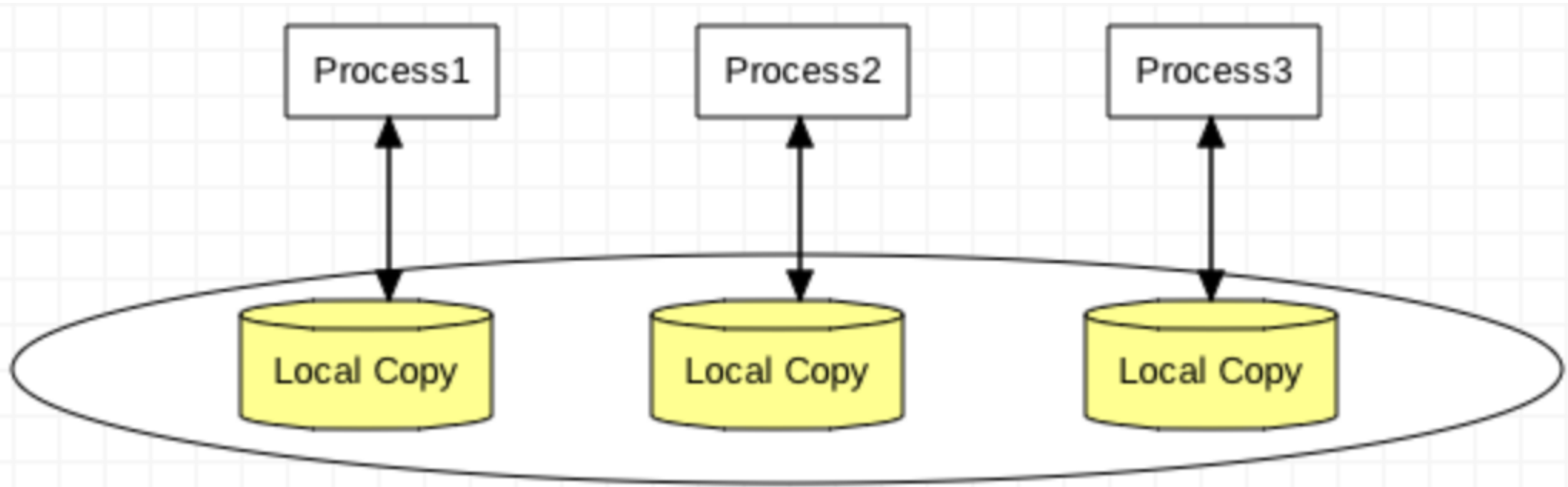


Consistency Models in Distributed System

- In distributed systems, where data is spread across multiple nodes, ensuring consistency—i.e., that all nodes have the same view of the data—is a fundamental challenge. The consistency model defines the rules that govern how and when updates to data are propagated to ensure that all nodes in the system eventually see the same data.
- Consistency model in distributed systems refers to the rules or protocols that dictate how updates to data are propagated and observed by different nodes in the system. It defines the level of agreement between these nodes regarding the state of the data. Consistency models are crucial in ensuring that distributed systems behave predictably and that data remains accurate and coherent across all nodes.



Take replication for example, several processors read data from different nodes for high performance.



The consistency model has to determine whether client B sees the write from client A or not.

Assume that the following case occurs:

- The row X is replicated on nodes M and N
- The client A writes row X to node M
- After a period of time t , client B reads row X from node N

Data Centric Consistency Models

Data-centric consistency models define consistency guarantees from the perspective of the data storage system. These models focus on how updates to data are propagated and how the system ensures a consistent view of data across all nodes. Key data-centric consistency models include:

- Strict Consistency
- Sequential Consistency
- Casual Consistency
- Eventual Consistency

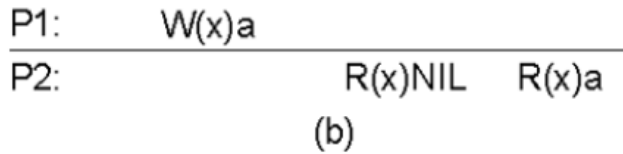
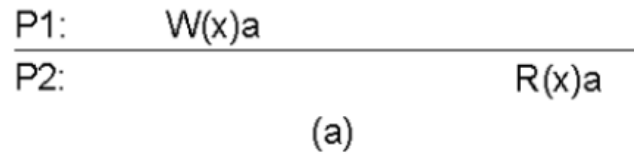
Client Centric Consistency Models

Client-centric consistency models define consistency guarantees from the perspective of the client interacting with the system. These models focus on ensuring a consistent experience for individual clients as they interact with the data. Key client-centric consistency models include:

- Monotonic Reads Consistency
- Monotonic Writes Consistency
- Read-your-writes Consistency
- Writes-follow-reads Consistency
- FIFO Consistency

Strict Consistency

Strict consistency, the strongest model, satisfies the normal expectation: any read on a data item X returns a value corresponding to the result of the most recent write on X. In other words, a write to a variable by any processor needs to be seen instantaneously by all processors.



In the above diagrams,

- a) A strictly consistent store
- b) A store that is non-strict consistent

The strict model need time constraint, which is hardly implemented without a global clock. Absolute time can be physically impossible because it takes time to propagate the copy information.

- 1. Definition:** Any read on a data item reflects the most recent write.
- 2. Characteristics:** Idealized model, difficult to implement in distributed systems due to network delays and partitions.

Sequential Consistency

The sequential consistency model is a weaker memory model than strict consistency meaning A write to a variable does not have to be seen instantaneously. The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some **sequential order** and **the operations of each individual process** appear in this sequence **in the order specified by its program**.

P1:	W(x)a		
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)b	R(x)a

(a)

A sequentially consistent data store.

P1:	W(x)a		
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

(b)

A data store that is not sequentially consistent.

Sequential consistency can produce non-deterministic results. This is because the sequence of sequential operations between processors can be different during different runs of the program. All memory operations need to happen in the program order.

- 1. Definition: The result of any execution is the same as if the operations of all the processes were executed in some sequential order.
- 2. Characteristics: Ensures all processes see operations in the same order, but not necessarily the real-time order.

Causal Consistency

Causal consistency is a weakening model of sequential consistency by *categorizing events into those causally related and those that are not*. It defines that only write operations that are causally related need to be seen in the same order by all processes. Concurrent writes may be seen in a different order on different machines.

A sequence is allowed with a causally-consistent store, but not with sequentially or strictly consistent store.

P1:	W(x)a		W(x)c	
P2:		R(x)a	W(x)b	
P3:		R(x)a		R(x)c
P4:		R(x)a		R(x)c

1. Definition: Writes that are causally related must be seen by all processes in the same order, but concurrent writes may be seen in different orders.
2. Characteristics: More relaxed than sequential consistency; ensures causally related operations are seen in the same order.

Eventual Consistency

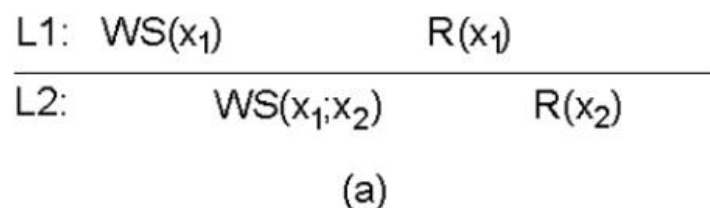
An eventual consistency is a weak consistency model in the system with the lack of simultaneous updates. It defines that if no update takes a very long time, all replicas eventually become consistent.

The most popular system that implements eventual consistency is DNS (Domain Name System). Updates to a name are distributed according to a configured pattern and in combination with time-controlled caches; eventually, all clients will see the update.

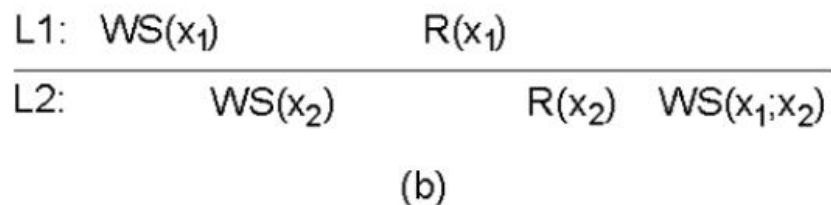
1. **Definition:** If no new updates are made to a data item, eventually all accesses will return the last updated value.
2. **Characteristics:** Ensures convergence over time; widely used in large-scale distributed systems like DNS and NoSQL databases.

Monotonic Reads Consistency

If a process has seen a particular value for the object, any subsequent accesses will never return any previous values.



A monotonic-read consistent data store



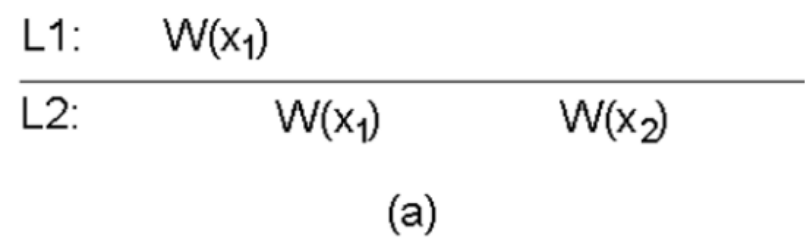
A data store that does not provide monotonic reads.

$WS(x_i)$: write set = sequence of operations on x at node L_i

1. Definition: If a client reads a value of a data item, any subsequent read will return the same value or a more recent value.
2. Characteristics: Ensures clients never see older versions of data after having seen a newer version.

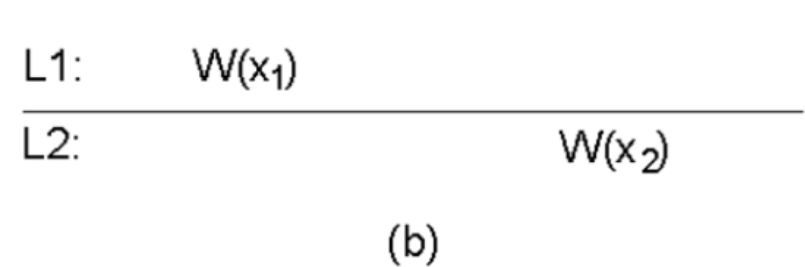
Monotonic Writes Consistency

In this case the system guarantees to serialize the writes by the same process. Systems that do not guarantee this level of consistency are notoriously hard to program.



A monotonic-write consistent data store.

1. Definition: A write operation by a client is guaranteed to be seen by any subsequent write by the same client.



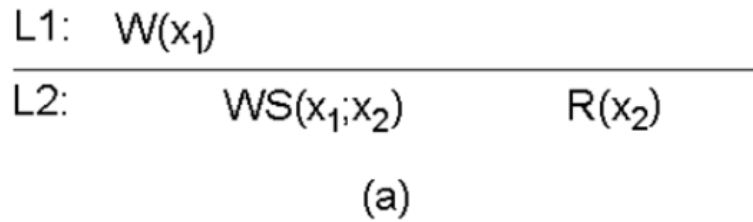
A data store that does not provide monotonic-write consistency.

2. Characteristics: Ensures the order of write operations is preserved for each client.

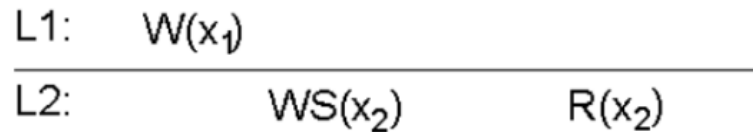
Eg. Value of x is 0, then it needs to update to 10 first and then 20 but got delayed by some reasons, so, x must update to 10 first and then only to 20.

Read-your-writes Consistency

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.



A data store that provides read-your-writes consistency.



A data store that does not.

- Definition: After a client writes a value, any subsequent read by that client will return the written value or a more recent value.
- Characteristics: Ensures immediate consistency for the client's own updates.

First WRITE then READ Operation

Eg. If a password needs to change, first change it (write), then use the updated value (read)

Writes-follow-reads Consistency

In Writes-follow-reads consistency, updates are propagated after performing the previous read operations. 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.

L1:	WS(x_1)	R(x_1)
L2:	WS(x_1, x_2)	W(x_2)

(a)

A writes-follow-reads
consistent data store

L1:	WS(x_1)	R(x_1)
L2:	WS(x_2)	W(x_2)

A data store that does not
provide writes-follow-reads
consistency

- Definition: A write operation by a client following a read operation is guaranteed to be based on the value returned by the read.
- Characteristics: Ensures causality between reads and writes for a client.

First READ operation then WRITE operation

Eg. Giving friends to review on Blogs. They first read it and then only review (Write) it.

FIFO Consistency

In FIFO consistency, writes done by a single process are seen by all other processes in the order in which they were issued, **but** writes from different processes may be seen in a different order by different processes.

P1: W(x)a

P2: R(x)a W(x)b W(x)c

P3: R(x)b R(x)a R(x)c

P4: R(x)a R(x)b R(x)c

Applications and Trade-offs

- **Data-Centric Models:** Typically used in systems where global consistency and coordination are critical, such as banking systems or tightly-coupled databases. These models often require more complex algorithms and protocols to ensure consistency across all nodes.
- **Client-Centric Models:** Often used in distributed systems where availability and partition tolerance are prioritized, such as web applications and content delivery networks. These models can provide a more responsive user experience while maintaining a reasonable level of consistency.

In practice, the choice of consistency model depends on the specific requirements of the application, including factors like performance, availability, network conditions, and the nature of the data.

Consistency Protocols

Consistency protocol is the implementation of a consistency model. Some main approaches include:

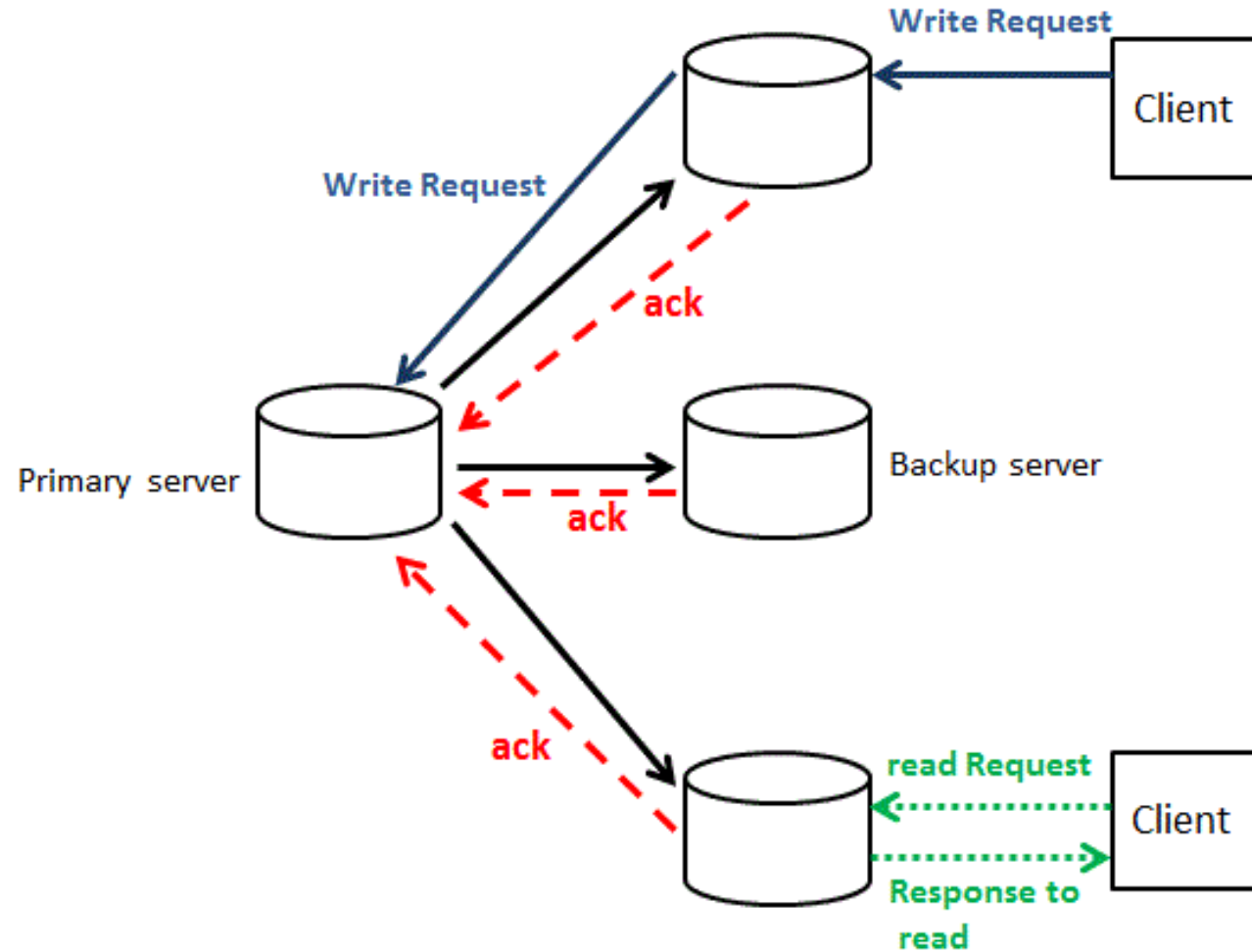
- primary-based protocols (remote write, local write)
- replicated-write protocols (active replication, quorum based)

1. Primary-based Protocols

- Primary-based protocols can be considered as a class of consistency protocols that are simpler to implement.
- For instance, sequential ordering is a popular consistency model when consistent ordering of operations is considered. The sequential ordering can be determined as primary-based protocol.
- In these protocols, there is an associated primary for each data item in a data store to coordinate write operations on that data item.

Remote-write Protocols

In the primary-backup protocol, the simplest primary-based protocol, write operations are routed to a single server and read operations can be performed locally.



Remote Write Protocol

In a remote write protocol, the write operation is performed on a designated node (often the master or primary node), which then updates other nodes (replicas). This can ensure better consistency at the cost of increased write latency.

Example:

Consider a distributed database with three nodes (A, B, and C), where node A is designated as the primary node for writes.

1. **Client Request:** The client sends a write request to node B.
2. **Remote Write:** Node B forwards the write request to node A (the primary node).
3. **Primary Update:** Node A updates the record locally.
4. **Replication:** Node A then propagates the update to nodes B and C.
5. **Acknowledgment:** Nodes B and C acknowledge the update back to node A.
6. **Completion:** Once node A receives acknowledgments from nodes B and C, it sends a success response back to node B, which then responds to the client.

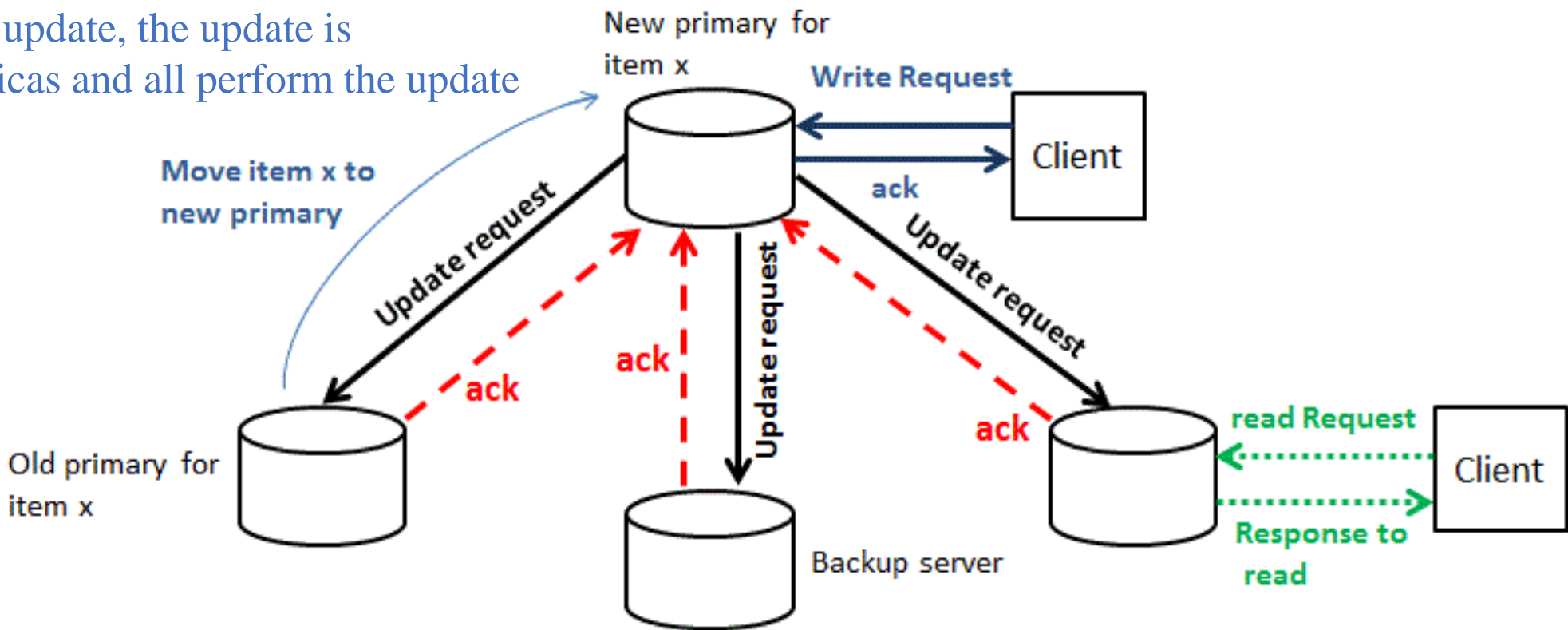
In this protocol, the write is coordinated through the primary node to ensure consistency before confirming the write to the client.



In local-write protocols, primary copy moves between processes willing to perform an update. To update a data item, a process first moves it to its location.

As a result, in this approach, successive write operations can be performed locally while each process can read their local copy of data items. After the primary finishes its update, the update is forwarded to other replicas and all perform the update locally.

Local-write Protocols



Local Write Protocol

In a local write protocol, the write operation is performed on the local node where the request is initiated, and then the changes are propagated to other nodes. This can be useful for reducing write latency but requires mechanisms to ensure consistency across nodes.

Example:

Consider a distributed database with three nodes (A, B, and C). A client wants to update a record on node A.

1. **Client Request:** The client sends a write request to node A to update a record.
2. **Local Write:** Node A updates the record locally.
3. **Propagation:** Node A then propagates the update to nodes B and C to ensure they have the latest data.
4. **Acknowledgment:** Nodes B and C acknowledge the update back to node A.
5. **Completion:** Once node A receives acknowledgments from nodes B and C, it sends a success response back to the client.

In this protocol, the write is considered successful once the local write is completed and acknowledgments are received from other nodes.

Replicated-write Protocols

Unlike the primary-based protocol, all updates are carried out to all replicas in Replicated-write protocols.

Active Replication

In active replication, updates are sent to each replica in the form of an operation in order to be executed. All updates need to be performed in the same order in all replicas.

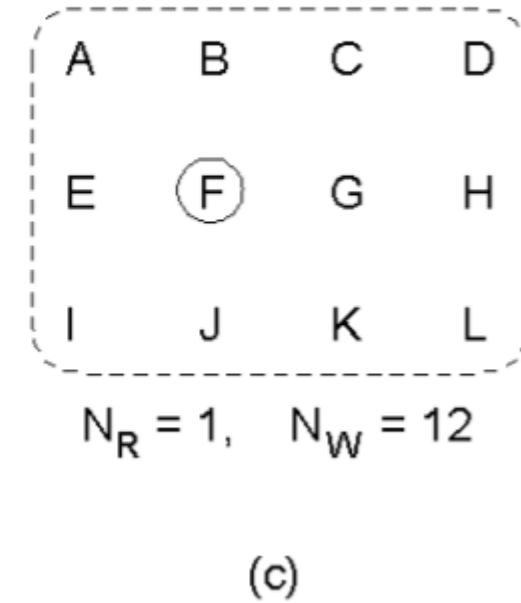
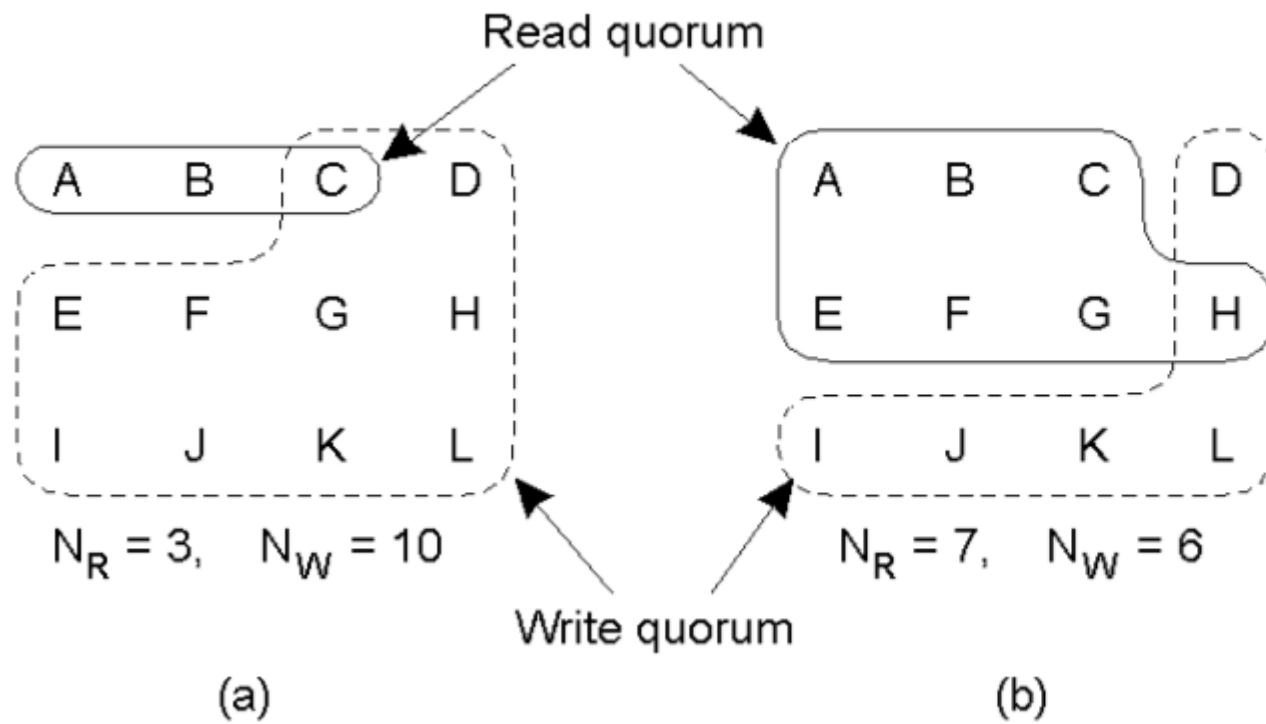
In order to make all the servers receive the same sequence of operations, an atomic broadcast protocol must be used. An atomic broadcast protocol guarantees that either all the servers receive a message or none, plus that they all receive messages in the same order.

Quorum-based Protocols

Voting can be another approach in replicated-write protocols. In this approach, a client requests and receives permission from multiple servers in order to read and write a replicated data.

As an example, suppose in a distributed file system, a file is replicated on N servers. To update a file, a client must send a request to more than $N/2$ in order to make their agreement to perform an update. After the agreement, changes are applied on the file and a new version number is assigned to the updated file.

Similarly, for reading replicated file, a client sends a request to $N/2+1$ servers in order to receive the associated version number from those servers. Read operation is completed if all received version numbers are the most recent version.



Three voting-case examples:

- a) A correct choice of read and write set
- b) A choice that may lead to write-write conflicts
- c) A correct choice, known as ROWA (read one, write all)

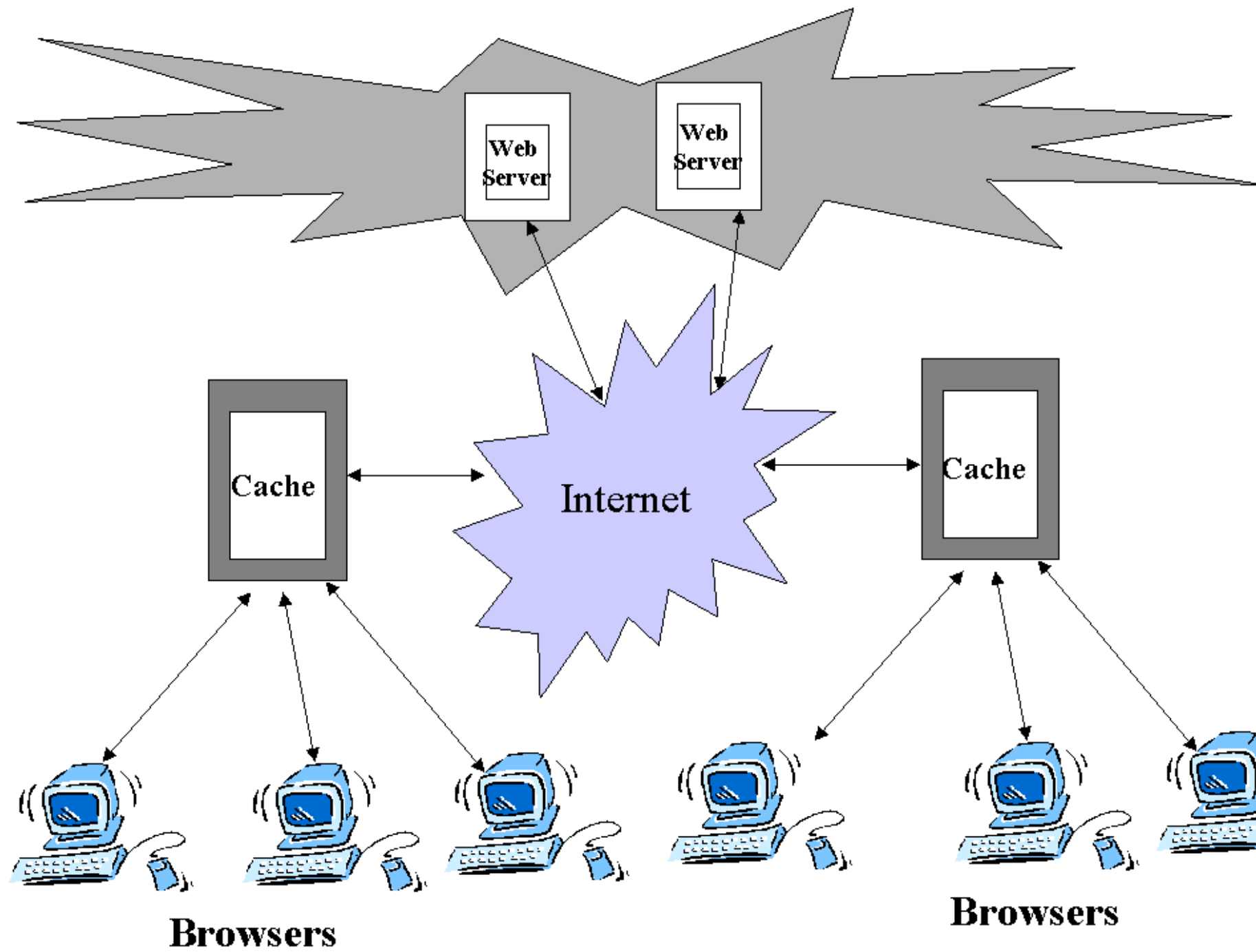
The constraints:

1. $N_R + N_W > N$
2. $N_W > N/2$

Caching and Replication in web

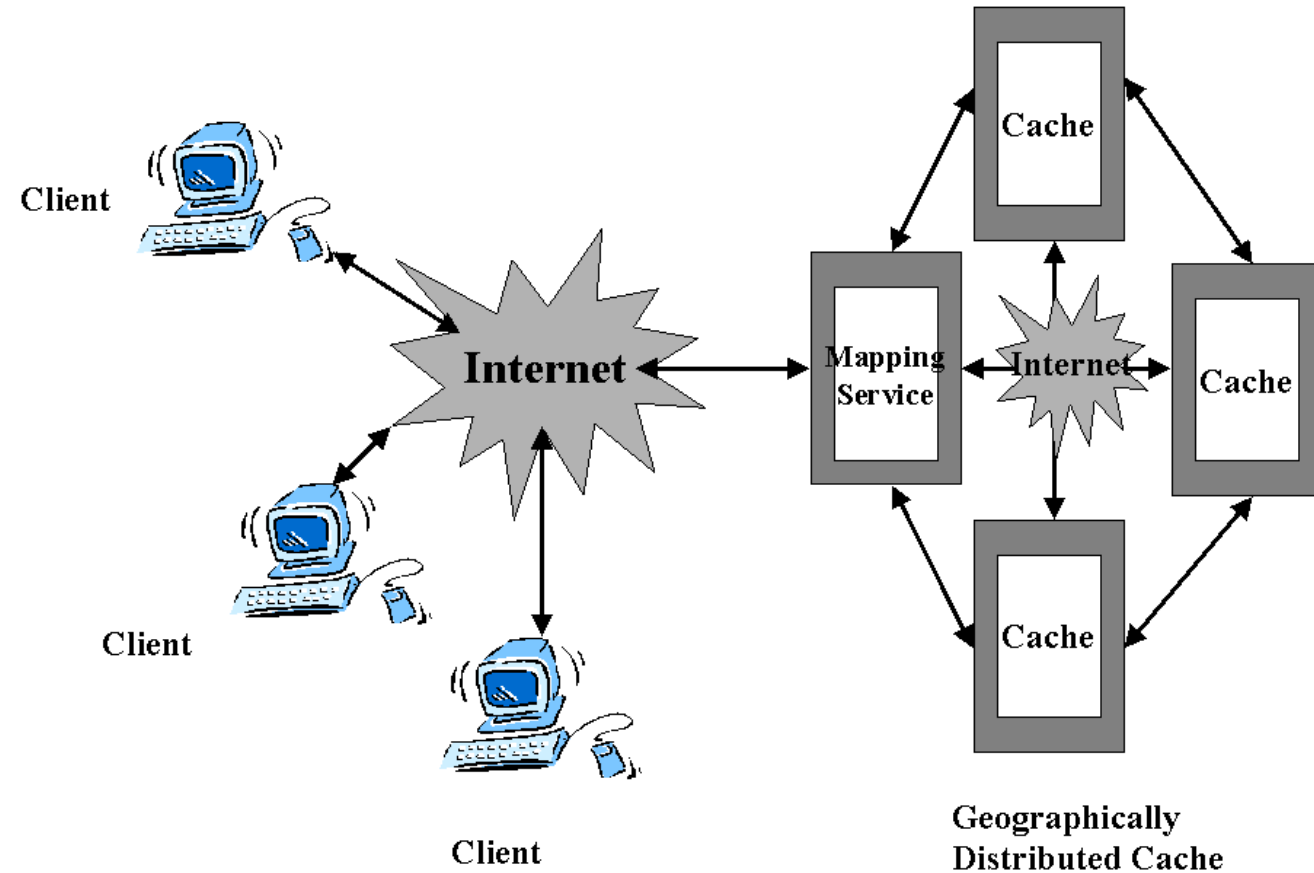
What is a Web Cache ?

- 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.



Replication

- Replication is a technique similar to caching, but is generally considered to be more active. The process of replication copies cache content and pushes it on to one or more cache servers across the network.
- Replication is required to distribute objects among the servers to maintain the freshness of content across servers, which results in reduced upstream network traffic. Typically the same content is pushed across several machines making it more efficient to use Multicast.
- Replication is critical in global operations, where cost of international traffic is high and ways have to be found to mirror data without using too much bandwidth.



Assignment 4/ Old Questions

- Write down the differences between consistency and replication. Explain the data centric consistency model in details.
- Explain different client centric consistency models.
- What do you mean by consistency models? Explain each of them in brief.
- Explain briefly about web cache process in distributed system.