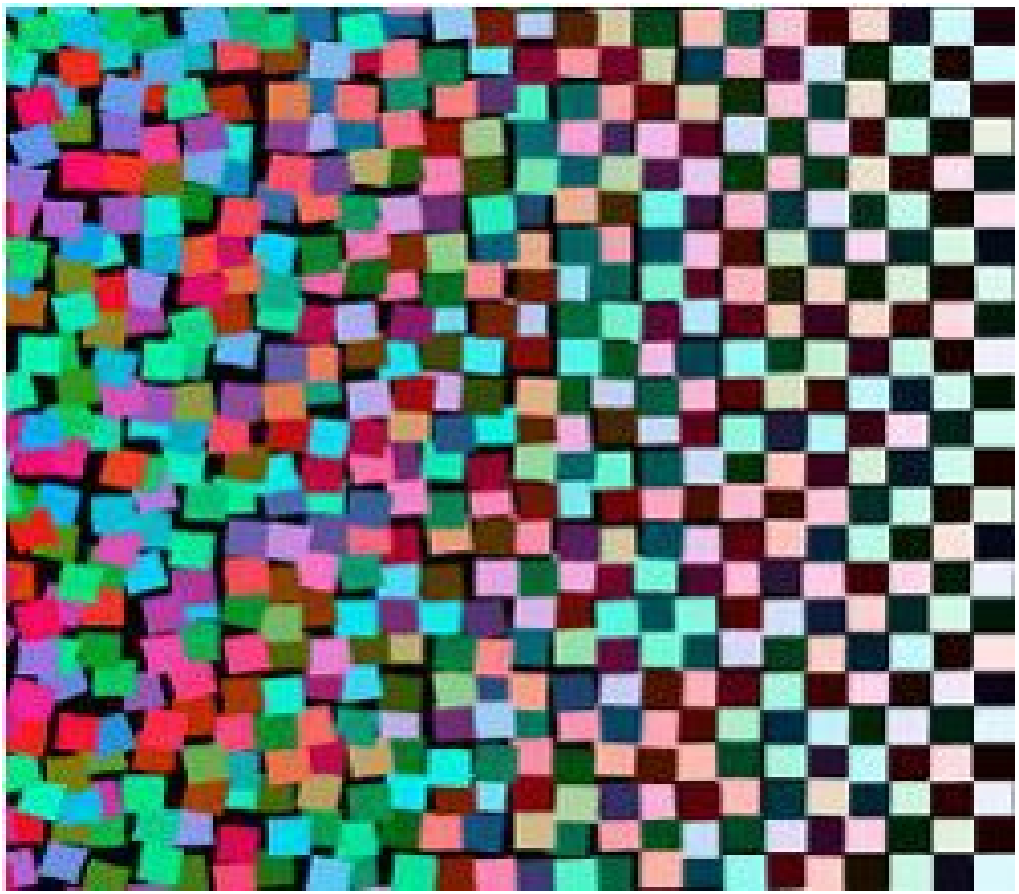
Consistency and Replication



Replicas and Consistency???

Replicas and Consistency???

Replicas and Consistency???

Tatiana Maslany in the show Orphan Black: The story of a group of clones that discover each other and the secret organization Dyad, which was responsible for their creation.

Reasons for Replication

- To increase the reliability of a system.

Reasons for Replication

- To increase the reliability of a system.
- To increase the performance of a system. This can be of two types:

Reasons for Replication

- To increase the reliability of a system.
- To increase the performance of a system. This can be of two types:
  - Scaling in numbers
  - Scaling in geographical area

Reasons for Replication

- To increase the reliability of a system.
- To increase the performance of a system. This can be of two types:
  - Scaling in numbers
  - Scaling in geographical area
- Having multiple copies leads to the problem of consistency. When and how the copies are made consistent determines the price of replication.

Reasons for Replication

- To increase the reliability of a system.
- To increase the performance of a system. This can be of two types:
  - Scaling in numbers
  - Scaling in geographical area
- Having multiple copies leads to the problem of consistency. When and how the copies are made consistent determines the price of replication.
- **Example**: Client caching of web pages in web browser gains performance for the client at the cost of consistency.

Replication as a Scaling Technique

- Placing copies of data close to client processes can help with scaling. But keeping copies up to date requires more network bandwidth. Updating too often may be a waste. Not updating often enough is the flip side.

Replication as a Scaling Technique

- Placing copies of data close to client processes can help with scaling. But keeping copies up to date requires more network bandwidth. Updating too often may be a waste. Not updating often enough is the flip side.
- How to keep the replicas consistent? Use global ordering using Lamport timestamps or use a coordinator. This may require a lot of communication for a large system.

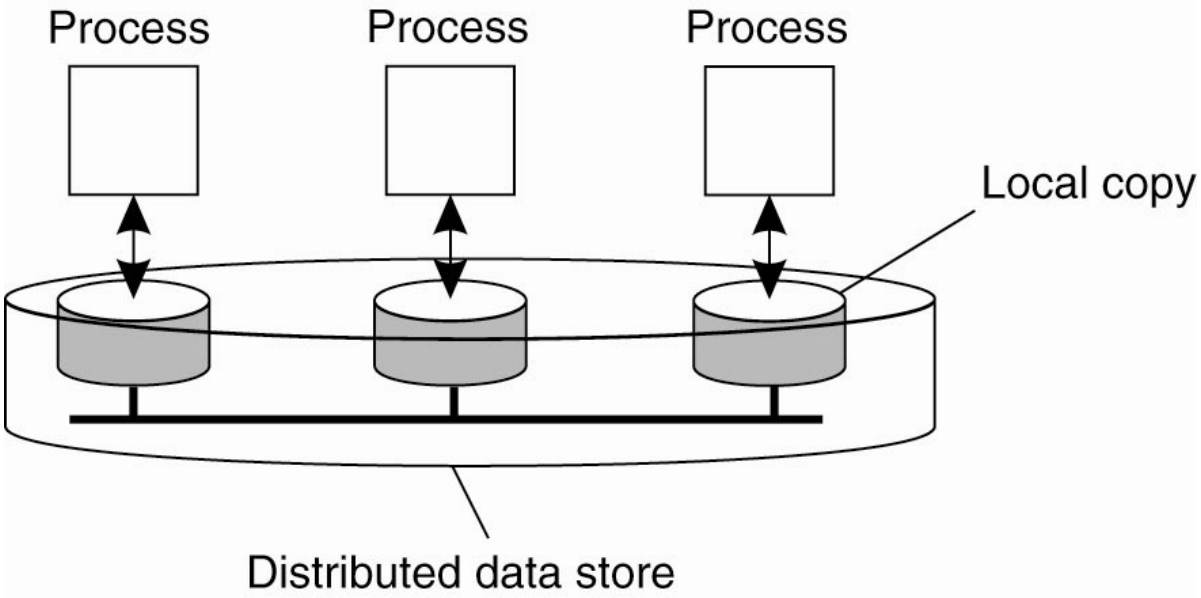
Replication as a Scaling Technique

- Placing copies of data close to client processes can help with scaling. But keeping copies up to date requires more network bandwidth. Updating too often may be a waste. Not updating often enough is the flip side.
- How to keep the replicas consistent? Use global ordering using Lamport timestamps or use a coordinator. This may require a lot of communication for a large system.
- In many cases, the real solution is to loosen consistency constraints. E.g. The updates do not have to be atomic. To what extent we can loosen depends highly on the access and update patterns as well as the application.

Replication as a Scaling Technique

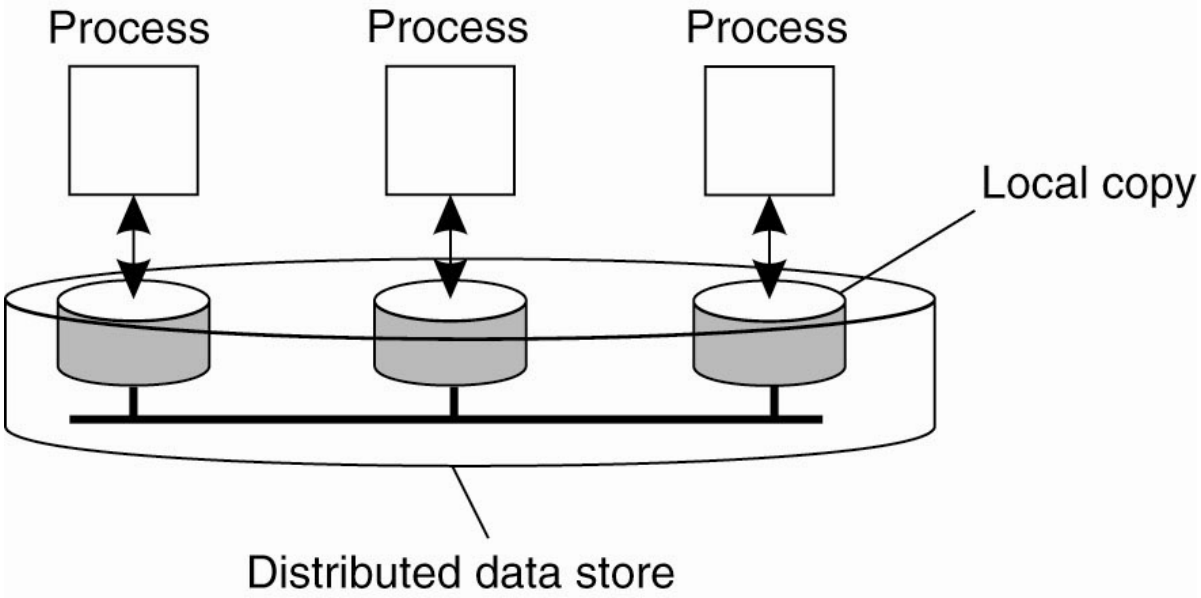
- Placing copies of data close to client processes can help with scaling. But keeping copies up to date requires more network bandwidth. Updating too often may be a waste. Not updating often enough is the flip side.
- How to keep the replicas consistent? Use global ordering using Lamport timestamps or use a coordinator. This may require a lot of communication for a large system.
- In many cases, the real solution is to loosen consistency constraints. E.g. The updates do not have to be atomic. To what extent we can loosen depends highly on the access and update patterns as well as the application.
- A range of consistency models are available.

Data Store



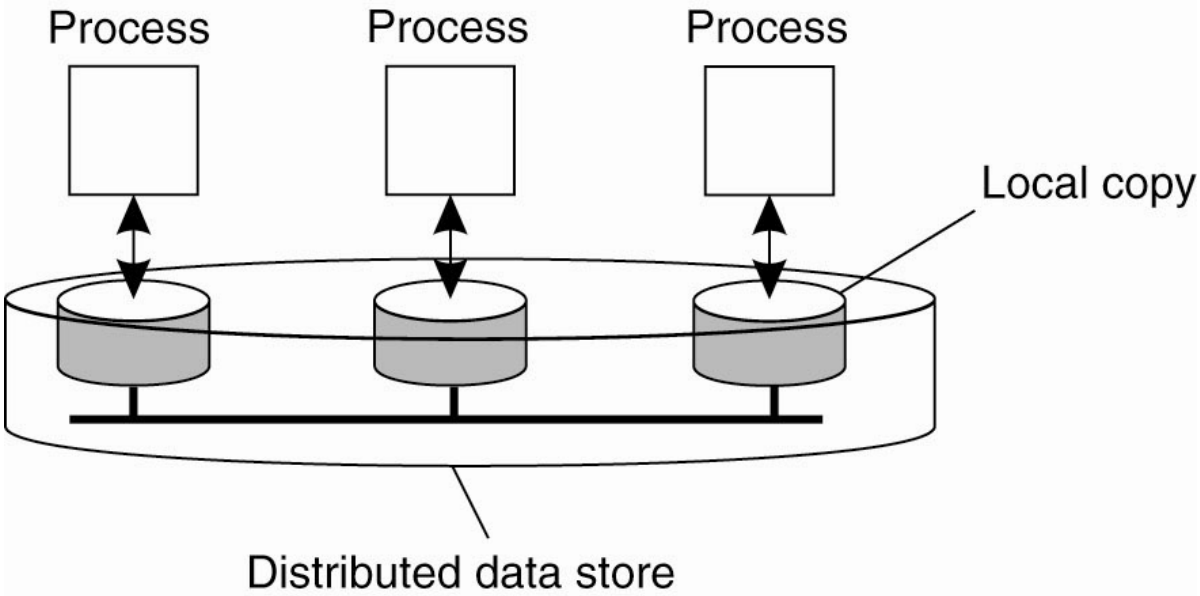
- Examples:
  - distributed shared memory
  - distributed shared database
  - distributed file system

Data Store



- Examples:
  - distributed shared memory
  - distributed shared database
  - distributed file system
- Each process has a local or nearby copy of the whole store or part of it.

Data Store



- Examples:
  - distributed shared memory
  - distributed shared database
  - distributed file system
- Each process has a local or nearby copy of the whole store or part of it.
- A data operation is classified as a write when it changes the data, and is otherwise classifies as a read operation.

What is a Consistency Model?

- A consistency model is essentially a contract between processes and the data store. It says that if processes agree to obey certain rules, the store promises to work correctly.

What is a Consistency Model?

- A consistency model is essentially a contract between processes and the data store. It says that if processes agree to obey certain rules, the store promises to work correctly.
- Models with minor restrictions are easy to use while models with major restrictions are more difficult to use. But then easy models don't perform as well. So we have to make trade-offs.

Consistency Models

- Data-centric consistency models
  - Continuous consistency
  - Sequential consistency
  - Causal consistency
  - Entry Consistency with Grouping operations
- Client-centric consistency models
  - Eventual consistency
  - Monotonic reads
  - Monotonic writes
  - Read your writes
  - Writes follow reads

Sequential Consistency (1)

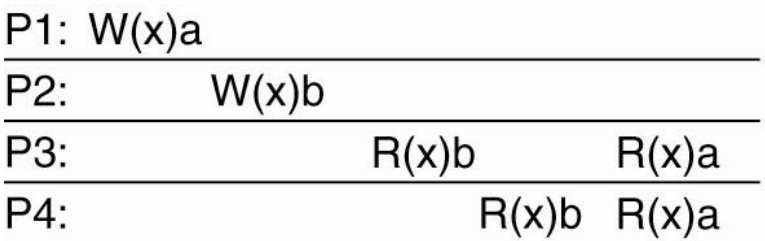
A data store is sequentially consistent when:

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.

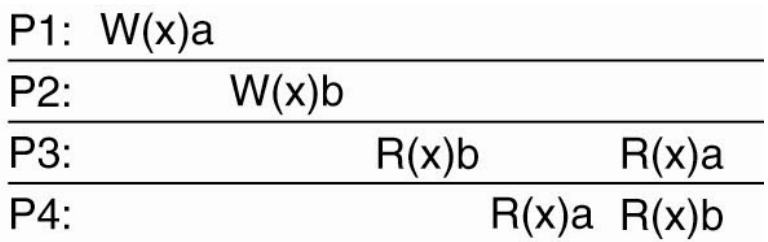
Sequential Consistency (2)

- Behavior of two processes operating on the same data item. The horizontal axis is time.

Sequential Consistency (3)



(a)



(b)

(a) A sequentially consistent data store.

(b) A data store that is not sequentially consistent.

Sequential Consistency (4)

Process P1	Process P2	Process P3
x ← 1;	y ← 1;	z ← 1;
print(y, z);	print(x, z);	print(x, y);

- Three concurrently-executing processes.

Sequential Consistency (5)

(a)	(b)	(c)	(d)
x ← 1;	x ← 1;	y ← 1;	y ← 1;
print(y, z);	y ← 1;	z ← 1;	x ← 1;
y ← 1;	print(x, z);	print(x, y);	z ← 1;
print(x, z);	print(y, z);	print(x, z);	print(x, z);
z ← 1;	z ← 1;	x ← 1;	print(y, z);
print(x, y);	print(x, y);	print(y, z);	print(x, y);
Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111
Signature: 001011	Signature: 101011	Signature: 110101	Signature: 111111

- Four valid execution sequences for the three processes. The vertical axis is time.
- Signature is output of P1, P2, P3 concatenated.

Causal Consistency (1)

For a data store to be considered causally consistent, it is necessary that the store obeys the following condition:

- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.
- Weaker than sequential consistency.

Causal Consistency (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

- This sequence is allowed with a causally-consistent store, but not with a sequentially consistent store. Why?

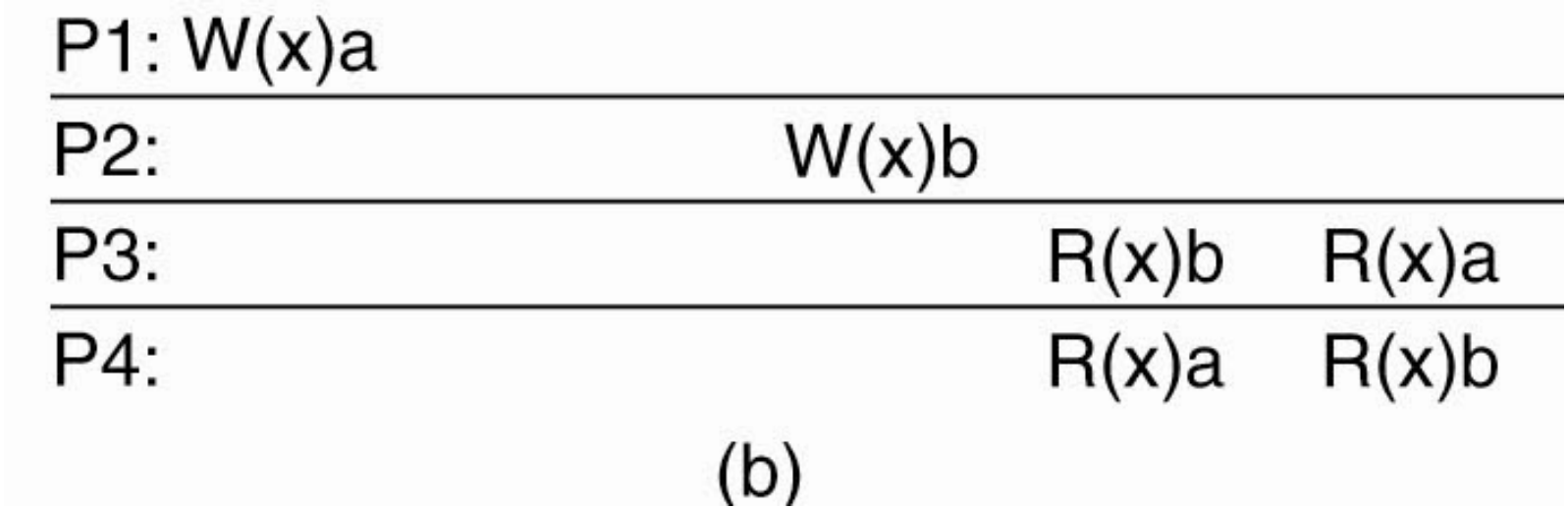
Causal Consistency (3)

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

(a)

- A violation of a causally-consistent store. Why?

Causal Consistency (4)



(b)

- A valid sequence of events in a causally-consistent store.

Causal Consistency (5)

- Implementing causal consistency requires keeping track of which processes have seen which writes.

Causal Consistency (5)

- Implementing causal consistency requires keeping track of which processes have seen which writes.
- It effectively means a dependency graph of which operation is dependent on which other operations must be constructed and maintained.
- This can be done using *vector timestamps*.

Entry Consistency with Grouping Operations (1)

Necessary criteria for correct synchronization:

- An acquire access of a synchronization variable is not allowed to perform until all updates to guarded shared data have been performed with respect to that process.

Entry Consistency with Grouping Operations (1)

Necessary criteria for correct synchronization:

- An acquire access of a synchronization variable is not allowed to perform until all updates to guarded shared data have been performed with respect to that process.
- Before exclusive mode access to synchronization variable by process is allowed to perform with respect to that process, no other process may hold synchronization variable, not even in nonexclusive mode.

Entry Consistency with Grouping Operations (1)

Necessary criteria for correct synchronization:

- An acquire access of a synchronization variable is not allowed to perform until all updates to guarded shared data have been performed with respect to that process.
- Before exclusive mode access to synchronization variable by process is allowed to perform with respect to that process, no other process may hold synchronization variable, not even in nonexclusive mode.
- After exclusive mode access to synchronization variable has been performed, any other process' next nonexclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.

Entry Consistency with Grouping Operations (2)

P1:	Acq(Lx)	W(x)a	Acq(Ly)	W(y)b	Rel(Lx)	Rel(Ly)						
P2:							Acq(Lx)	R(x)a			R(y)	NIL
P3:									Acq(Ly)	R(y)b		

- A valid event sequence for entry consistency.

Client-Centric Consistency Models

- Being able to handle concurrent operations on shared data while maintaining sequential consistency is fundamental to distributed systems. For performance reasons, this may be guaranteed only when processes use synchronization mechanisms.

Client-Centric Consistency Models

- Being able to handle concurrent operations on shared data while maintaining sequential consistency is fundamental to distributed systems. For performance reasons, this may be guaranteed only when processes use synchronization mechanisms.
- A special, important subclass of distributed data stores have the following characterized by:

Client-Centric Consistency Models

- Being able to handle concurrent operations on shared data while maintaining sequential consistency is fundamental to distributed systems. For performance reasons, this may be guaranteed only when processes use synchronization mechanisms.
- A special, important subclass of distributed data stores have the following characterized by:
  - Lack of simultaneous updates (or when such updates happen, they can be easily resolved)

Client-Centric Consistency Models

- Being able to handle concurrent operations on shared data while maintaining sequential consistency is fundamental to distributed systems. For performance reasons, this may be guaranteed only when processes use synchronization mechanisms.
- A special, important subclass of distributed data stores have the following characterized by:
  - Lack of simultaneous updates (or when such updates happen, they can be easily resolved)
  - Most operations involve reading data.

Client-Centric Consistency Models

- Being able to handle concurrent operations on shared data while maintaining sequential consistency is fundamental to distributed systems. For performance reasons, this may be guaranteed only when processes use synchronization mechanisms.
- A special, important subclass of distributed data stores have the following characterized by:
  - Lack of simultaneous updates (or when such updates happen, they can be easily resolved)
  - Most operations involve reading data.
- A very weak consistency model, called eventual consistency, is sufficient in such cases. This can be implemented relatively cheaply by client-centric consistency models.

Eventual Consistency (1)

- A database where most operations are reads. Only one, or very few processes perform update operations. The question then is how fast updates should be made available to only-reading processes.

Eventual Consistency (1)

- A database where most operations are reads. Only one, or very few processes perform update operations. The question then is how fast updates should be made available to only-reading processes.
- DNS system has an authority for each domain so write-write conflicts never happen. The only conflicts we need to handle are read-write conflicts. A lazy propagation is sufficient in this case.

Eventual Consistency (1)

- A database where most operations are reads. Only one, or very few processes perform update operations. The question then is how fast updates should be made available to only-reading processes.
- DNS system has an authority for each domain so write-write conflicts never happen. The only conflicts we need to handle are read-write conflicts. A lazy propagation is sufficient in this case.
- Web pages are updated by a single authority. There are normally no write-write conflicts to resolve. Clients such as Web browsers and proxies cache pages so they may return out-of-date Web pages. Users tolerate this inconsistency.

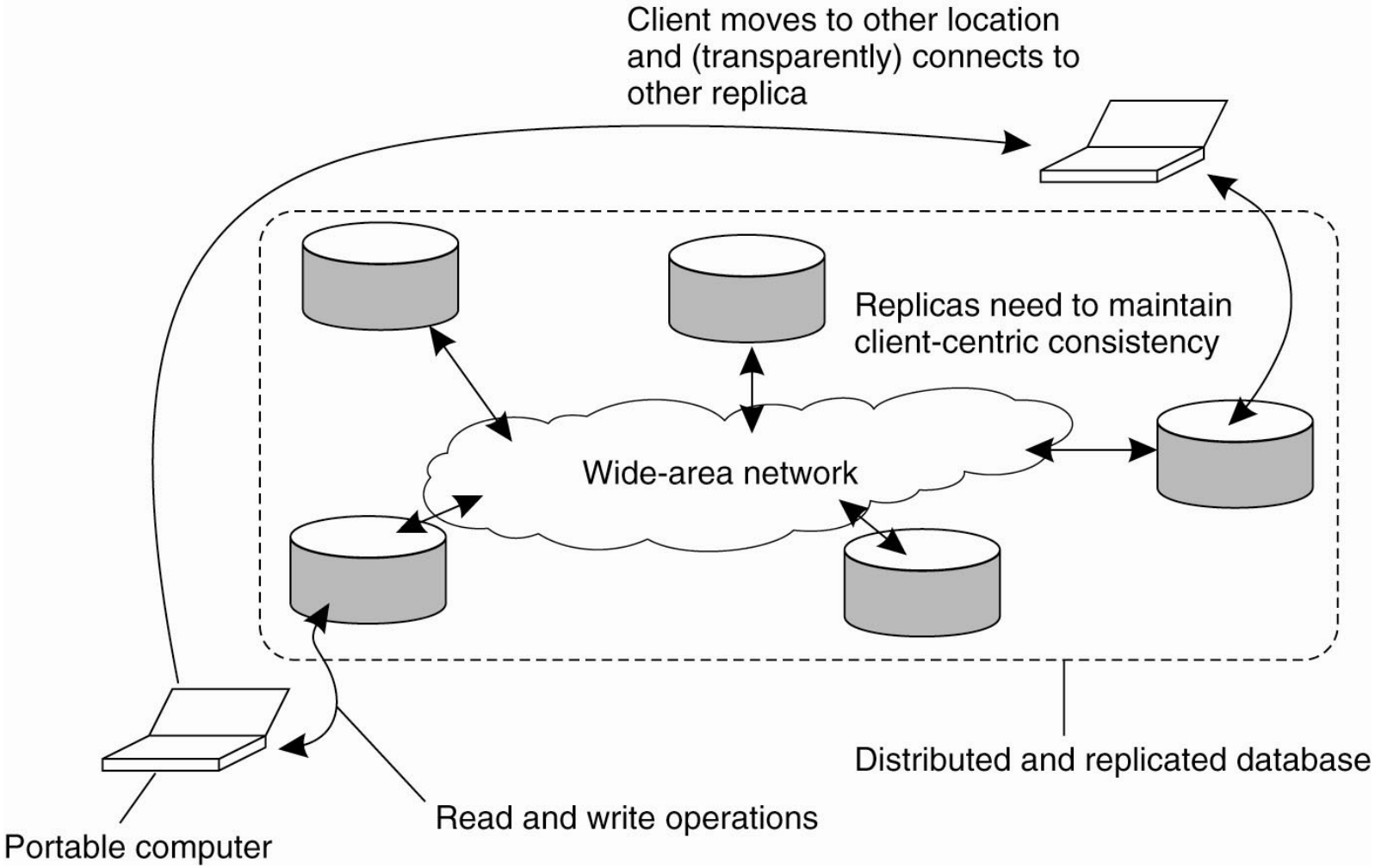
Eventual Consistency (1)

- A database where most operations are reads. Only one, or very few processes perform update operations. The question then is how fast updates should be made available to only-reading processes.
- DNS system has an authority for each domain so write-write conflicts never happen. The only conflicts we need to handle are read-write conflicts. A lazy propagation is sufficient in this case.
- Web pages are updated by a single authority. There are normally no write-write conflicts to resolve. Clients such as Web browsers and proxies cache pages so they may return out-of-date Web pages. Users tolerate this inconsistency.
- If no updates take place, all replicas will gradually becomes consistent. This form of consistency is called eventual consistency.

Eventual Consistency (1)

- A database where most operations are reads. Only one, or very few processes perform update operations. The question then is how fast updates should be made available to only-reading processes.
- DNS system has an authority for each domain so write-write conflicts never happen. The only conflicts we need to handle are read-write conflicts. A lazy propagation is sufficient in this case.
- Web pages are updated by a single authority. There are normally no write-write conflicts to resolve. Clients such as Web browsers and proxies cache pages so they may return out-of-date Web pages. Users tolerate this inconsistency.
- If no updates take place, all replicas will gradually becomes consistent. This form of consistency is called eventual consistency.
- Eventual consistency essentially requires only that updates are guaranteed to propagate to all replicas. Write-write conflicts are easy to solve when assuming that only a mall group of processes can perform updates. Eventual consistency is therefore often cheap to implement.

Eventual Consistency (2)



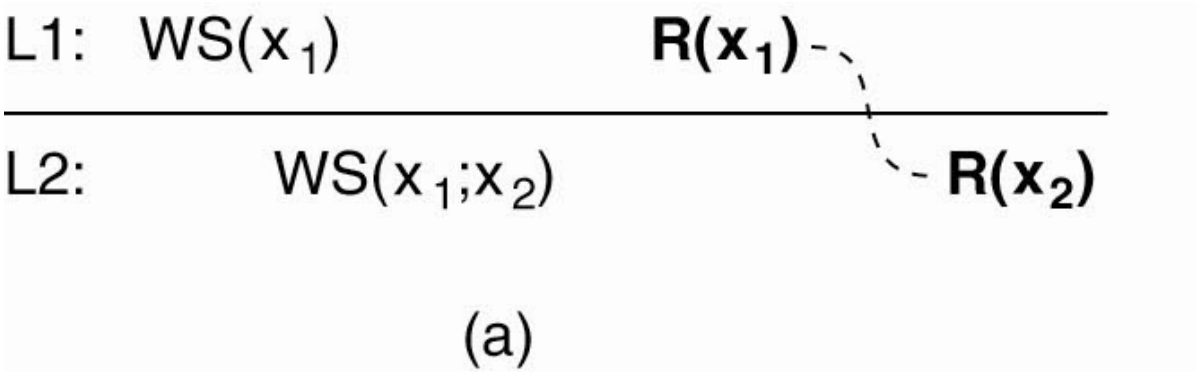
- The above problem can be alleviated by using *client-centric consistency*, which provides guarantees for a single client consistency in accessing the data store.

Monotonic Reads (1)

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

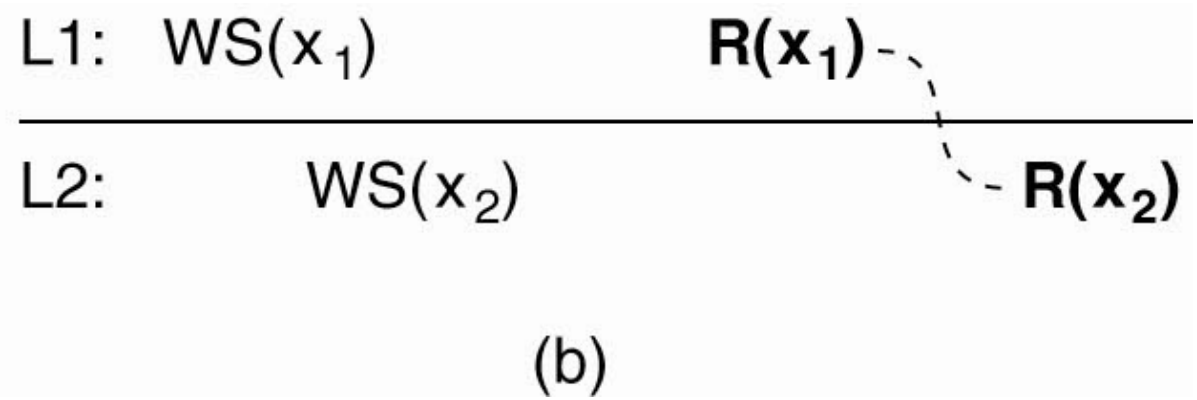
- If a process reads the value of a data item x, then any successive read operation on x by that process will always return that same value or a more recent value.
- But no guarantees on concurrent access by different clients.

Monotonic Reads (2)



- The read operations performed by a single process P at two different local copies of the same data store. (a) A monotonic-read consistent data store.

Monotonic Reads (3)



(b)

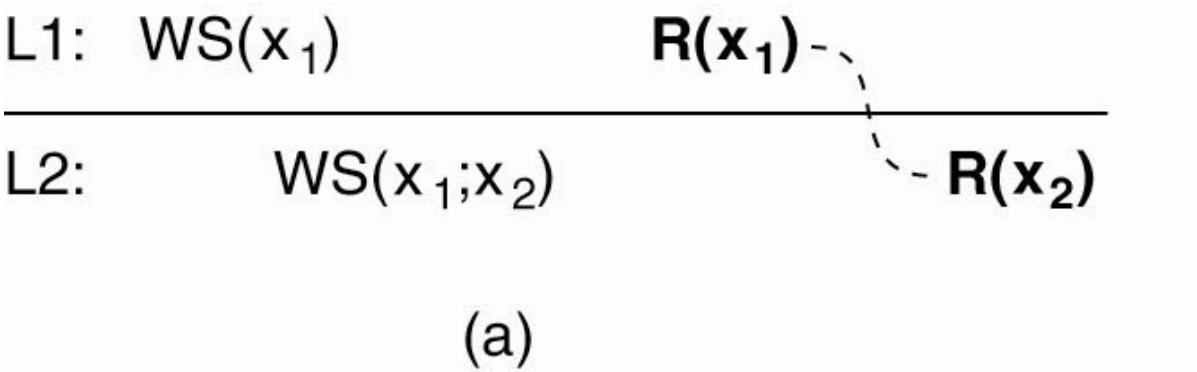
- The read operations performed by a single process P at two different local copies of the same data store. (b) A data store that does not provide monotonic reads.

Monotonic Writes (1)

In a monotonic-write consistent store, the following condition holds:

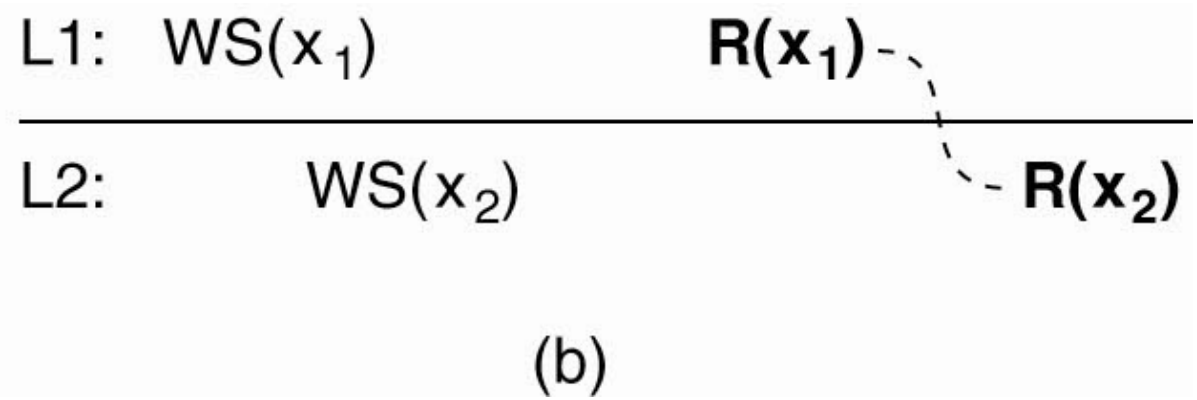
- A write operation by a process on a data item x is completed before any successive write operation on x by the same process.

Monotonic Writes (2)



- The read operations performed by a single process P at two different local copies of the same data store. (a) A monotonic-write consistent data store.

Monotonic Writes (3)



(b)

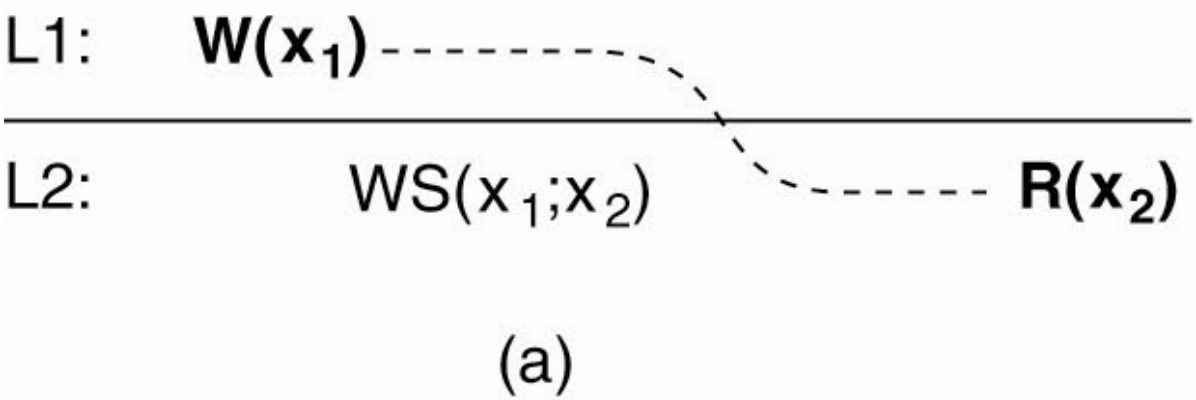
- The read operations performed by a single process P at two different local copies of the same data store. (b) A data store that does not provide monotonic reads.

Read Your Writes (1)

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

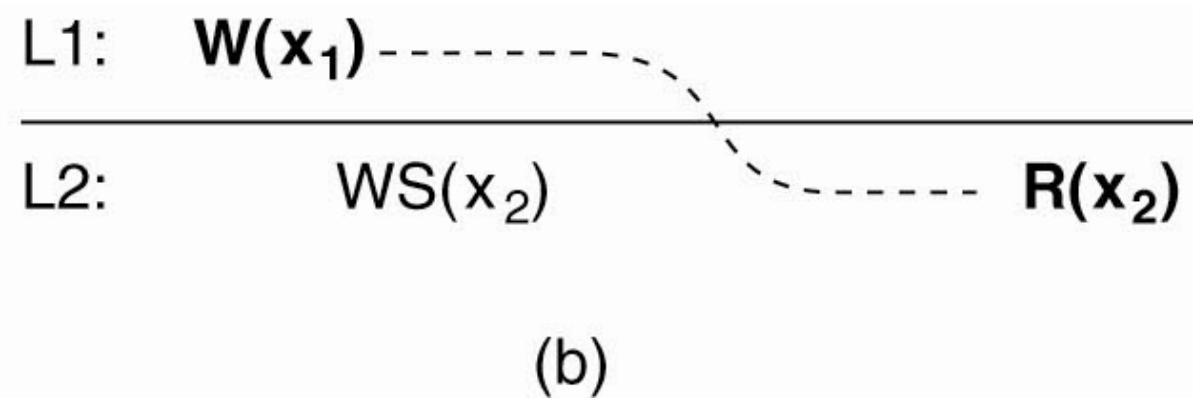
- The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process.

Read Your Writes (2)



- A data store that provides read-your-writes consistency. $WS(x_1)$ is the series of write operations that took place since initialization at a local copy. $WS(x_1; x_2)$ denotes that operations in $WS(x_1)$ also been performed at another local copy that has its set of operations in $WS(x_2)$.

Read Your Writes (3)



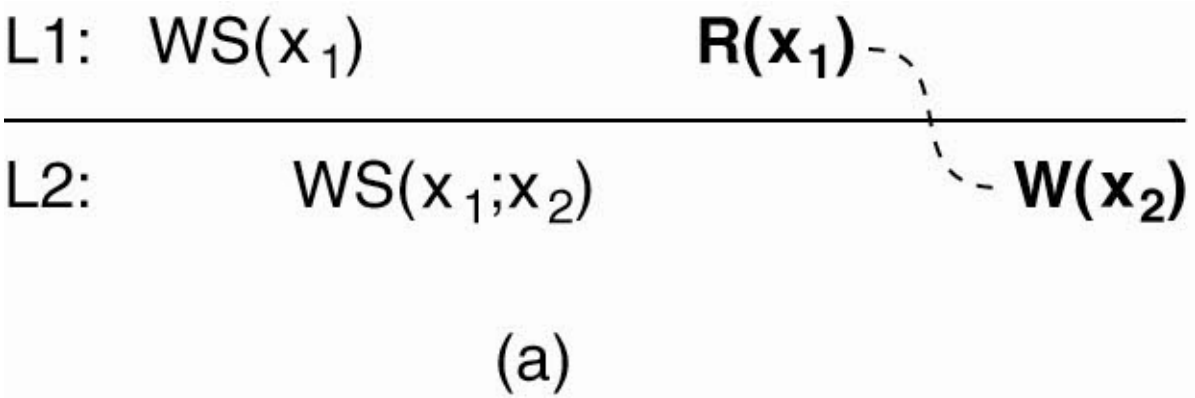
- A data store that does not follow *Read-Your-Writes* consistency model.

Writes Follow Reads (1)

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

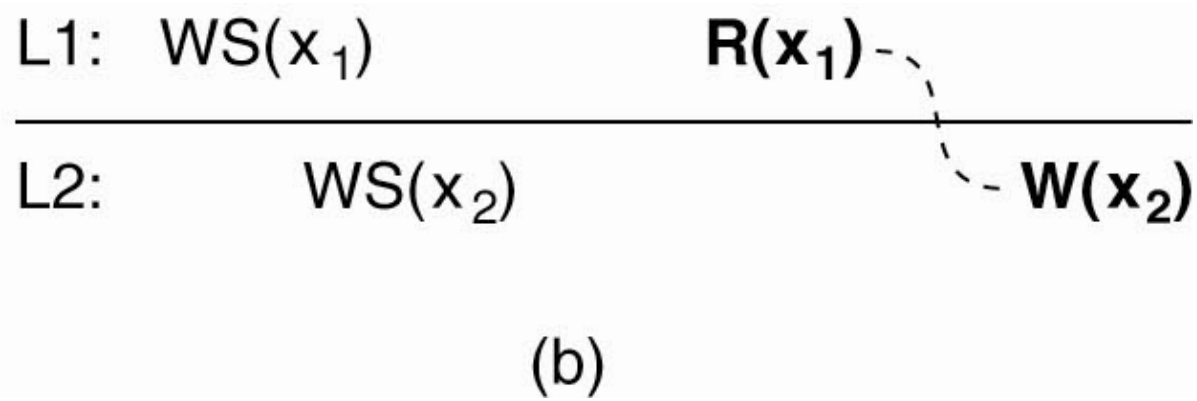
- A write operation by a process on a data item x following a previous read operation on x by the same process is guaranteed to take place on the same or a more recent value of x that was read.

Writes Follow Reads (2)



- A *Writes-Follow-Reads* consistent data store.

Writes Follow Reads (3)



- A data store that does not follow *Writes-Follow-Reads* consistency model.

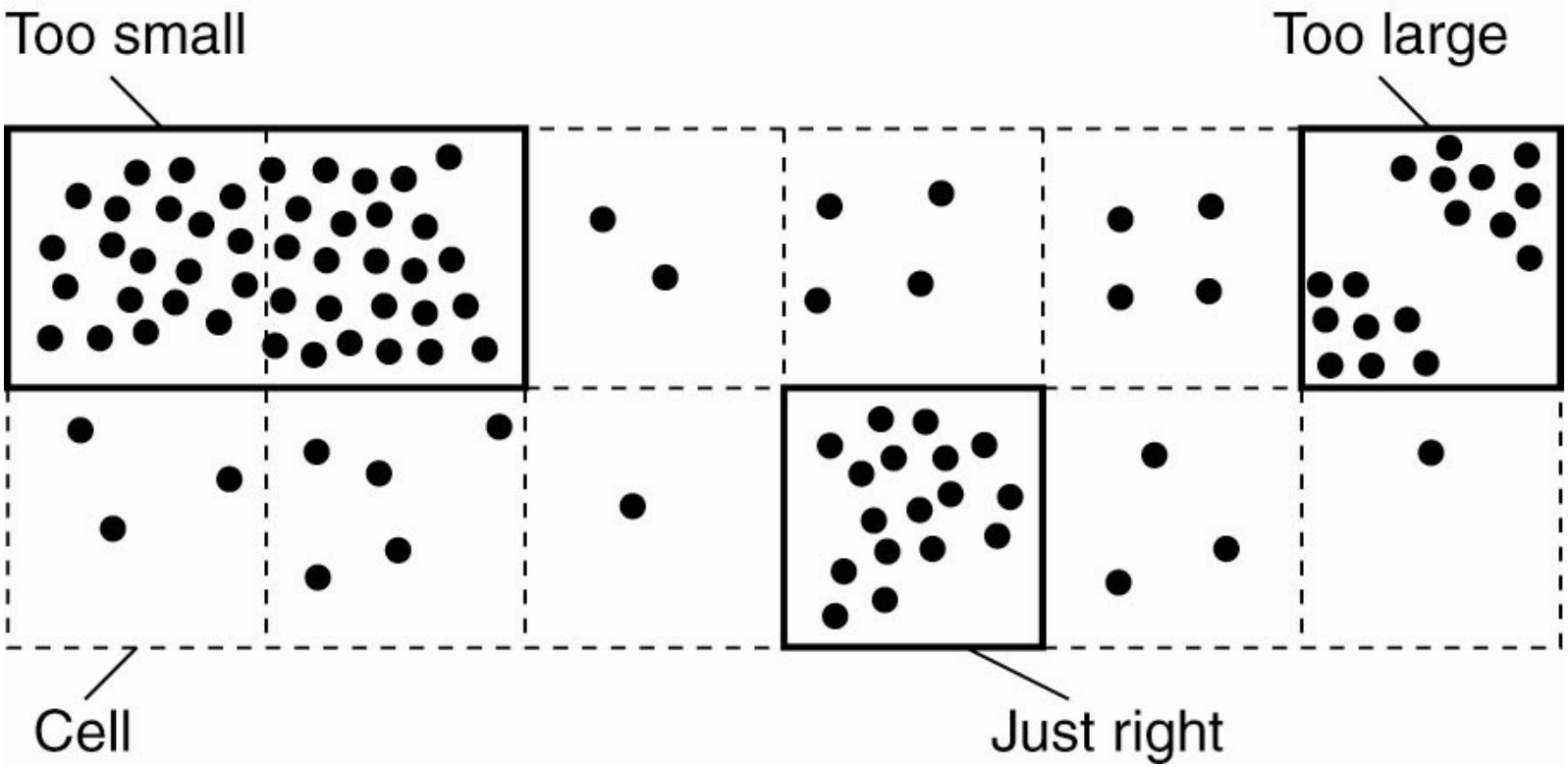
Replica Management

- A key issue for any distributed system that supports replication is to decide where, when and by whom replicas should be placed, and subsequently the mechanisms to use for keeping the replicas consistent.

Replica Management

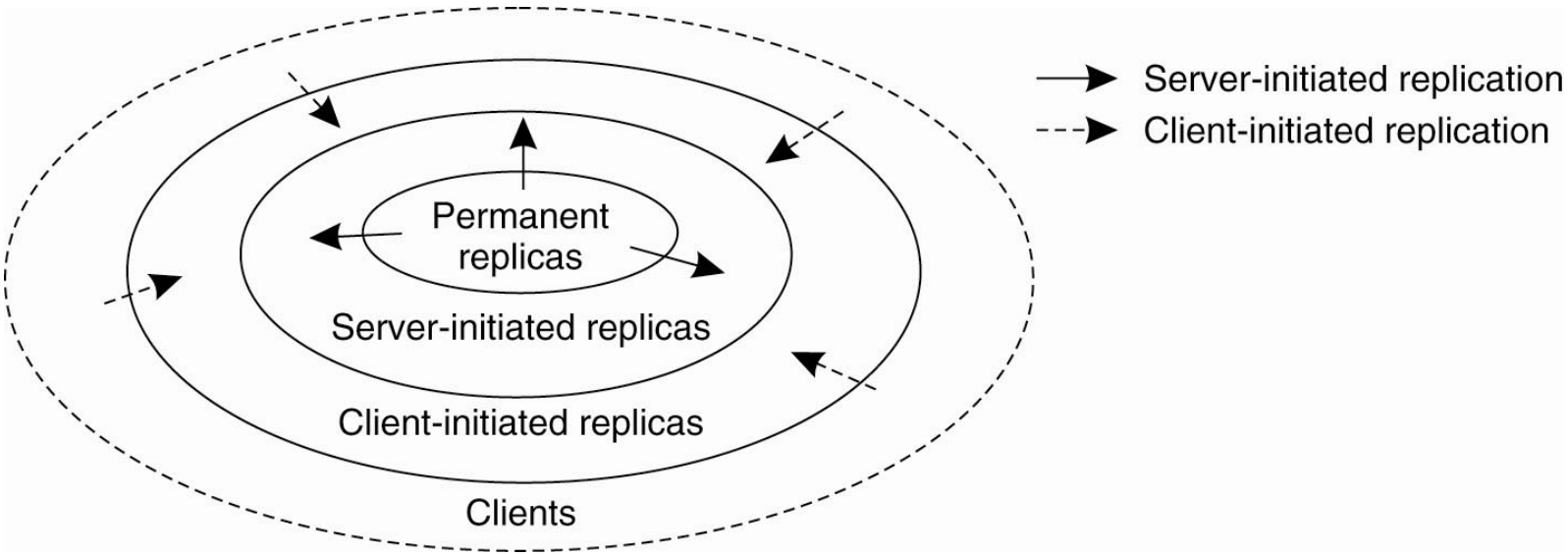
- A key issue for any distributed system that supports replication is to decide where, when and by whom replicas should be placed, and subsequently the mechanisms to use for keeping the replicas consistent. Two separate problems:
- placing replica servers
- placing content

Replica-Server Placement



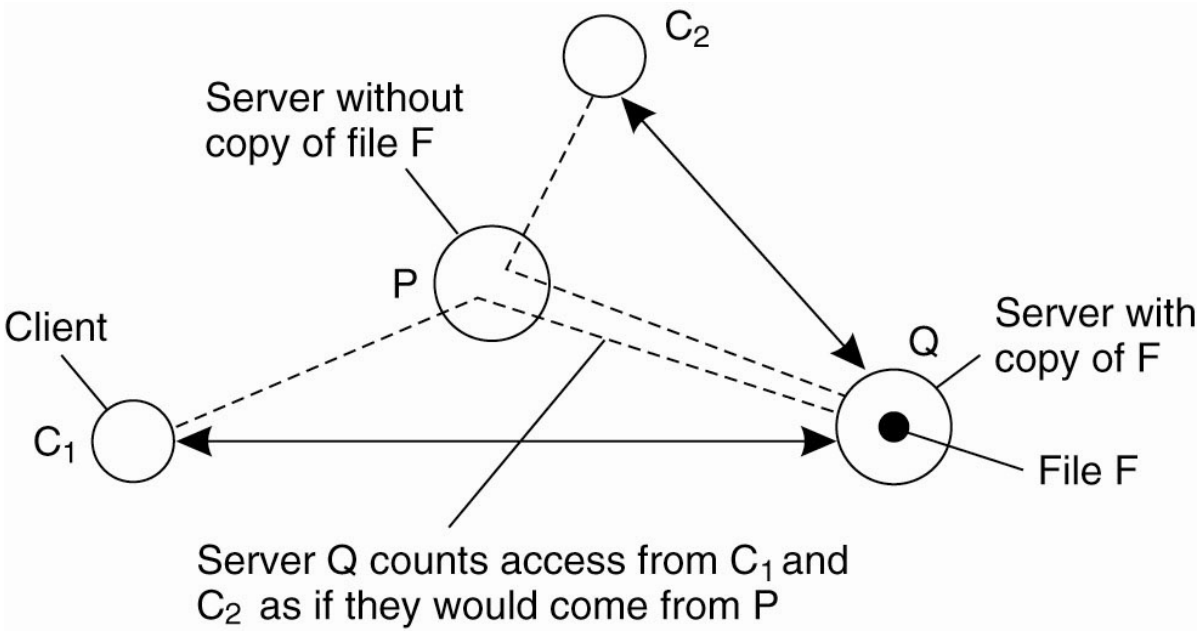
- Choosing a proper cell size for server placement.

Content Replication and Placement



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

Server-Initiated Replicas



- Counting access requests from different clients.
- Three possible actions: migrate, delete, replicate.

Client-Initiated Replicas

- Same as client caches.
- Used to improve access times.
- Sharing a cache between clients may or may not improve the hit rate. E.g. Web data, file servers
- Shared caches can be at departmental or organization level

Content Distribution (1)

- *Design issue*: What is actually to be propagated?

Content Distribution (1)

- *Design issue*: What is actually to be propagated?
  - Propagate only a notification of an update.

Content Distribution (1)

- *Design issue*: What is actually to be propagated?
  - Propagate only a notification of an update.
  - Transfer data from one copy to another.

Content Distribution (1)

- *Design issue*: What is actually to be propagated?
  - Propagate only a notification of an update.
  - Transfer data from one copy to another.
  - Propagate the update operation to other copies.

Content Distribution (1)

- *Design issue*: What is actually to be propagated?
  - Propagate only a notification of an update.
  - Transfer data from one copy to another.
  - Propagate the update operation to other copies.

Content Distribution (2)

- Push Protocol: Updates are propagated to other replicas without being asked. This is useful for keeping a relatively higher degree of consistency.
- A push-based approach is efficient when the read-to-update ratio is relatively high.

Content Distribution (2)

- Push Protocol: Updates are propagated to other replicas without being asked. This is useful for keeping a relatively higher degree of consistency.
- A push-based approach is efficient when the read-to-update ratio is relatively high.
- Pull Protocol: A server or a client requests another server to send it any updates it has at the moment. A pull-based approach is efficient when the read-to-update ratio is relatively low.

Content Distribution (2)

- Push Protocol: Updates are propagated to other replicas without being asked. This is useful for keeping a relatively higher degree of consistency.
- A push-based approach is efficient when the read-to-update ratio is relatively high.
- Pull Protocol: A server or a client requests another server to send it any updates it has at the moment. A pull-based approach is efficient when the read-to-update ratio is relatively low.
- A comparison between push-based and pull-based protocols in the case of multiple-client, single-server systems.

Issue	Push-based	Pull-based
State at server	List of client replicas and caches	None
Messages sent	Update (and possibly fetch update later)	Poll and update
Response time at client	Immediate (or fetch-update time)	Fetch-update time

Content Distribution (3)

- A lease is a promise by the server that it will push updates to the client for a specified time.

Content Distribution (3)

- A lease is a promise by the server that it will push updates to the client for a specified time.
- When a lease expires, the client is forced to poll the server for updates and pull in the modified data if necessary. We can use leases to dynamically switch between push and pull.

Content Distribution (3)

- A lease is a promise by the server that it will push updates to the client for a specified time.
- When a lease expires, the client is forced to poll the server for updates and pull in the modified data if necessary. We can use leases to dynamically switch between push and pull.
- Types of leases:
  - Aged-based leases. Grant long lasting leases to data that is expected to remain unmodified.

Content Distribution (3)

- A lease is a promise by the server that it will push updates to the client for a specified time.
- When a lease expires, the client is forced to poll the server for updates and pull in the modified data if necessary. We can use leases to dynamically switch between push and pull.
- Types of leases:
  - Aged-based leases. Grant long lasting leases to data that is expected to remain unmodified.
  - Renewal-frequency based leases. Give longer leases to clients where its data is popular.

Content Distribution (3)

- A lease is a promise by the server that it will push updates to the client for a specified time.
- When a lease expires, the client is forced to poll the server for updates and pull in the modified data if necessary. We can use leases to dynamically switch between push and pull.
- Types of leases:
  - Aged-based leases. Grant long lasting leases to data that is expected to remain unmodified.
  - Renewal-frequency based leases. Give longer leases to clients where its data is popular.
  - State-space overhead based leases. Start lowering expiration times as space runs low.

Content Distribution (3)

- A lease is a promise by the server that it will push updates to the client for a specified time.
- When a lease expires, the client is forced to poll the server for updates and pull in the modified data if necessary. We can use leases to dynamically switch between push and pull.
- Types of leases:
  - Aged-based leases. Grant long lasting leases to data that is expected to remain unmodified.
  - Renewal-frequency based leases. Give longer leases to clients where its data is popular.
  - State-space overhead based leases. Start lowering expiration times as space runs low.
- Using multicasting can be much more efficient than unicasting for the updates.

Implementations: Primary-based Protocols

- Implementations tend to prefer simpler consistency models.

Implementations: Primary-based Protocols

- Implementations tend to prefer simpler consistency models.
- Primary-based protocols are used for implementing sequential consistency.

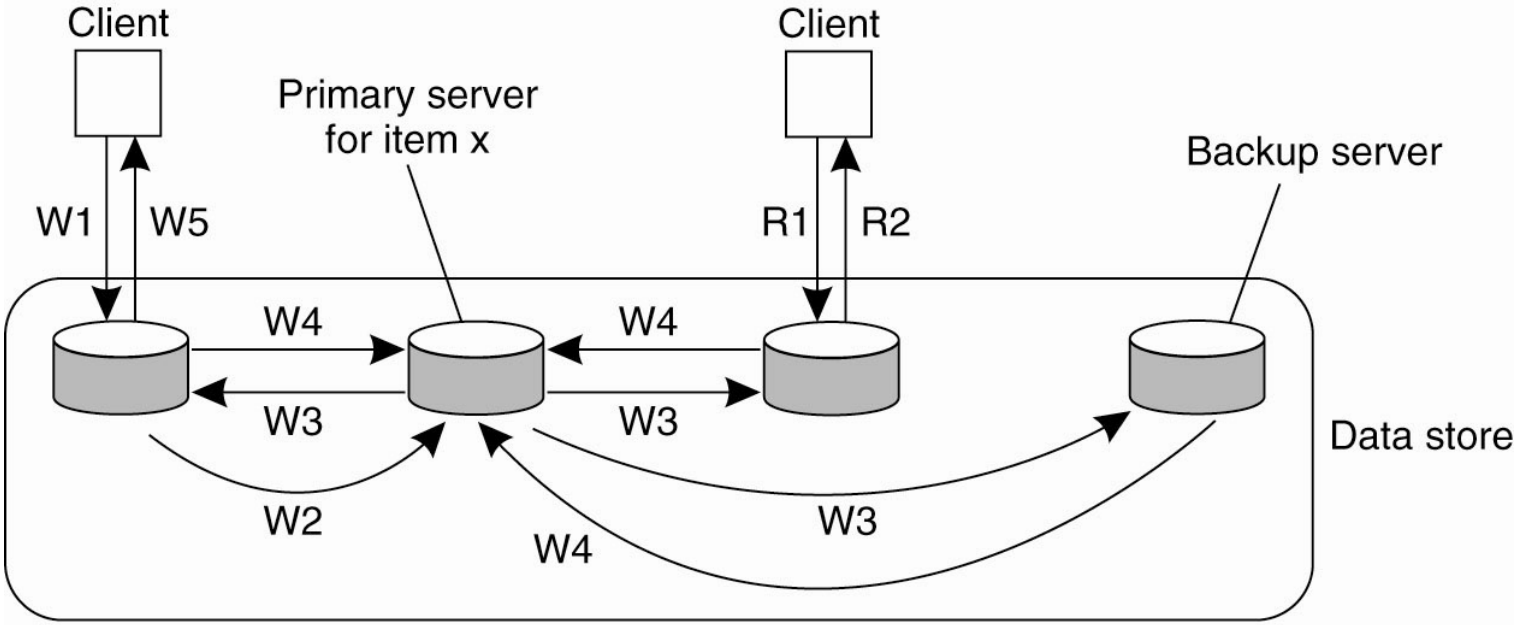
Implementations: Primary-based Protocols

- Implementations tend to prefer simpler consistency models.
- Primary-based protocols are used for implementing sequential consistency.
- In a primary-based protocol, each data item x in the data store has an associated primary, which is responsible for write operations on x

Implementations: Primary-based Protocols

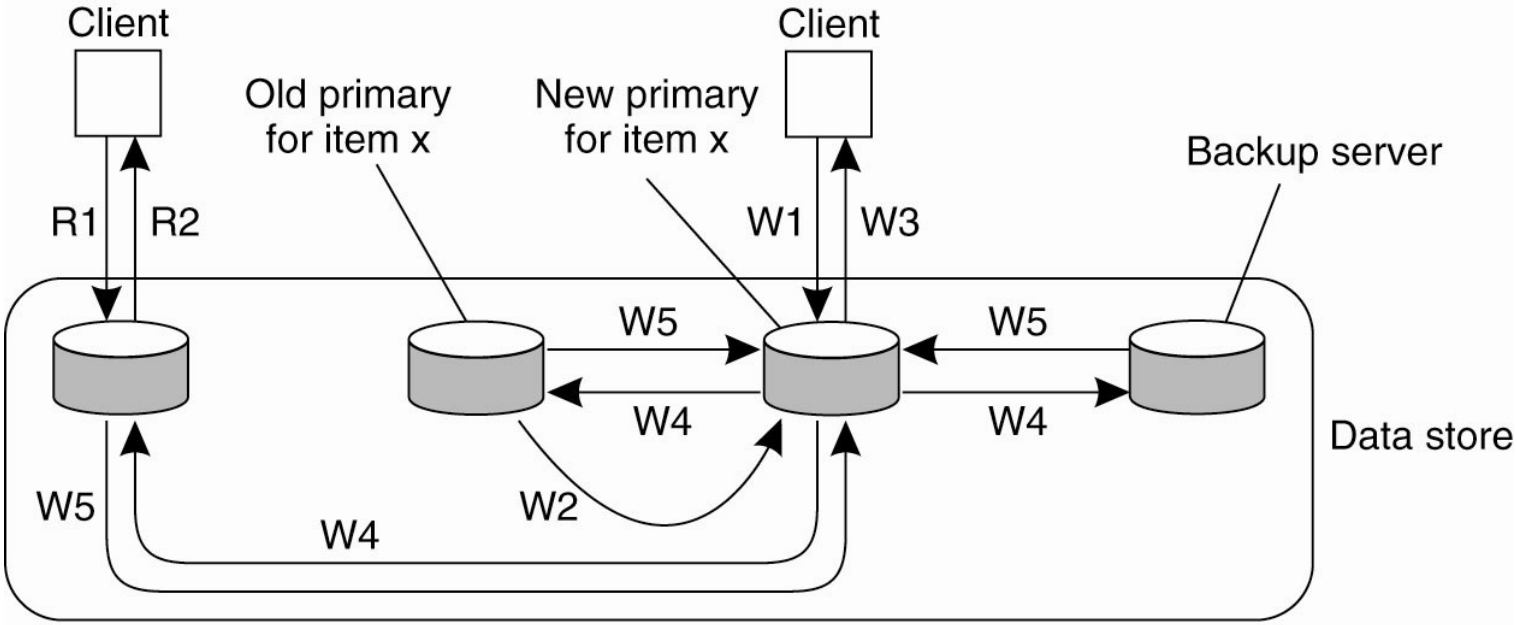
- Implementations tend to prefer simpler consistency models.
- Primary-based protocols are used for implementing sequential consistency.
- In a primary-based protocol, each data item x in the data store has an associated primary, which is responsible for write operations on x
- Primary can be fixed at a remote server or write operations can be carried out locally after moving the primary to the process where the write operation was initiated.

Remote Write Protocol



- The principle of a primary-backup protocol. Implements sequential consistency.
- Non-blocking version: the primary acknowledges after updating its copy and informs backup servers afterwards

Local Write Protocol



- Primary-backup protocol in which the primary migrates to the process wanting to perform an update. Updates have to be propagated back to other replicas.

Replicated-Write Protocols (1)

- Write operations can be carried out at multiple replicas instead of just one.

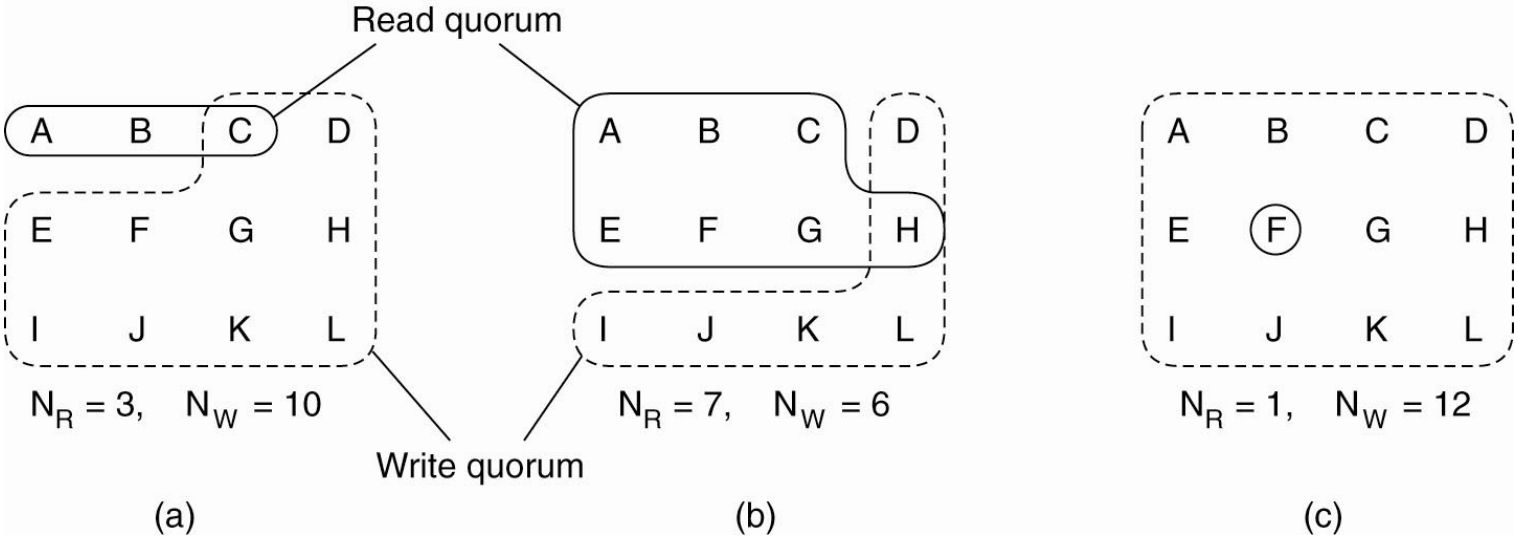
Replicated-Write Protocols (1)

- Write operations can be carried out at multiple replicas instead of just one.
- Active replication: the write operation is forwarded to all replicas. It requires operations to be carried out in the same order everywhere. So it requires totally ordered multicasts using either Lamport timestamps or a central coordinator.

Replicated-Write Protocols (1)

- Write operations can be carried out at multiple replicas instead of just one.
- Active replication: the write operation is forwarded to all replicas. It requires operations to be carried out in the same order everywhere. So it requires totally ordered multicasts using either Lamport timestamps or a central coordinator.
- Quorum-based protocol: the write operation is done by majority voting.

Replicated-Write Protocols (2)



- Quorum-based Protocol. Three examples of the voting algorithm.
  - (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).
    $N_R + N_W > N, N_W > N/2$

Implementing Client-Centric Consistency

- Coming soon!

Exercises

- (Chapter 7) Problems 1, 2, 9, 10, 13, 17, 18, 20.

References

- Eventually Consistent by Werner Vogels (CTO, Amazon)
- How eventual is eventual consistency? posted on Basho Blog
- 10 Lessons from 10 Years of Amazon Web Services by Werner Vogels (CTO, Amazon)