Distributed Systems
CS6421
Consistency and Replication

Prof. Roozbeh Haghnazar

Slides Credit:

Prof. Tim Wood and Prof. Roozbeh Haghnazar

Includes material adapted from Van Steen and Tanenbaum's Distributed Systems book

Last Time…

- Fault Tolerance
  - Types of Failures
  - Two Generals Problem
  - Fault Tolerance Algorithms
  - Centralized FT: Raft/Paxos

This Time…

- Replication and Consistency
  - Why replicate
  - What is consistency?
  - Consistency Models
  - Quorum Replication

DistSys Challenges

- **Heterogeneity**
- Openness
- **Security**
- **Failure Handling**
- Concurrency
- **Quality of Service**
- **Scalability**
- **Transparency**

Any questions about these? You will need to relate your project to them!

Problem

- Given that synchronization and locking is so difficult, do we really need it in a distributed system?
- Is there a better way?

Reasons for Replication

- Data are replicated to increase the reliability of a system.
- Replication for performance
  - Scaling in numbers
  - Scaling in geographical area
- Caveat
  - Gain in performance
  - Cost of increased bandwidth for maintaining replication

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Reasons for Replication

- Reliability.
- Performance.

- Replication is the solution.

How do we keep them up-to-date?
How do we keep them consistent?

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

More on Replication

- Replicas allows remote sites to continue working in the event of local failures.
- It is also possible to protect against data corruption.
- Replicas allow data to reside close to where it is used.
- This directly supports the distributed systems goal of enhanced *scalability*.
- Even a large number of replicated "local" systems can improve performance: think of clusters.

- So, what's the catch?

- It is not easy to keep all those replicas *consistent*.

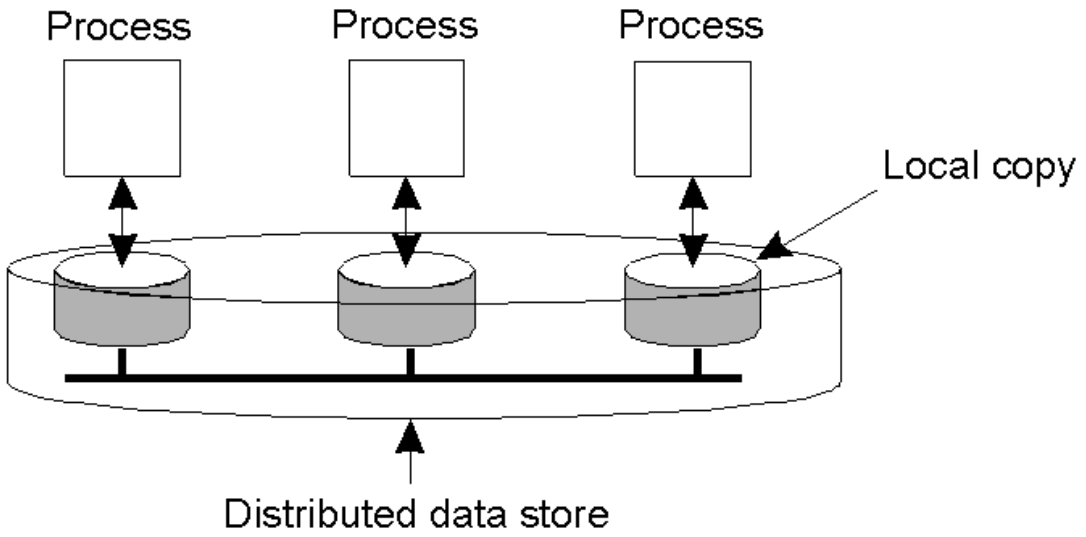
Consistency models

- What is a consistency model?
  - It is an agreement and contract between a distributed data store and related processes.

- Data-Centric
  - Continuous
  - Consistent ordering of operation
    - Sequential
    - Causal
- Client-Centric

Data-Centric Consistency Models

- A data-store can be read from or written to by any process in a distributed system.
- A local copy of the data-store (replica) can support "fast reads".
- However, a write to a local replica needs to be propagated to *all* remote replicas.



CONTINUOUS CONSISTENCY

- There are different ways for applications to specify what inconsistencies they can tolerate.
- Yu and Vahdat [2002] take a general approach by distinguishing three independent axes for defining inconsistencies:
  - deviation in numerical **values** between replicas
  - deviation in **staleness** between replicas
  - deviation with respect to the **ordering** of update operations
- They refer to these deviations as forming ***continuous consistency*** ranges.

EXAMPLE OF NUMERICAL DEVIATIONS

Company	Stock price
Apple	$1100.00

+0.01
+0.03

Company	Stock price
Apple	$1100.00

+0.03

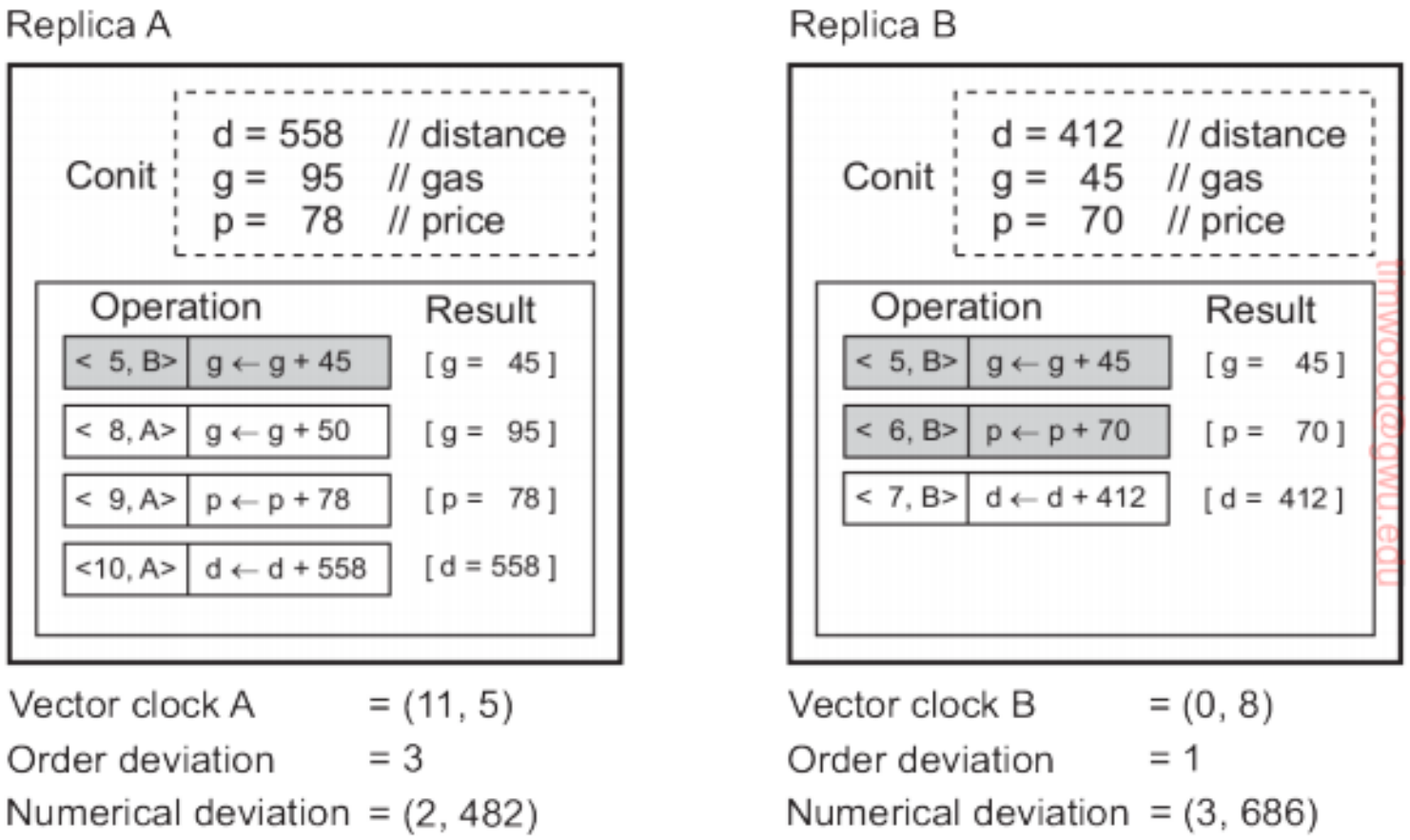
Company	Stock price
Apple	$1100.00

+0.03

It can be done by:
- Percentage
- Number of updates (web cash)

Continuous Consistency

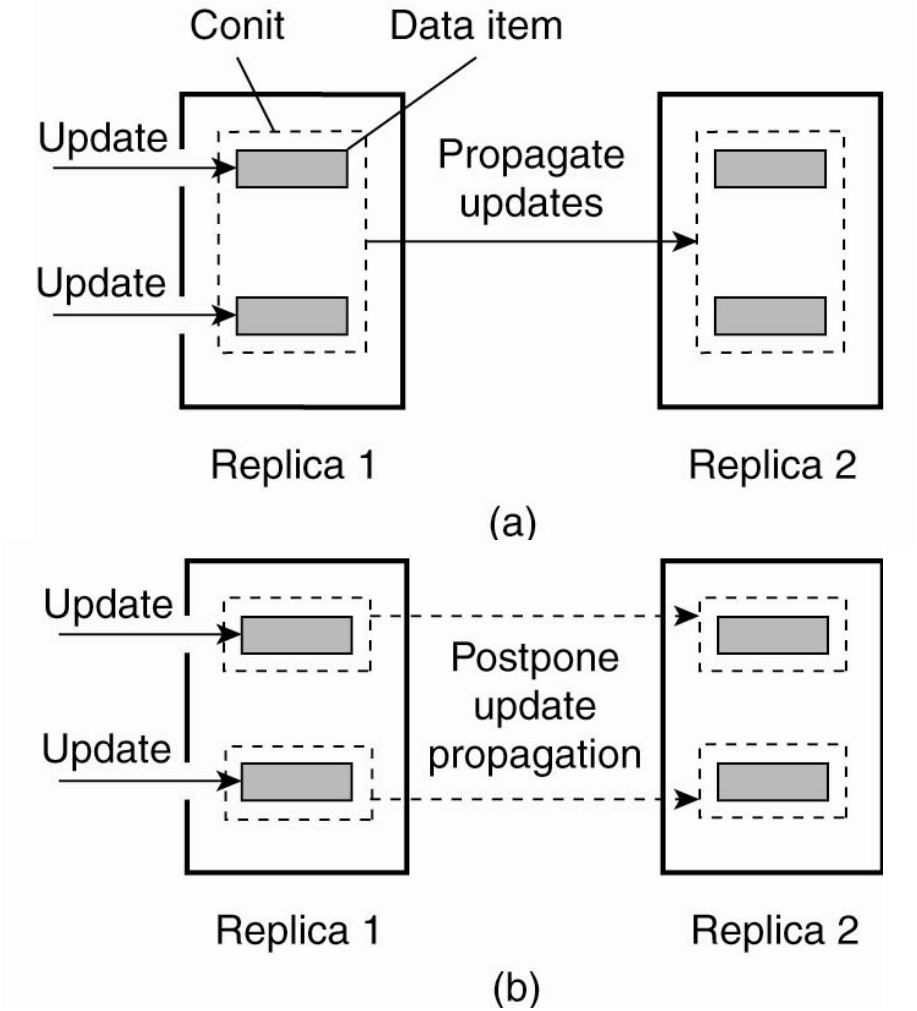
- Each replica server maintains a two-dimensional vector clock



Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved. 0-13-239227-5

Continuous Consistency

- Choosing the appropriate granularity for a conit.
  - (a) Two updates lead to update propagation.
  - (b) No update propagation is needed



Consistent ordering of operations

- **Sequential consistency**
- **Causal consistency**
- **Grouping operations**

- There is a huge body of work on data-centric consistency models from the past decades. An important class of models comes from the field of parallel programming.

- Confronted with the fact that in parallel and distributed computing multiple processes will need to share resources and access these resources simultaneously, researchers have sought to express the semantics of concurrent accesses when shared resources are replicated.

Prof. Tim Wood & Prof. Roozbeh Haghnazar

Consistency Models

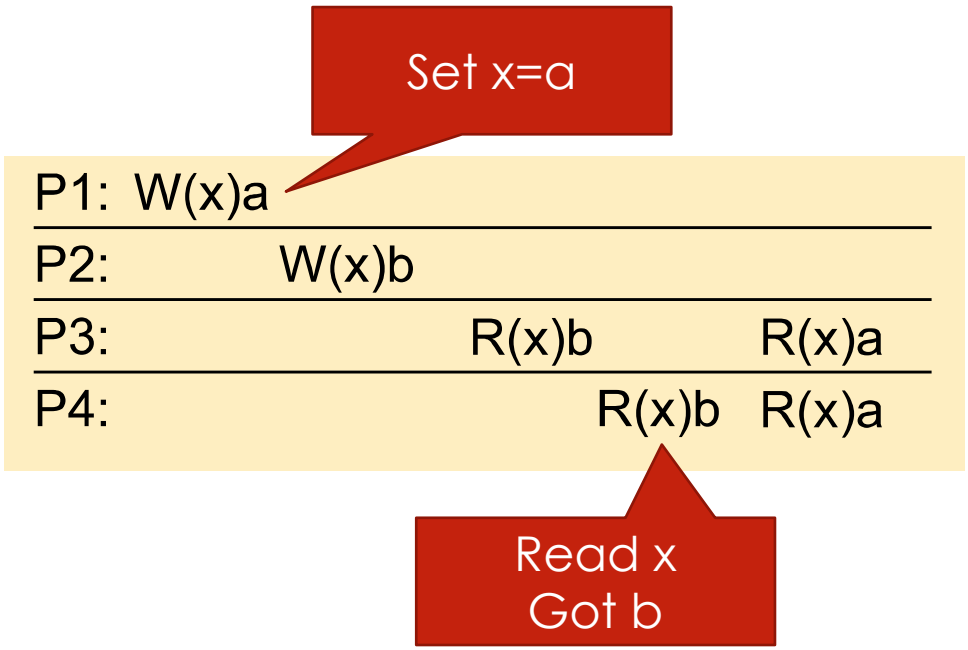
Consistency versus coherence

- A consistency model describes what can be expected when multiple processes concurrently operate on a set of data. The set is then said to be consistent if it adheres to the rules described by the model.

- Where data consistency is concerned with a set of data items, coherence models describe what can be expected to hold for only a single data item [Cantin et al., 2005].

- In this case, we assume that a data item is replicated; it is said to be coherent when the various copies abide to the rules as defined by its associated consistency model.

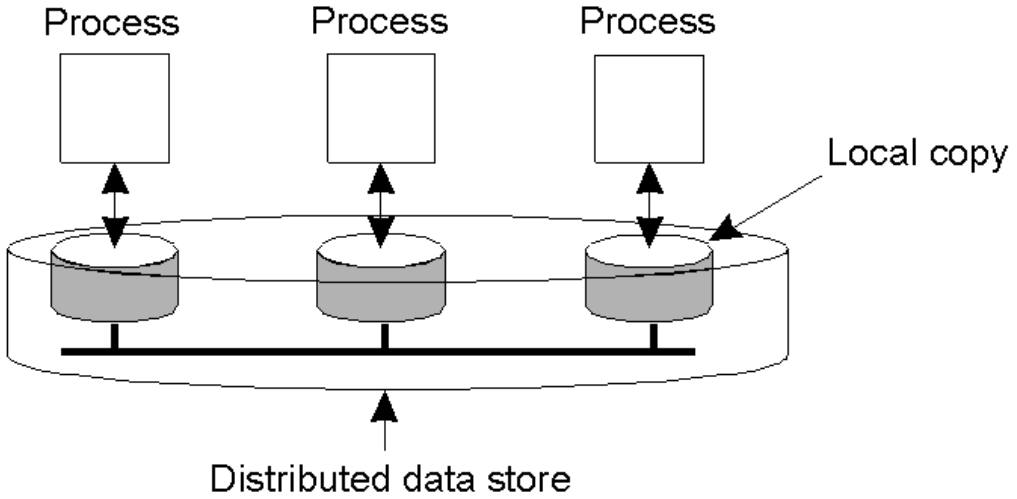
Consistency Model Diagram Notation



$W_i(x)a$ – a write by process 'i' to item 'x' with a value of 'a'. That is, 'x' is set to 'a'.

$R_i(x)b$ – a read by process 'i' from item 'x' producing the value 'b'. That is, reading 'x' returns 'b'.

Time moves from left to right in all diagrams.



Sequential Consistency

- The result of any execution is the same as if the operations of **all processes** were executed in some sequential order, and
- The operations of each individual process appear in this sequence in the order specified by its program.

Any ordering of reads/writes is fine, but all processes must see the same ordering

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

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

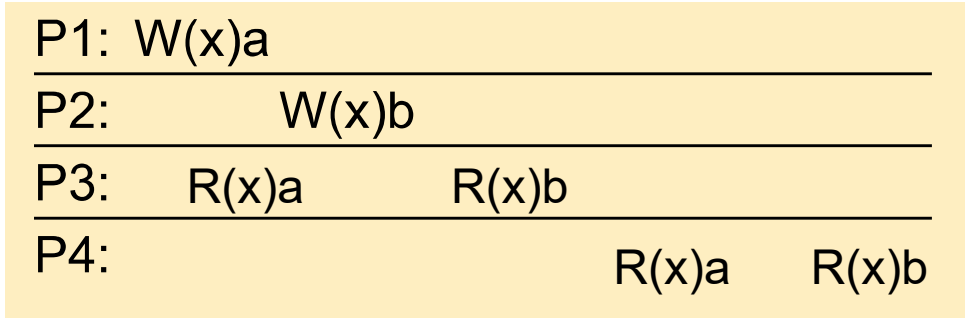
P1:	W(x)a				
P2:		W(x)b			
P3:				R(x)b	R(x)a
P4:				R(x)a	R(x)b

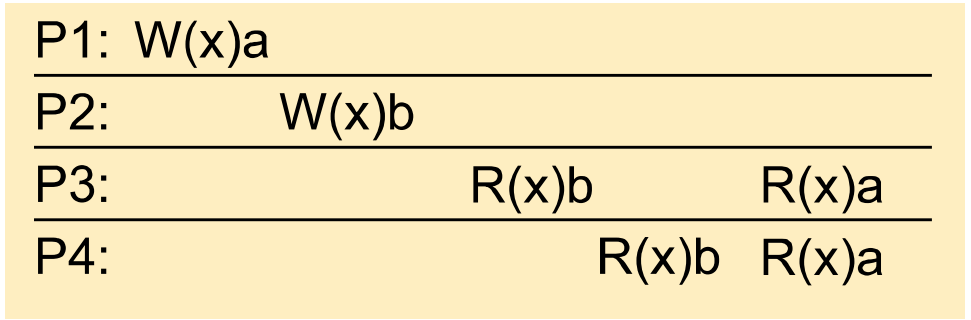
Which are sequentially consistent?

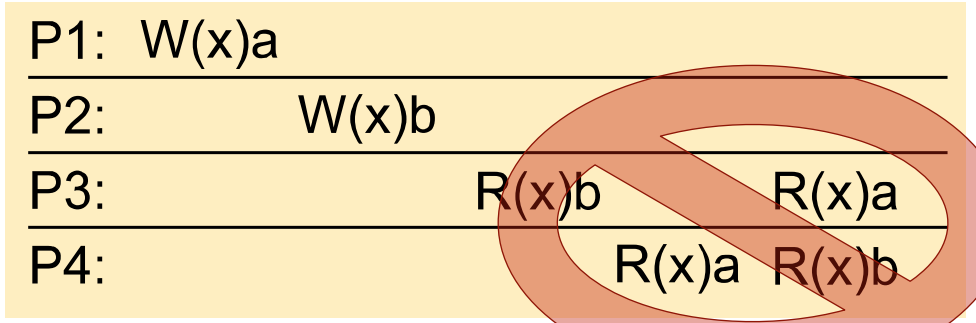
Sequential Consistency

- The result of any execution is the same as if the operations of **all processes** were executed in some sequential order, and
- The operations of each individual process appear in this sequence in the order specified by its program.

Any ordering of reads/writes is fine, but all processes must see the same ordering







Which are sequentially consistent?

Causal Consistency

- 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 by different processes.

Reading a value means your future writes may be causally related to that operation!

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

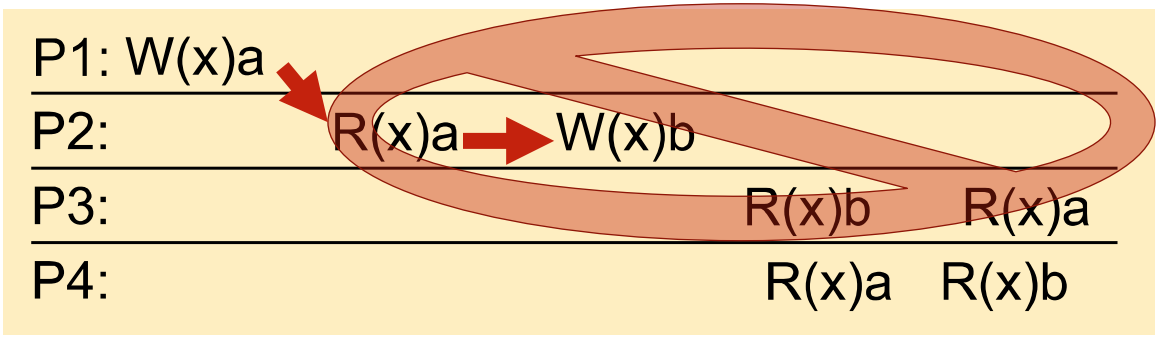
P1: W(x)a				
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

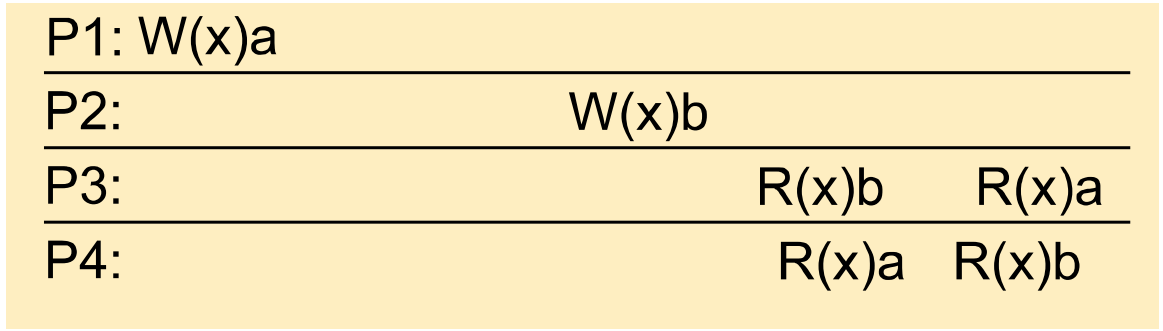
P1: W(x)a				
P2:		W(x)b		R(x)c
P3:	W(x)c		R(x)b	R(x)a
P4:			R(x)a	R(x)b

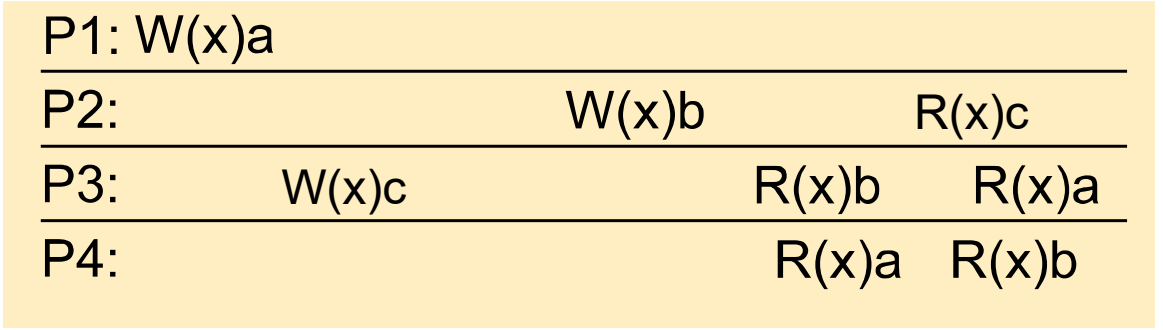
Causal Consistency

- 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 by different processes.

Reading a value means your future writes may be causally related to that operation!





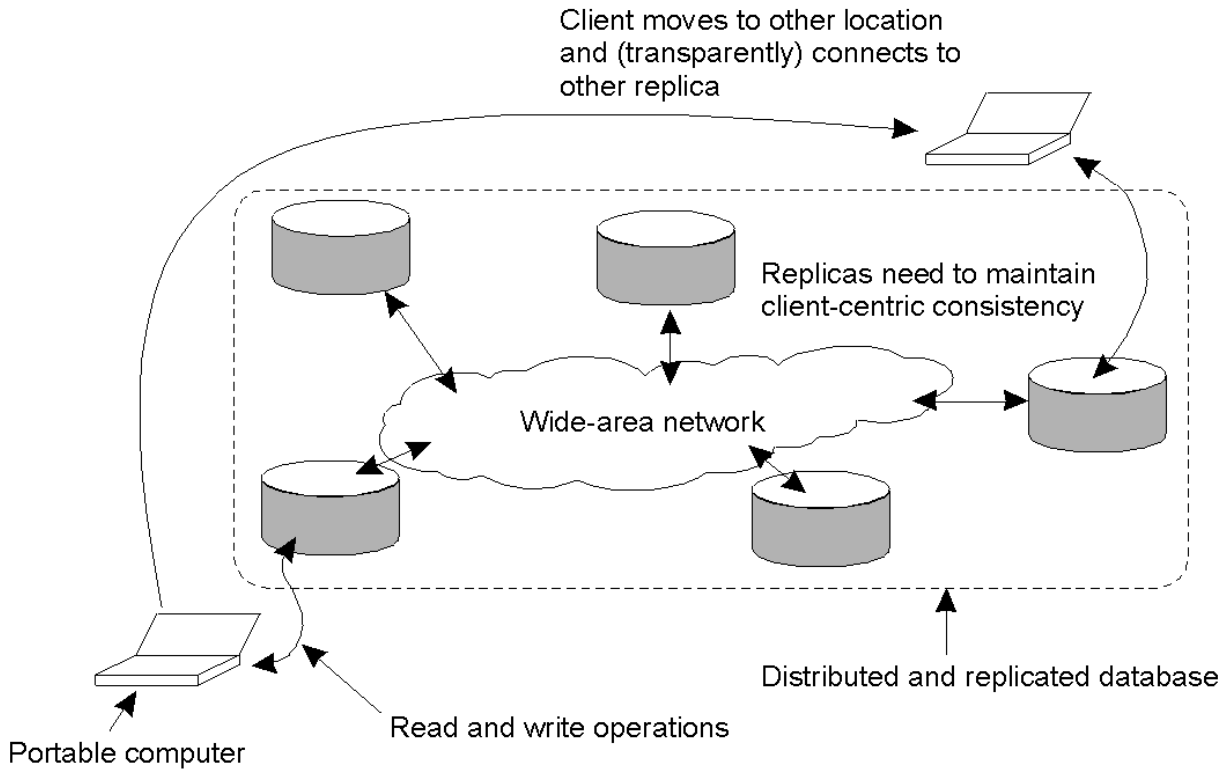


EVENTUAL CONSISTENCY

- The only requirement is that all replicas will *eventually* be the same.
- All updates must be guaranteed to propagate to all replicas … *eventually*!
- This works well if every client always updates the same replica.
- Things are a little difficult if the clients are *mobile*.

Eventual Consistency: Mobile Problems

- The principle of a mobile user accessing different replicas of a distributed database.
- When the system can guarantee that a single client sees accesses to the data-store in a consistent way, we then say that "client-centric consistency" holds.



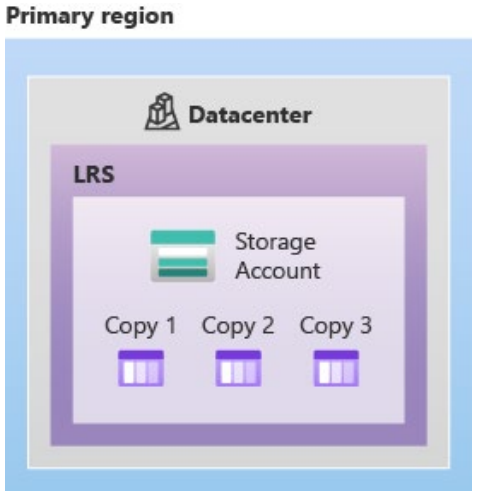
Prof. Tim Wood & Prof. Roozbeh Haghnazar

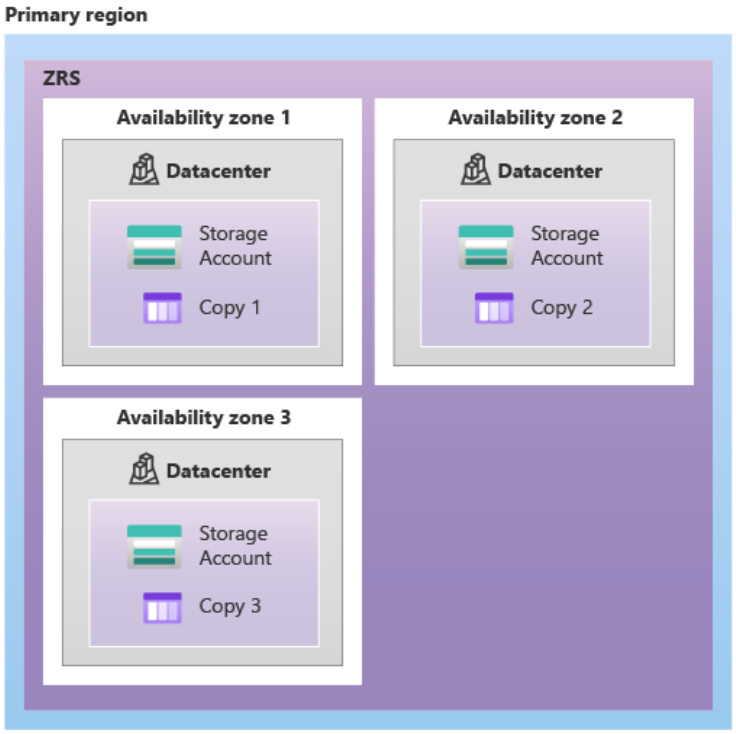
Replication In Azure Storage

REDUNDANCY IN THE PRIMARY REGION

Data in an Azure Storage account is always replicated three times in the primary region. Azure Storage offers two options for how your data is replicated in the primary region:

- **Locally redundant storage (LRS)** copies your data synchronously three times within a single physical location in the primary region. LRS is the least expensive replication option, but is not recommended for applications requiring high availability or durability. LRS provides at least 99.999999999% (11 nines) durability of objects over a given year
- **Zone-redundant storage (ZRS)** copies your data synchronously across three Azure availability zones in the primary region. For applications requiring high availability, Microsoft recommends using ZRS in the primary region, and also replicating to a secondary region. ZRS offers durability for Azure Storage data objects of at least 99.9999999999% (12 9's) over a given year

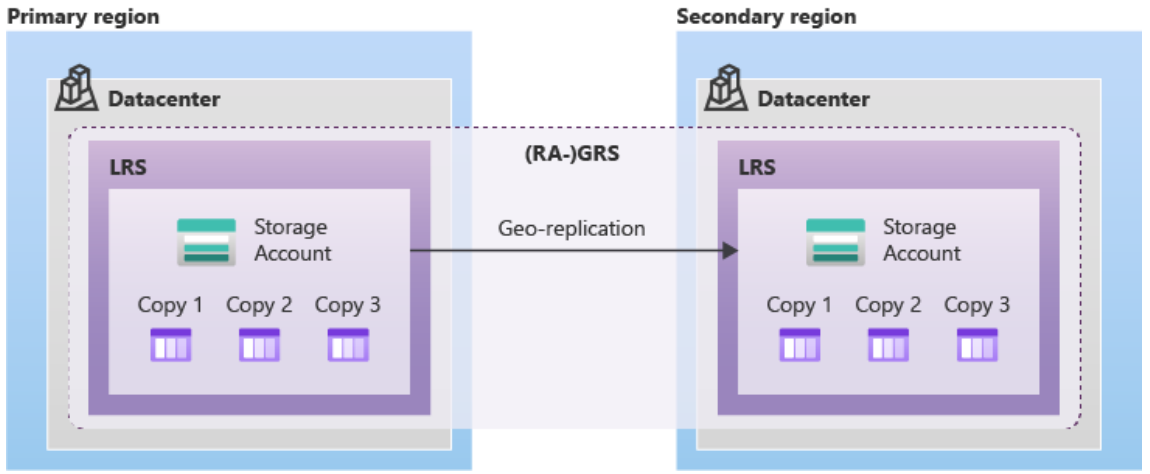


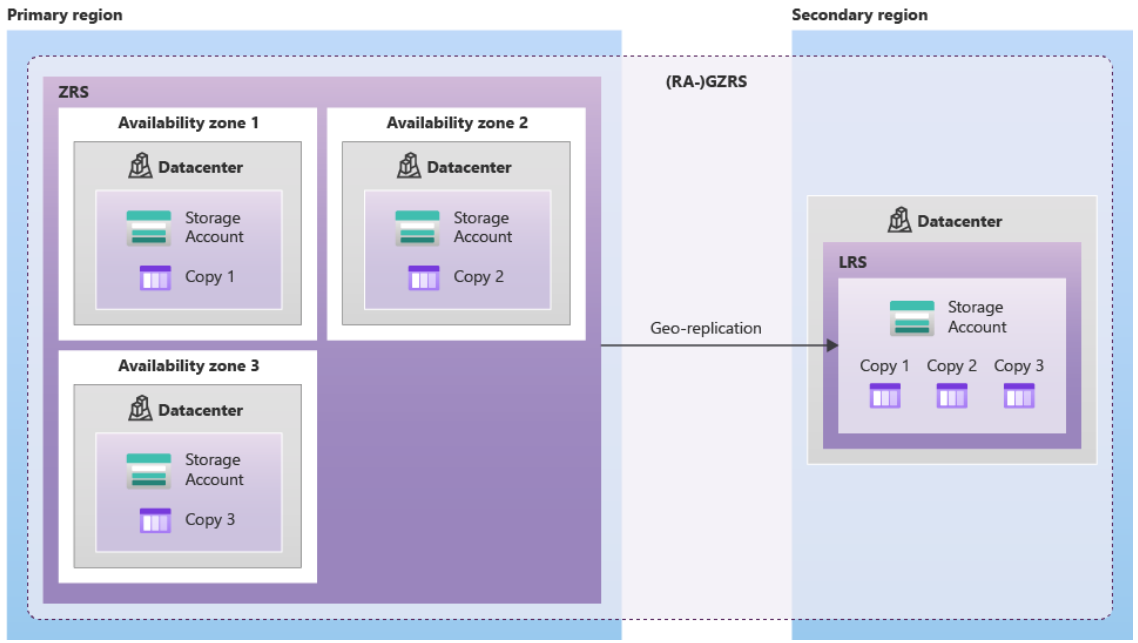


REDUNDANCY IN A SECONDARY REGION

For applications requiring high durability, you can choose to additionally copy the data in your storage account to a secondary region that is hundreds of miles away from the primary region. If your storage account is copied to a secondary region, then your data is durable even in the case of a complete regional outage or a disaster in which the primary region isn't recoverable:

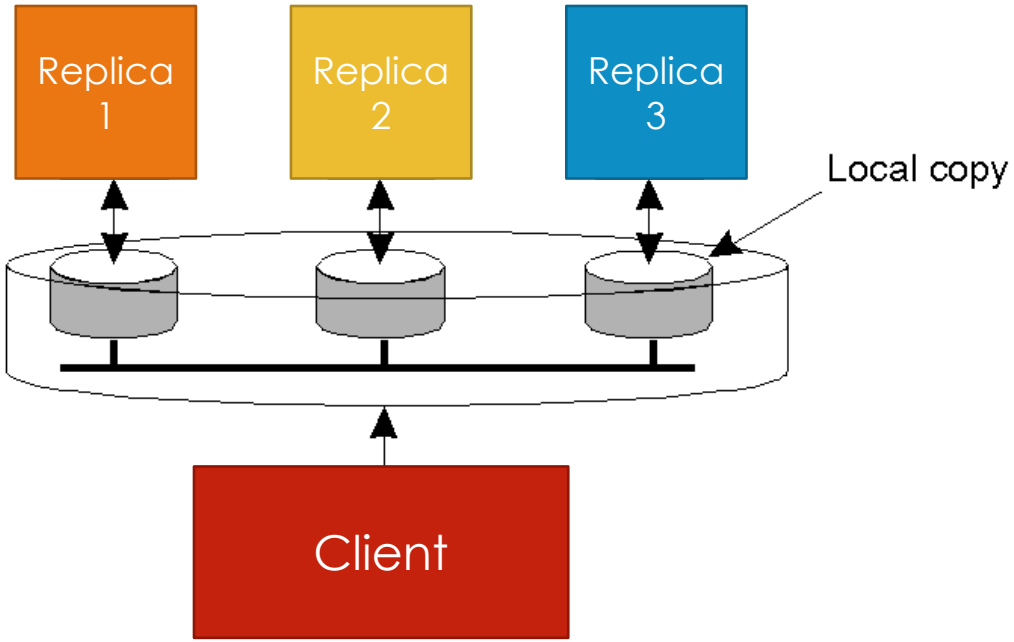
- **Geo-redundant storage (GRS)** copies your data synchronously three times within a single physical location in the primary region using LRS. It then copies your data asynchronously to a single physical location in the secondary region. Within the secondary region, your data is copied synchronously three times using LRS. GRS offers durability for Azure Storage data objects of at least 99.99999999999999% (16 9's) over a given year
- **Geo-zone-redundant storage (GZRS)** copies your data synchronously across three Azure availability zones in the primary region using ZRS. It then copies your data asynchronously to a single physical location in the secondary region. Within the secondary region, your data is copied synchronously three times using LRS. GZRS offers durability for Azure Storage data objects of at least 99.99999999999999% (16 9's) over a given year





Client-Centric Consistency Models

- **Monotonic Reads**: Reads never go backwards
- **Monotonic Writes**: Writes never go backwards
- **Read your Writes**: My own writes must be visible
- **Writes follow reads**: If a write is based on a read, it must happen after it



What does the client see?

Client-Centric Consistency Models

- **Monotonic Reads**: If a process reads the value of a data item *X*, any subsequent read operation on *X* by that process will always return that same value or a more recent value

Example

Automatically reading your personal calendar updates from different servers. Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

Example

Reading (not modifying) incoming mail while you are on the move. Each time you connect to a different e-mail server, that server fetches (at least) all the updates from the server you previously visited.

Client-Centric Consistency Models

- **Monotonic Writes**: A write operation by a process on a data item *X* is completed before any successive write operation on *X* by the same process.

Example

Updating a program at server S_2, and ensuring that all components on which compilation and linking depends, are also placed at S_2.

Example

Maintaining versions of replicated files in the correct order everywhere (propagate the previous version to the server where the newest version is installed).

Client-Centric Consistency Models

- **Read your Writes**: 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.

Example

Updating your Web page and guaranteeing that your Web browser shows the newest version instead of its cached copy.

Client-Centric Consistency Models

- **Writes follow reads**: 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.

Example

If I read and then comment on an article, nobody should see my comment until after they see the article

Prof. Tim Wood & Prof. Roozbeh Haghnazar

Quorum Replication

Problem

- In a distributed system, whenever a server takes any action, it needs to ensure that in the event of a crash the results of the actions are available to the clients. → **Replication**
- how many other servers need to confirm the replication before the original server can be confident that the update is fully recognized?
  - If the original server waits for too many replications, then it will respond slowly - **reducing liveness.**
  - But if it doesn't have enough replications, then the update could be lost - a **failure of safety.**
- It's critical to balance between the overall system performance and system continuity.

Quorum Based Systems

- A quorum is the minimum number of votes that a distributed transaction has to obtain in order to be allowed to perform an operation in a distributed system. A quorum-based technique is implemented to enforce consistent operation in a distributed system.

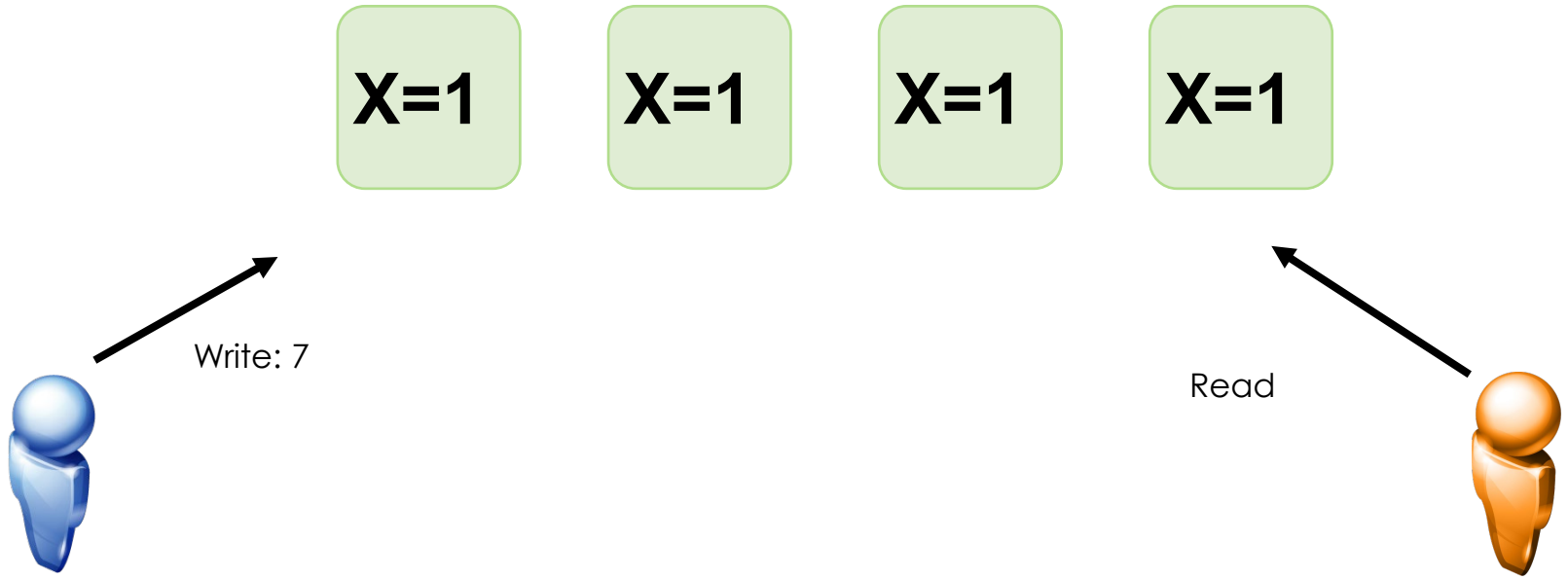
- Quorum: a set of responses that agree with each other of a particular size

  - A cluster agrees that it's received an update when a majority of the nodes in the cluster have acknowledged the update. **We call this number a quorum**.

- Crash fault tolerance: Need a quorum of **1**
  - **f** others can fail (thus need f+1 total replicas)
- Data fault tolerance: Need a quorum **f+1**
  - **f** others can fail (thus need 2f+1 total replicas)
  - Need a majority to determine correctness

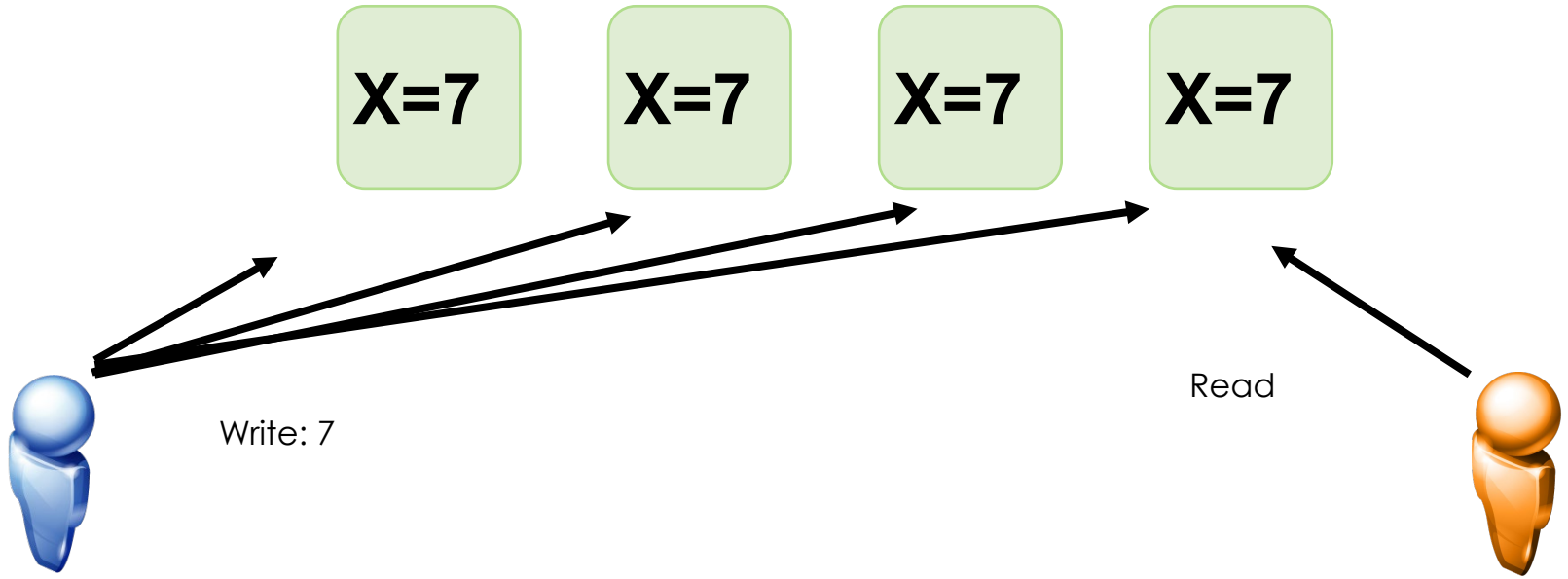
QUORUM

- 4 Replicas
  - Some nodes might be temporarily offline
- How many replicas to send to for a read or write?
  - Must wait for a response from each one



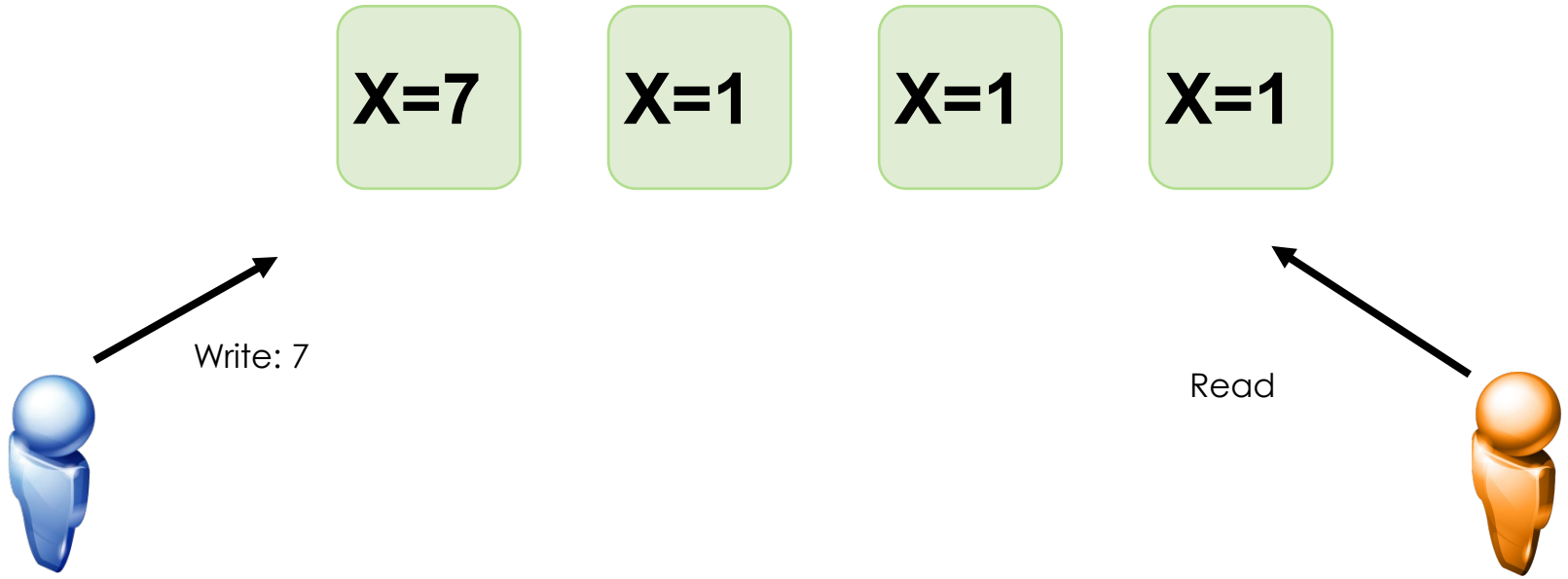
QUORUM

- Q=4



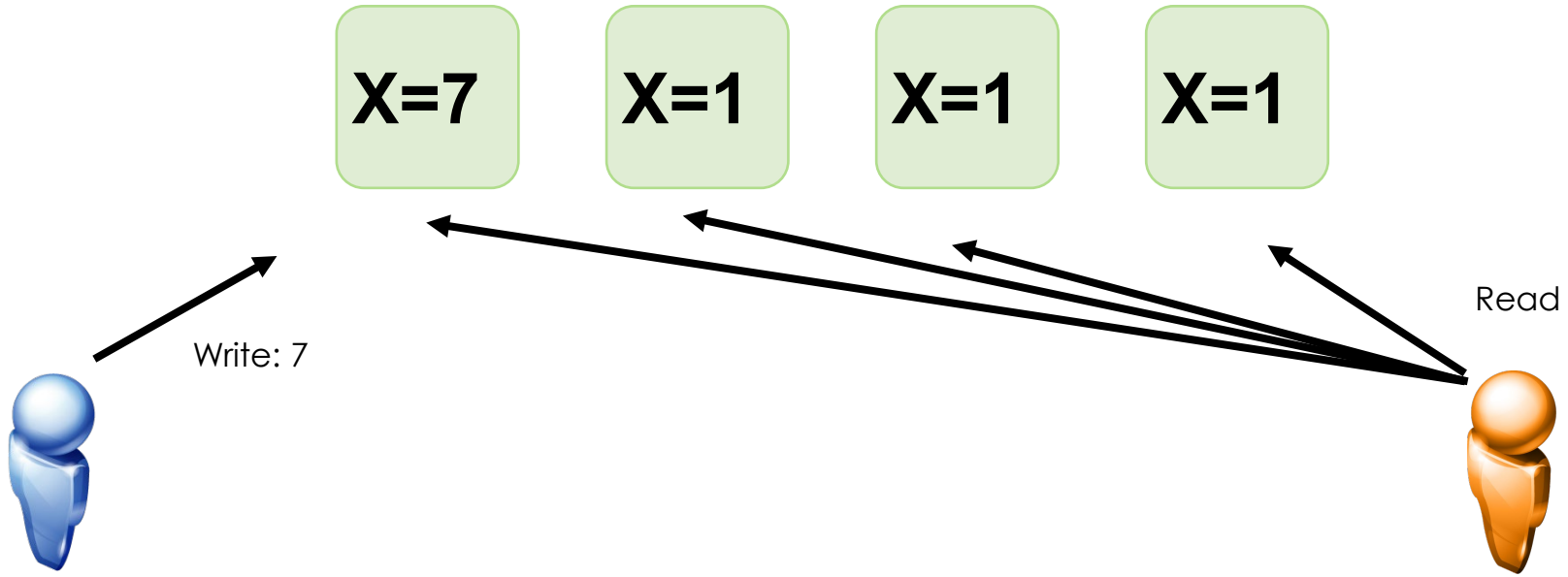
QUORUM

- Q=1



Quorum + Tradeoff

- Q=1



DYNAMO DB

- Object Store from Amazon
  - Technical paper at SOSP 2007 conference (top OS conference)
- Stores **N** replicas of all objects
  - But a replica could be out of date!
  - Might be saved across multiple data centers
  - Gradually pushes updates to all replicas to keep in sync
- When you read, how many copies, **R**, should you read from before accepting a response?
- When you write, how many copies, **W**, should you write to before confirming the write?

DYNAMO DB

- Read and Write Quorum size: how will the system behave?

- R=1→ *I want to just read from 1 server (**randomly**) and get the most updated data →read optimization*
- W=1→ *I want to just write to 1 server and get the most updated data→read optimization*
- R=1, W=1 →?
- R = N/2+1 → ?
- R=1, W = N → ?
- R=N, W=1 → ?

DYNAMO DB

- Read and Write Quorum size: how will the system behave?

- R=1→ *I want to just read from 1 server (**randomly**) and get the most updated data →read optimization*
- W=1→ *I want to just read from 1 server and get the most updated data→read optimization*
- R=1, W=1 →?
- R = N/2+1 → Most consistency → if N=10, W=?
- R=1, W = N → ?
- R=N, W=1 → ?

R=10/2 + 1 =6 → W= N/2 = 5

DYNAMO DB

- Read and Write Quorum size: how will the system behave?
  - The following conditions guarantee read consistency but with different approaches

- R=1, W = N → Fast Read

- R=N, W=1 → Fast Write

R+W>N → Always have some overlaps.

Dynamo DB

- Read and Write Quorum size:

- R=1 — fastest read performance, no consistency guarantees
- W = 1 — fast writes, reads may no be consistent
- R = N/2+1 (reading from majority)
- R=1, W = N - slow writes, but reads are consistent
- R=N, W=1 - slow reads, fast writes, consistent
- Standard: N=3, R=2, W=2
  - Ensures overlap

QUORUM

- How do N, R, and W affect:
- **Performance:** small R,W → better performance
- **Consistency:** R+W>N → Consistency
- **Durability:** if W ~ N → better durability
- **Availability:** Larger N → More availability

DynamoDB lets the user tune these for their needs

QUORUM

- How do N, R, and W affect:
- **Performance:**
  - low R or W -> higher performance
  - for a fixed R or W: higher N gives higher performance
  - higher N means more synchronization traffic
- **Consistency:**
  - R + W > N — guarantees consistency
  - R+w << N — much less likely to be consistent
- **Durability:**
  - N=1 vs N=100, more N = more durability
- **Availability:**
  - Higher N or W => higher availability