Consistency and Replication

Thomas Schwarz, SJ

Reasons for Replication

- Reasons for Replication
  - Fault Tolerance
    - Server Failure
    - Loss of Communication
  - Performance:
    - Scaling
      - Bring objects neared to user (Akamai)
    - Avoid congested objects

Reasons Against Replication

- Storage costs
- Transparency
  - Updates to replicated objects are difficult to order
  - Users can get stale data
  - Users can get inconsistent data

Reasons Against Replication

- Replicas need to be consistent
  - Strict (Tight) Consistency — costly
  - Eventual Consistency — deal with stale data

Replica Management

- Concurrent access to objects
  - Simple solution: No protection at the object
    - Application using an object needs to manage concurrency
  - Simple solution: Protection at the object
    - Object has mechanism for mutual exclusion
    - Example: Java
      - Declare object's methods *synchronized*
      - Only one thread is allowed to proceed with a synchronized method
      - Other threads are blocked

Replica Management

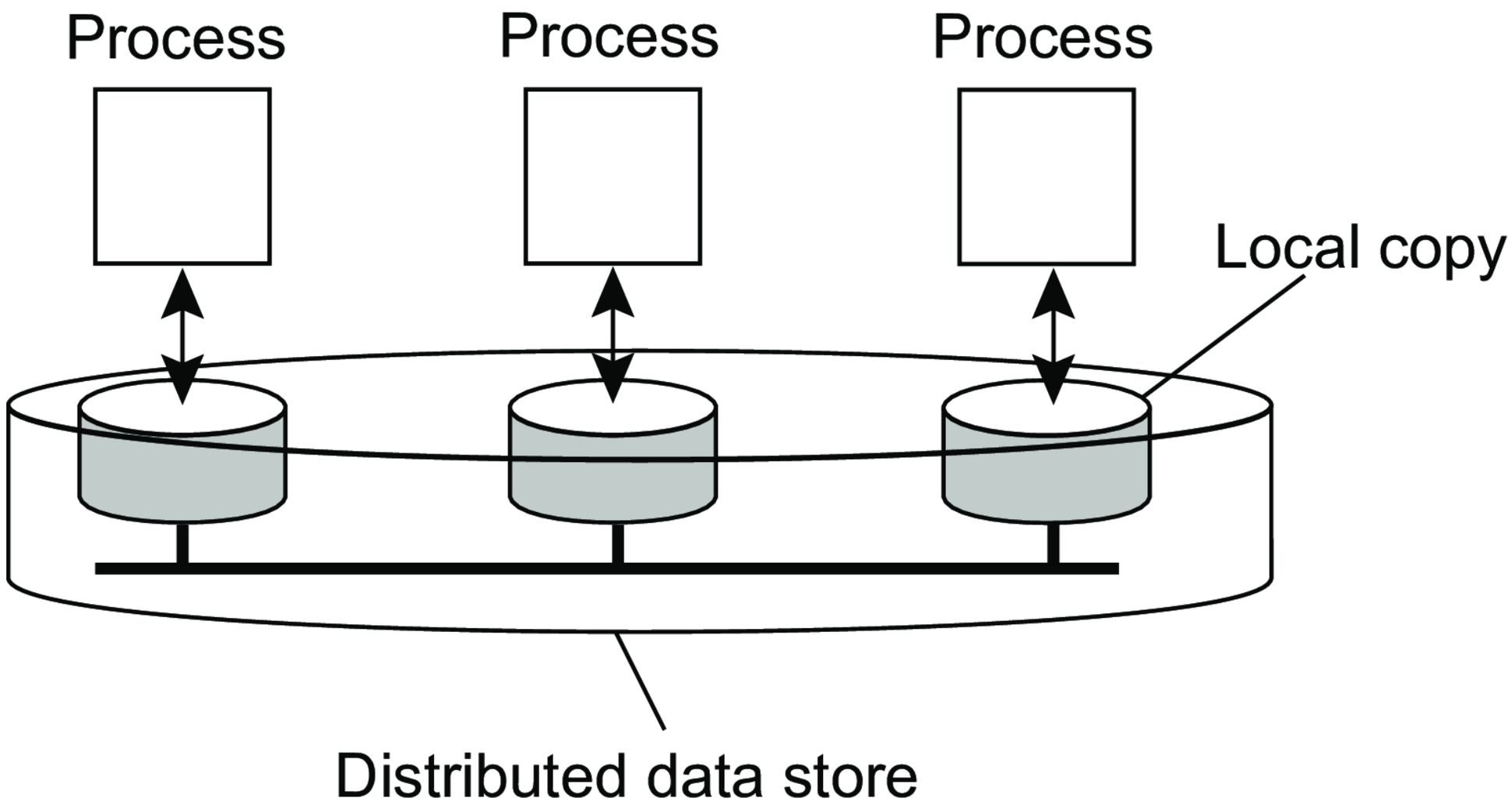
- Concurrent access to replicated objects
- Consistency handled
  - At object (*Replication Awareness)*:
    - Object aware that it is replicated
    - Handles updates at replica by itself
  - By the distributed system (more common)
    - Example: Corba

Replica Management

- Replication and Scaling
  - Replication and caching are scaling techniques
  - Trade-off between:
    - Keeping objects up to date
      - More network traffic
    - Keeping load at objects low
      - Less long-haul network traffic

Consistency Models

- Model a *Distributed Data Store*



Consistency Models

- To keep replicas consistent, we generally need to ensure that all conflicting operations are done in the the same order everywhere
  - Conflicting operations: From the world of transactions
    - Read–write conflict: a read operation and a write operation act concurrently
    - Write–write conflict: two concurrent write operations
- Issue
  - Guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability.
  - Solution: weaken consistency requirements so that hopefully global synchronization can be avoided

Consistency Models

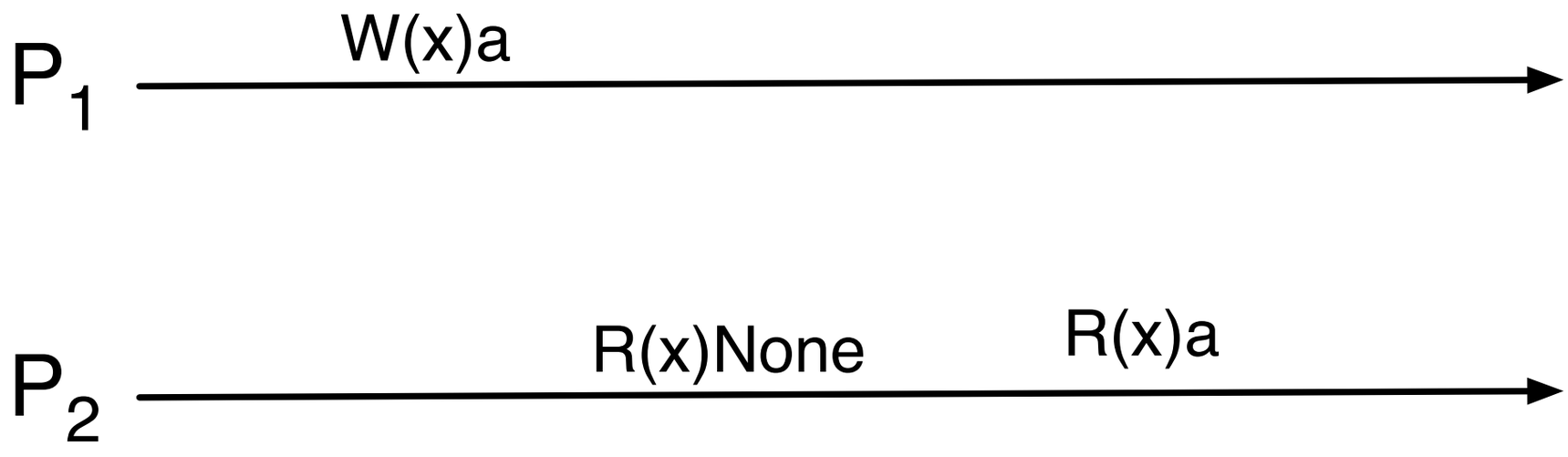
- Consistency model
  - A contract between a (distributed) data store and processes
  - Data store specifies precisely what the results of read and write operations are in the presence of concurrency.

Consistency Models

- Tight consistency is not scalable
- Brewer's Theorem:
  - Can only achieve two of the three goals
    - Consistency
    - Availability
    - Partition tolerance

Consistency Models

- Read and write operations
  - $W_i(x)a$: Process P_i writes value a to x
  - $R_i(x)b$: Process P_i reads value b from x
    - All data items initially have value NIL
- We omit the index when possible and draw according to time (x-axis):

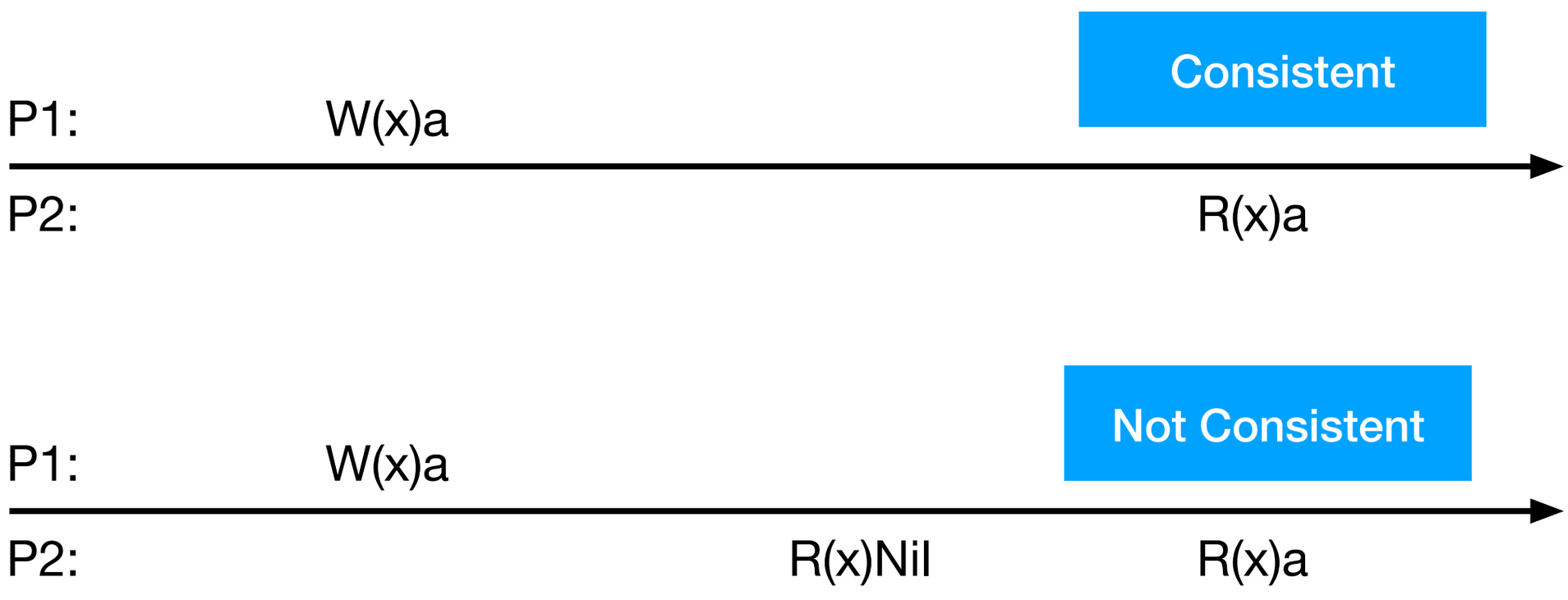


Consistency Models

- Strict consistency
  - *Any read on a data item x returns a value corresponding to the result of the most recent write on x*
- Assumes notion of global time
  - Relax global-time-assumption by
    - Dividing time into consecutive, non-overlapping intervals
    - Each operation takes place during a single interval
    - Each interval has a global timestamp

Consistency Models

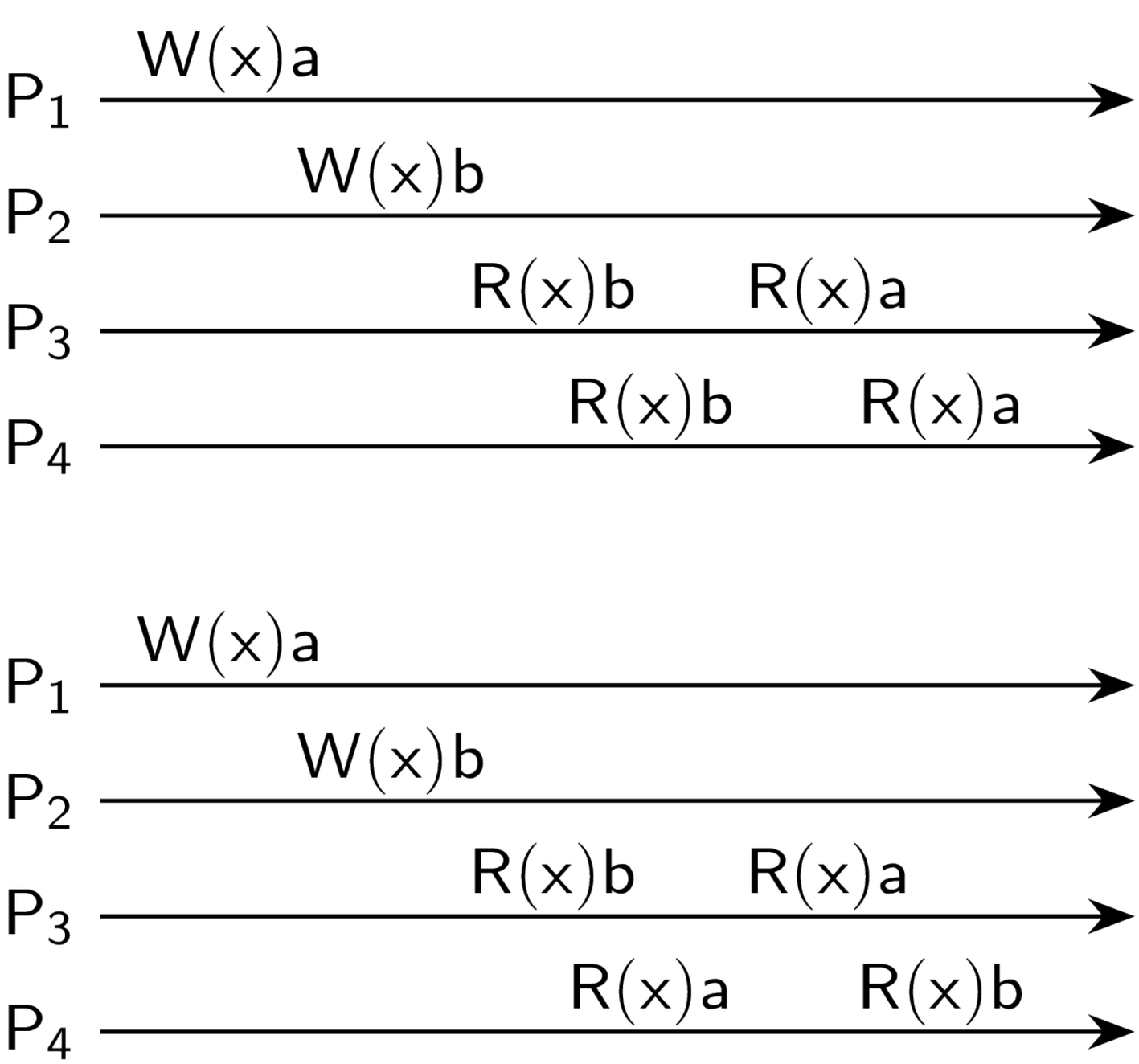
- Examples for strict consistency



Consistency Models

- Sequential Consistency (Lamport)
  - *The results of any execution is the same as if the 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*
- All processes see the same order of interleaving

Consistency Models



Consistency Models

- Linearizable (Herlihy and Wing, 1991)
  - Weaker than strict consistency, but stronger than sequential consistency
  - Assumes a globally available clock
    - with finite precision
  - Assigns timestamp to all operations

Consistency Models

- Data store is linearizable
- *The result of any execution is the same as if the operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in the sequence specified by their time stamps. In addition, if the time stamp of one operation precedes the time stamp of the other operation for sure, then the first operation happens before the last operation.*

Consistency Models

- Example
  - Variables initialized to 0

P1	P2	P3
`x=1;`	`y=1;`	`z=1;`
`print(y,z)`	`print(x,z)`	`print(x,y)`

Consistency Models

- Signature: output of P1 concatenated with output of P2 concatenated with output of P3

P1

```
x=1;
print(y,z)
```

P2

```
y=1;
print(x,z)
```

P3

```
z=1;
print(x,y)
```

```
x=1;
print(y,z);
y=1
print(x,z);
z=1
print(x,y)


//prints 001011
//signature 001011
```

Consistency Models

P1

```
x=1;
print(y,z)
```

P2

```
y=1;
print(x,z)
```

P3

```
z=1;
print(x,y)
```

```
x=1;
y=1;
print(x,z);
print(y,z);
z=1;
print(x,y)


//prints 101011
//signature 101011
```

Consistency Models

P1	P2	P3
`x=1;`	`y=1;`	`z=1;`
`print(y,z)`	`print(x,z)`	`print(x,y)`

```
y=1;
z=1;
print(x,y);
print(x,z);
x=1;
print(y,z);



//prints 010111
//signature 110101
```

Consistency Models

```
y=1;
x=1;
z=1;
print(x,z);
print(x,z);
print(y,z);
```

```
//prints 111111
//signature 111111
```

Consistency Models

- Not all possible 64 signature patterns are valid
  - 000000 needs processes to print before incrementing
  - 001001 is also impossible
    - 00 means that y = z = 0 and x=1 when P1 prints
      - So P1 finishes before P2 and P3 start
    - Next 10 means that P2 starts before P3 starts
    - Next 01 means that P3 must finish before P1 starts
    - Contradiction

Consistency Models

- There are 6! = 720 ways of ordering the statements, but many violate execution order
- Remaining 90 produce various write sequences and signatures while maintaining sequential consistency
- Any process interacting with the data store needs to function with the < 64 valid signatures

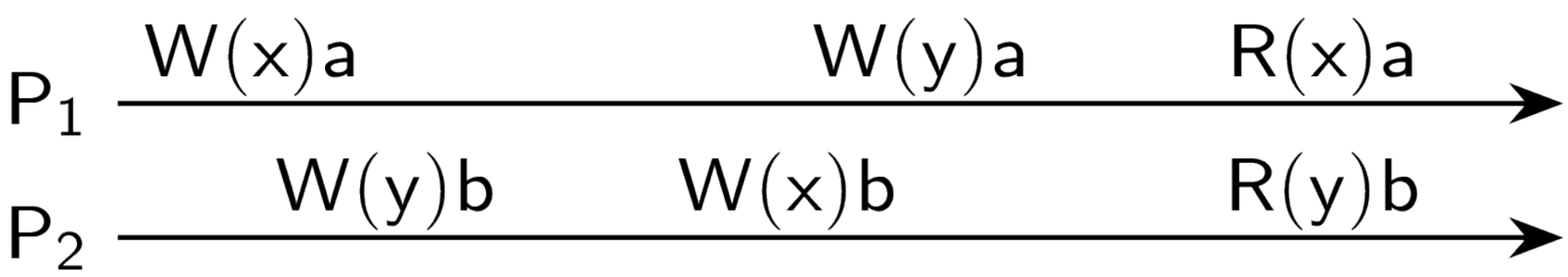
Consistency Models

Execution 1	Execution 2	Execution 3	Execution 4
P_1: x ← 1;	P_1: x ← 1;	P_2: y ← 1;	P_2: y ← 1;
P_1: print(y,z);	P_2: y ← 1;	P_3: z ← 1;	P_1: x ← 1;
P_2: y ← 1;	P_2: print(x,z);	P_3: print(x,y);	P_3: z ← 1;
P_2: print(x,z);	P_1: print(y,z);	P_2: print(x,z);	P_2: print(x,z);
P_3: z ← 1;	P_3: z ← 1;	P_1: x ← 1;	P_1: print(y,z);
P_3: print(x,y);	P_3: print(x,y);	P_1: print(y,z);	P_3: print(x,y);
Prints: 001011	*Prints:* 101011	*Prints:* 010111	*Prints:* 111111
Signature: 0 0 1 0 1 1	*Signature:* 1 0 1 0 1 1	*Signature:* 1 1 0 1 0 1	*Signature:* 1 1 1 1 1 1
(a)	(b)	(c)	(d)

Outcomes of sequentially consistent orderings

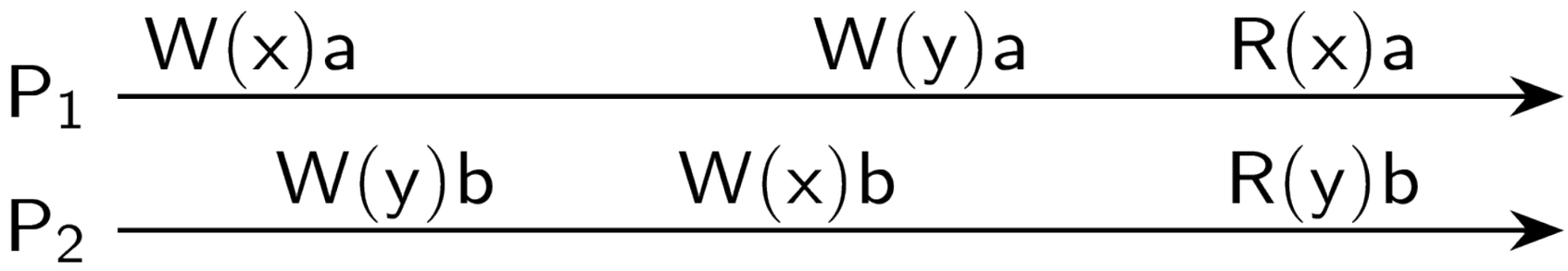
Consistency Models

- Sequential consistency is difficult to understand



Sequential consistent for x, sequential consistent for y, but not sequential consistent for both x and y

Consistency Models

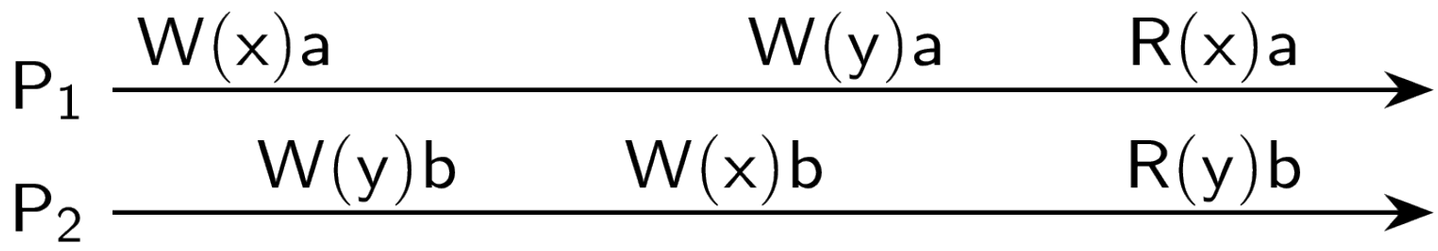


Sequential consistency for x:
R(x)a **happens before** W(x)b

Sequential consistency for y:
R(y)b **happens before** W(y)a

But:
W(y)a < R(x)a < W(x)b < R(y)b < W(y)a
Contradiction!

Consistency Models



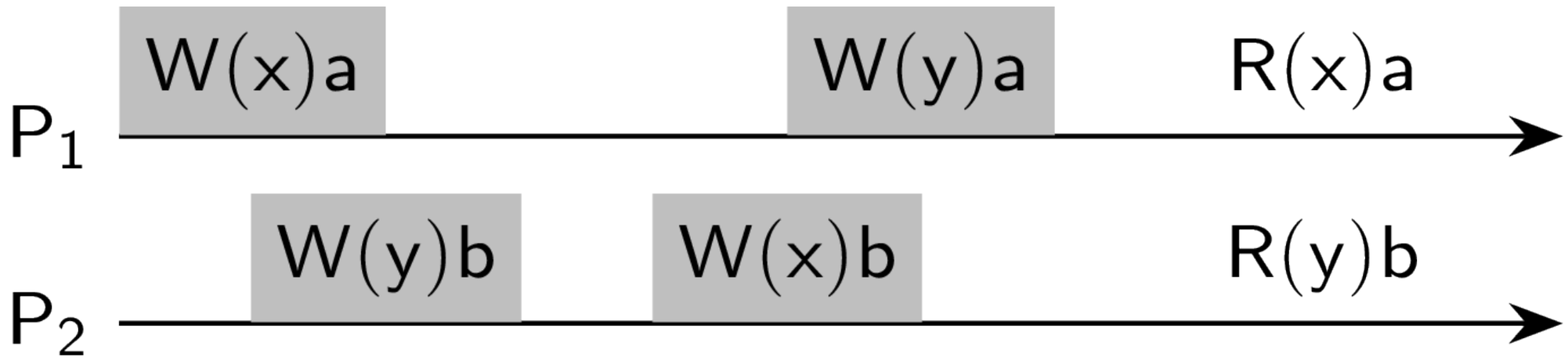
Ordering of operations	Result	
$W_1(x)a; W_1(y)a; W_2(y)b; W_2(x)b$	$R_1(x)b$	$R_2(y)b$
$W_1(x)a; W_2(y)b; W_1(y)a; W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_1(x)a; W_2(y)b; W_2(x)b; W_1(y)a$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b; W_1(x)a; W_1(y)a; W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b; W_1(x)a; W_2(x)b; W_1(y)a$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b; W_2(x)b; W_1(x)a; W_1(y)a$	$R_1(x)a$	$R_2(y)a$

Consistency Models

- Conclusion:
  - Sequential consistency is not composable

Consistency Models

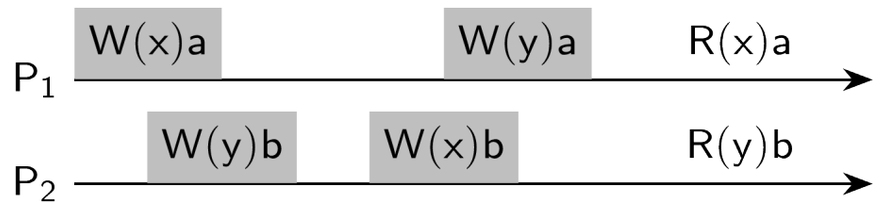
- Linearizable sequential consistency:
  - Effects of operations take place between beginning and end of an operation



Results of writes take place within the time limits indicated in gray

Consistency Models

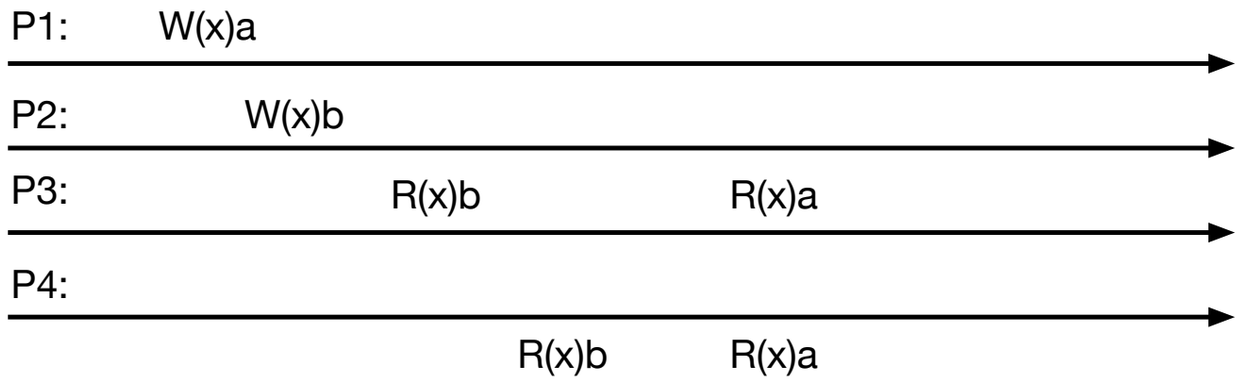
- Now we have less possible orderings

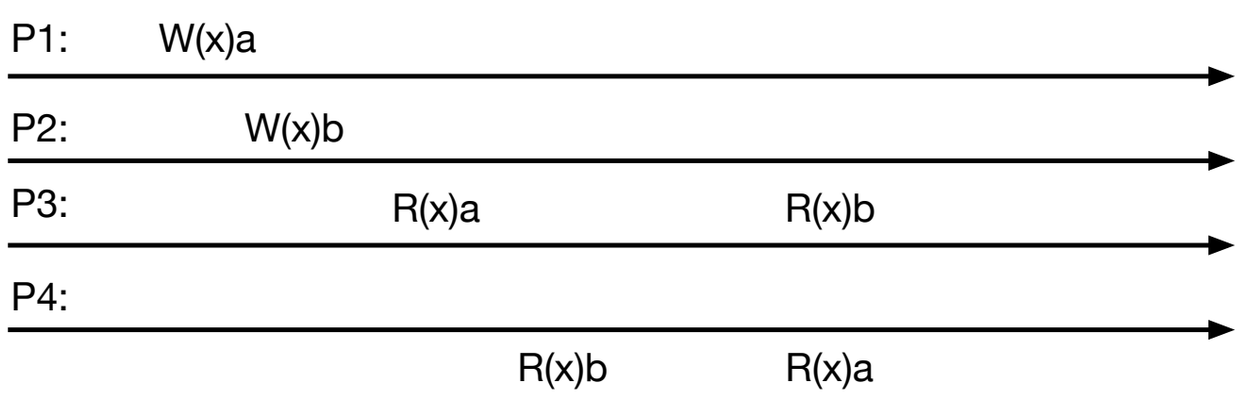


Ordering of operations	Result	
$W_1(x)a; W_2(y)b; W_1(y)a; W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_1(x)a; W_2(y)b; W_2(x)b; W_1(y)a$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b; W_1(x)a; W_1(y)a; W_2(x)b$	$R_1(x)b$	$R_2(y)a$
$W_2(y)b; W_1(x)a; W_2(x)b; W_1(y)a$	$R_1(x)b$	$R_2(y)a$

Consistency Models

- Sequential consistency can be investigated in terms of histories
- Quiz: Find a history for the first result that
  - Maintains program order
  - Respects data coherence





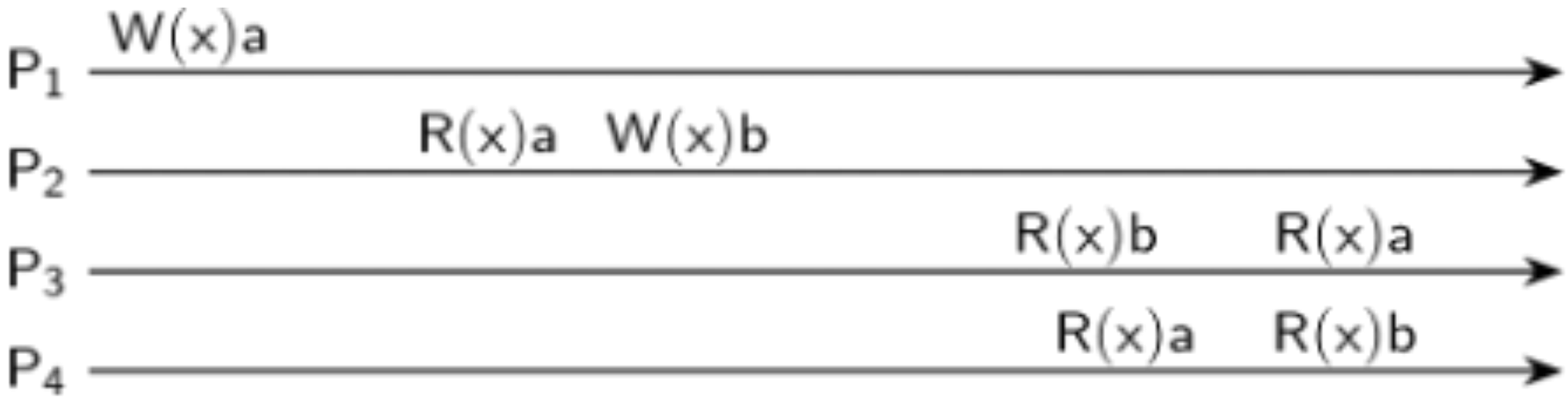
Consistency Models

- Sequential consistency is programmer friendly
- But difficult to implement efficiently
  - Lipton and Sandberg (1988)
    - Sequential consistent store with
      - *r* read time
      - *w* write time
      - *t* packet transfer time between nodes
    - has r+w>t
- Impossible to have both effective reads and writes

Consistency Models

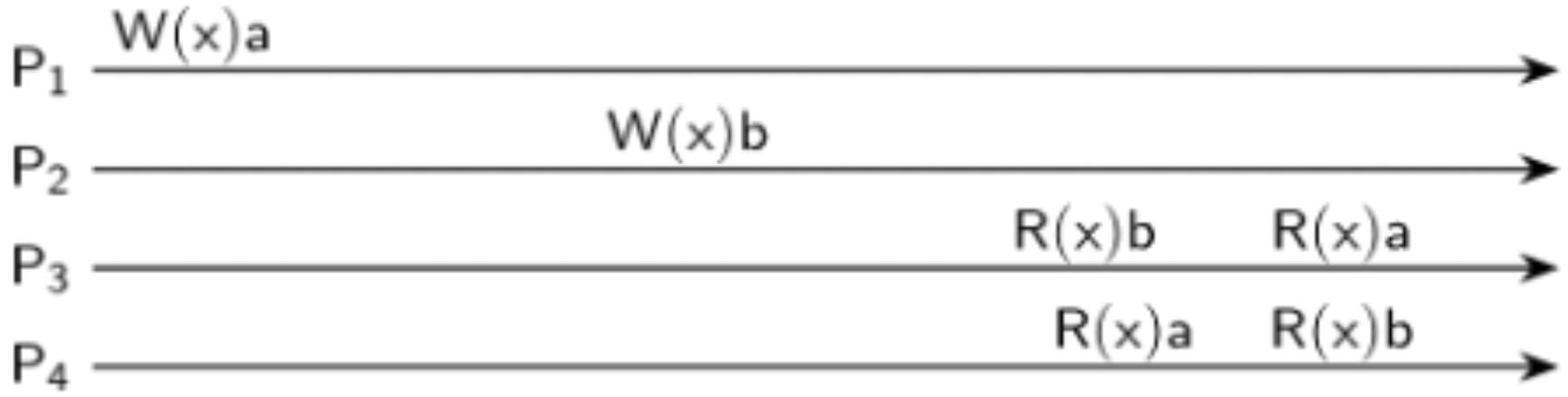
- Causal consistency:
  - Weakening of sequential consistency
  - *Definition:*
    - *Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order by different processes.*

Consistency Models



A violation of a causally-consistent store

Consistency Models

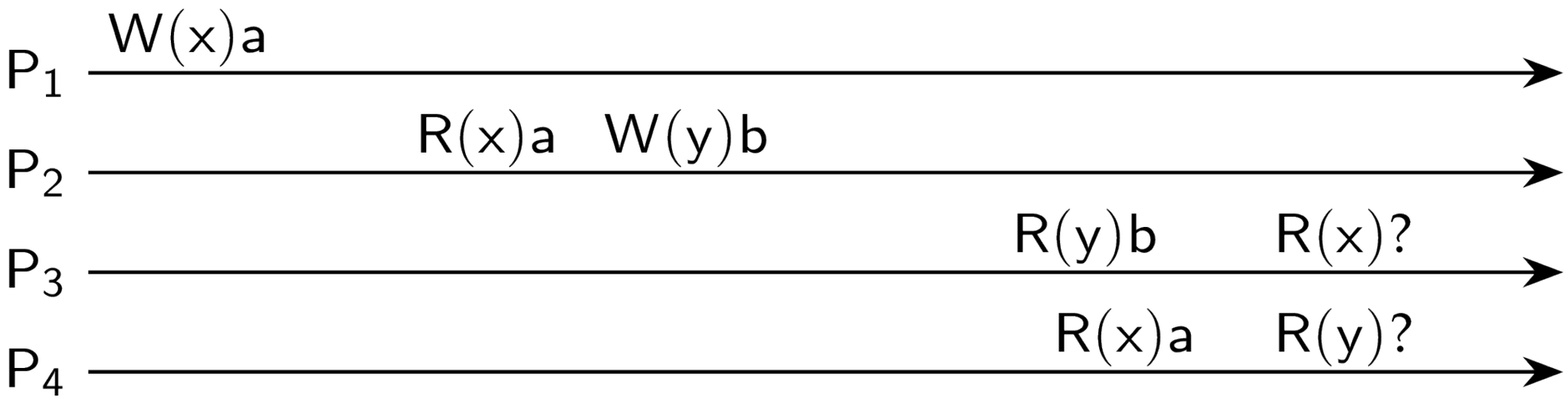


A correct sequence of events in a causally-consistent store

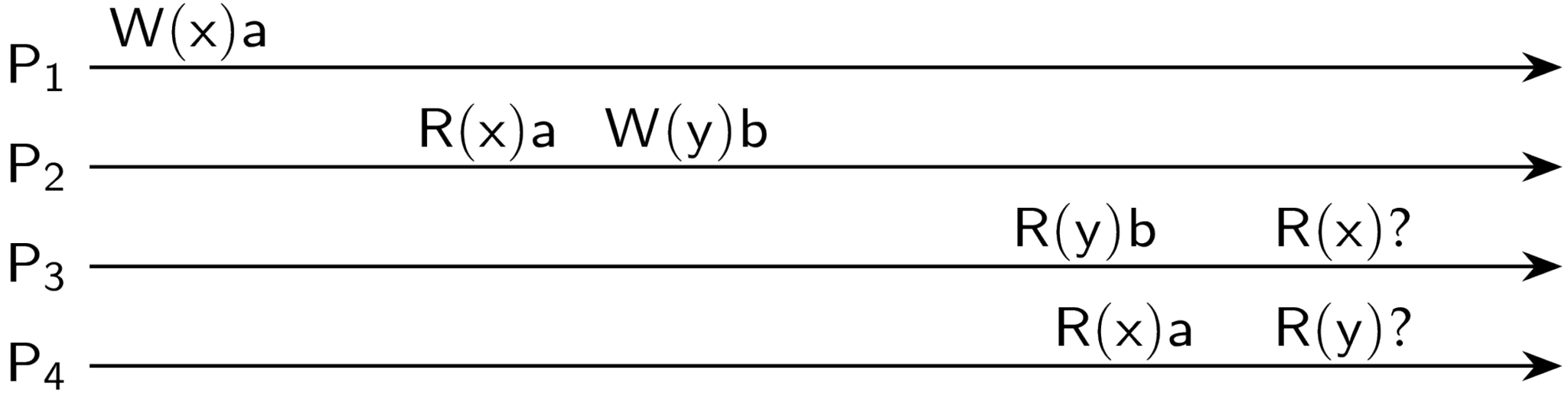
Consistency Models

- Implementing causal consistency
  - Keep track which processes have seen which write

- Subtle Issues: Example

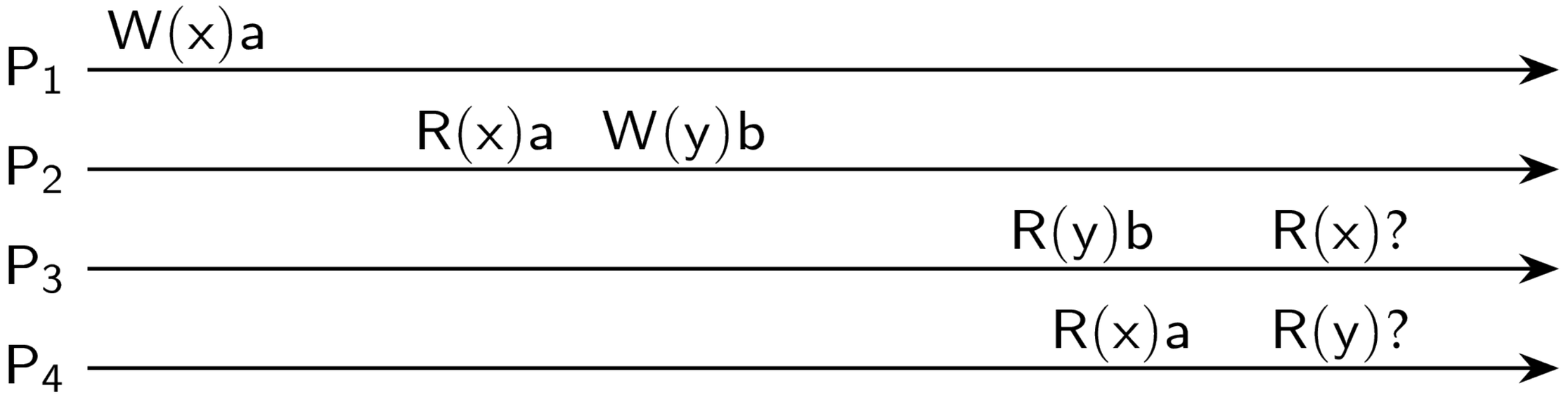


Consistency Models



- P_3 reads b from y
- Thus, $R_2(x)a\ W_2(y)b$ have happened before. There are no other writes to x, so we need to read a from x

Consistency Models



- P_4 reads a from x
- So, $W_1(x)a$ has to come first
- But there is no ordering with regards to $W_2(y)b$, so both $R_4(y)$None and $R_4(y)b$ are possible

Consistency Models

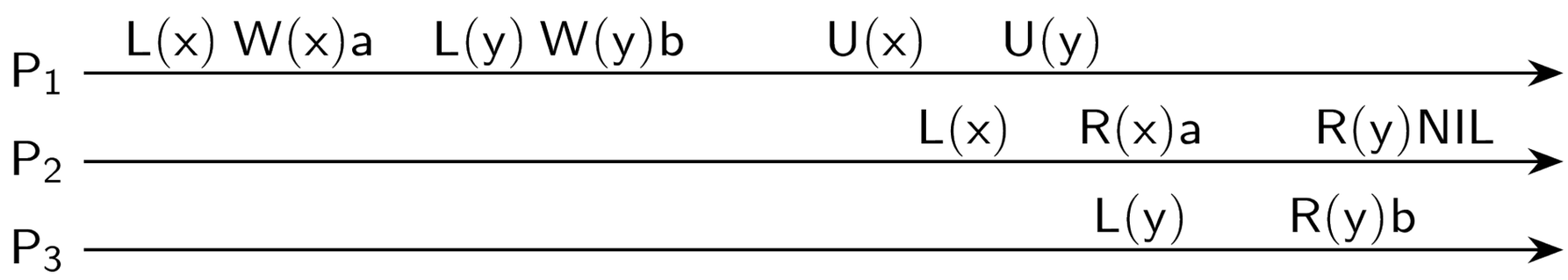
- Grouping
  - Developed for shared-memory multiprocessor systems
  - Use critical sections e.g. locks
    - Coarse-grained critical sections:
      - All shared items have a single lock
    - Fine-grained critical sections
      - Items have individual locks

Consistency Models

- Locks:
  - Acquiring a lock
  - Exclusive and non-exclusive access to a lock

Consistency Models

- Entry consistency:
  - Each data item has a lock
  - Example:
  -



P_2 does not lock y, so it can read either NIL or b

Consistency Models

- Entry consistency
  - Accesses to locks are sequentially consistent.
  - No access to a lock is allowed to be performed until all previous writes have completed everywhere.
  - No data access is allowed to be performed until all previous accesses to locks have been performed.

Consistency vs Coherence

- Consistency models describe what can be expected if multiple processes operate concurrently on data
- Coherence models describe what can be expected to hold for a *single data item*.

Eventual Consistency

- A typical case:
  - In a database, most transactions perform only reads
  - Updates are rare
  - **Question**: How fast should the updates propagate?

Eventual Consistency

- Example: Web sites
  - Web sites are under the authority of a singly user (webmaster)
  - Browsers and Web proxies keep copies
    - Do not check for updates immediately

Eventual Consistency

- In these scenarios:
  - Never have ***write-write*** conflicts
  - Only have ***read-write*** conflicts
  - Can then propagate updates in a lazy fashion

- ***Eventual Consistency***

Eventual Consistency

- **Strong eventual consistency**
- Basic idea:
  - If there are conflicting updates, have a globally determined resolution mechanism (for example, using NTP, simply let the “most recent” update win).

Program Consistency

- P is a monotonic problem if
  - for any input sets S and T , P(S) ⊆ P(T).
- Observation: A program solving a monotonic problem can start with incomplete information, but is guaranteed not to have to roll back when missing information becomes available.

Program Consistency

- Example: Filling a shopping cart
  - As long as we only add items, eventual consistency is easy
  - If we allow removing items, it becomes difficult
  - Solution:
    - Keep the add and the remove operations separate
    - Reconcile at the end

Continuous Consistency

- Defining Inconsistencies
  - Replicas can have different numerical values
  - Replicas can differ in staleness
  - Replicas can have different update histories

Continuous Consistency

- Replicas can have different numerical values
  - E.g. Currency rates cannot differ by more than 0.1%
- Replicas can differ in staleness
  - Last time data was updated
    - E.g. Weather reports are updated every few hours
    - Primary site can push updates lazily

Continuous Consistency

- Replicas can have different update histories
  - Updates could be applied to local copies tentatively
  - After reaching agreement, replicas can roll-back some local updates, …

Consistency Units

- Consistency Unit — Conit
  - Unit over which consistency is to be measured
  - Examples:
    - Record representing an exchange rate
    - Record representing a stock price
    - Current weather report

Consistency Units

- Example: Data on a fleet of cars with attention to gas prices

Replica A

Conit

```
d = 558    // distance
g =  95    // gas
p =  78    // price
```

Operation		Result
< 5, B>	g ← g + 45	[g = 45]
< 8, A>	g ← g + 50	[g = 95]
< 9, A>	p ← p + 78	[p = 78]
<10, A>	d ← d + 558	[d = 558]

Vector clock A = (11, 5)
Order deviation = 3
Numerical deviation = (2, 482)

Replica B

Conit

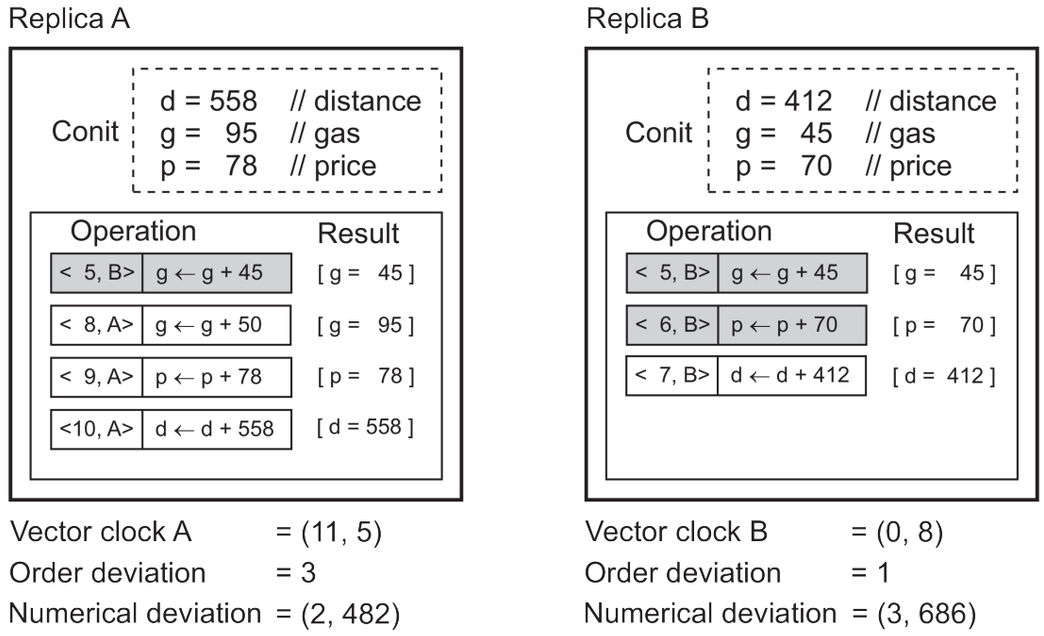
```
d = 412    // distance
g =  45    // gas
p =  70    // price
```

Operation		Result
< 5, B>	g ← g + 45	[g = 45]
< 6, B>	p ← p + 70	[p = 70]
< 7, B>	d ← d + 412	[d = 412]

Vector clock B = (0, 8)
Order deviation = 1
Numerical deviation = (3, 686)

Consistency Units

- Example:
  - Driver reports:
    - amount of gasoline g
    - price paid p
    - total distance d
- Conit is (g, p, d)
- Conit is distributed over two servers

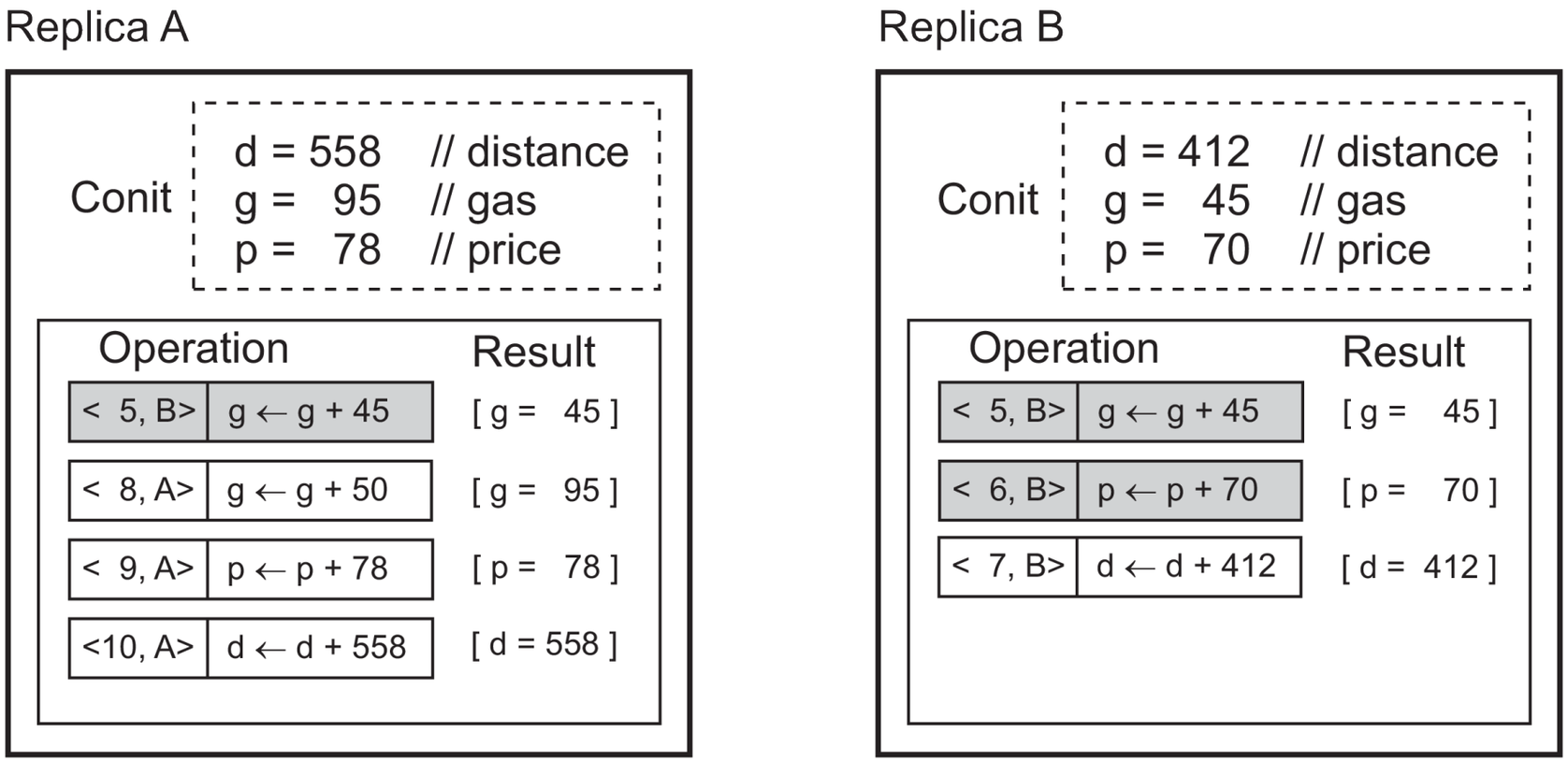


Consistency Units

- Example:
  - Replicas maintain a two-dimensional vector clock
  - $< T, R >$ operation carried out by replica R at time T

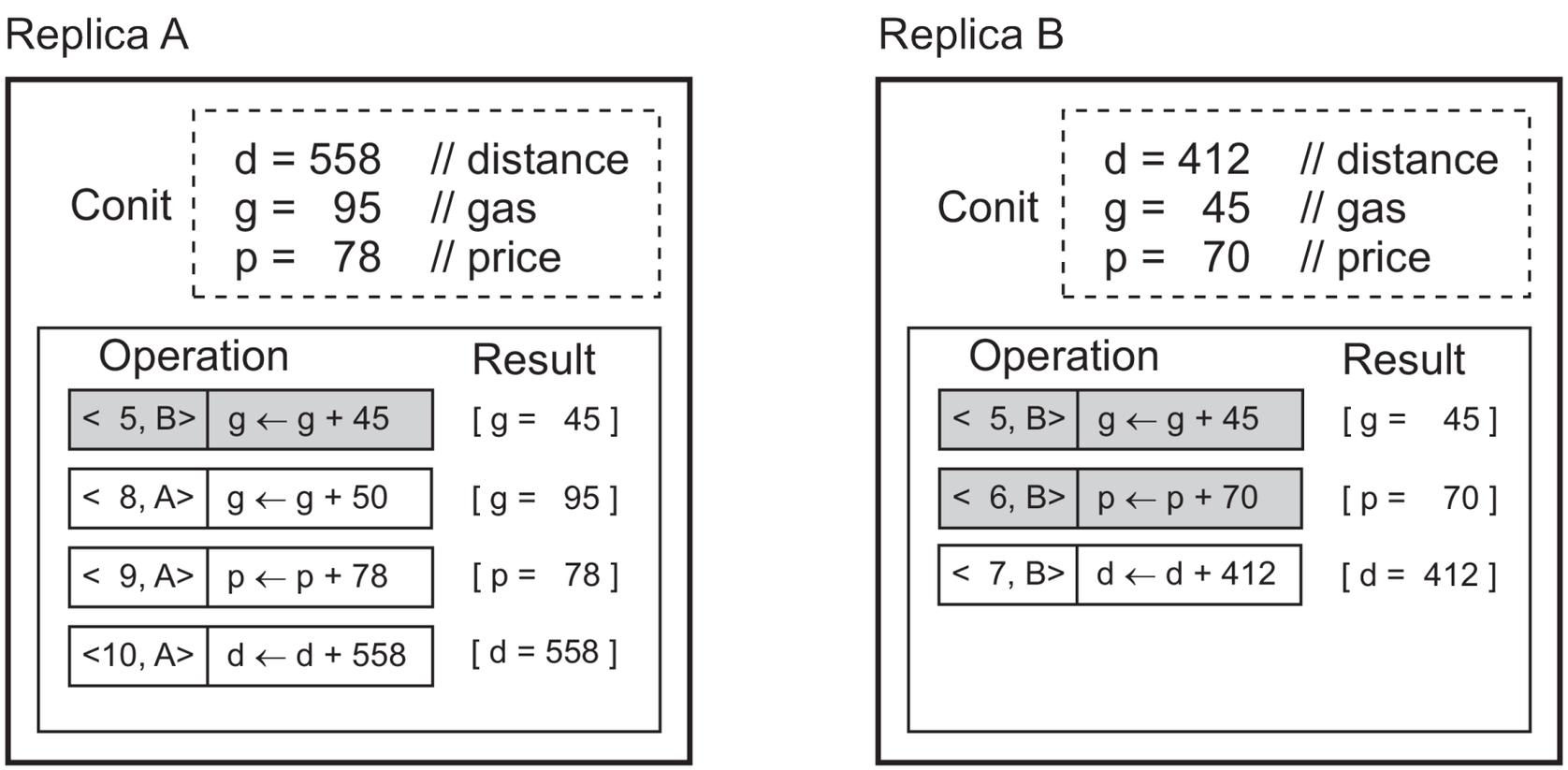
Consistency Units

- Replica A just received an update from B and committed it
- It has three tentative updates



Consistency Units

- B has committed two updates



Consistency Units

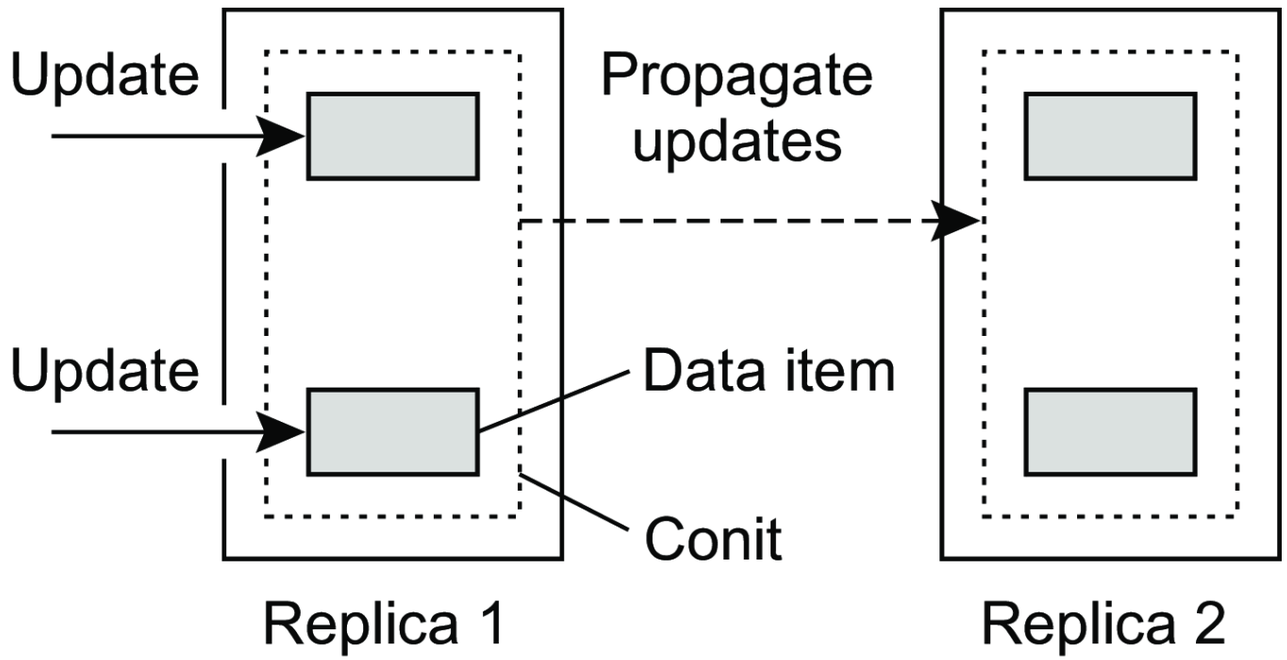
- Order deviation:
  - Number of tentative updates
    - A has 3, B has 1
- Numerical deviation:
  - Number of operations at other replicas not yet seen
  - Sum of corresponding values
  - A has (2, 482), B has(3,686)

Consistency Units

- Possible to specify consistency
  - Bounds on *numerical deviation* or *order deviation*
  - This implies communication between replicas
  - Assumption is that this communication is less costly than communication to keep replicas synchronized

Consistency Units

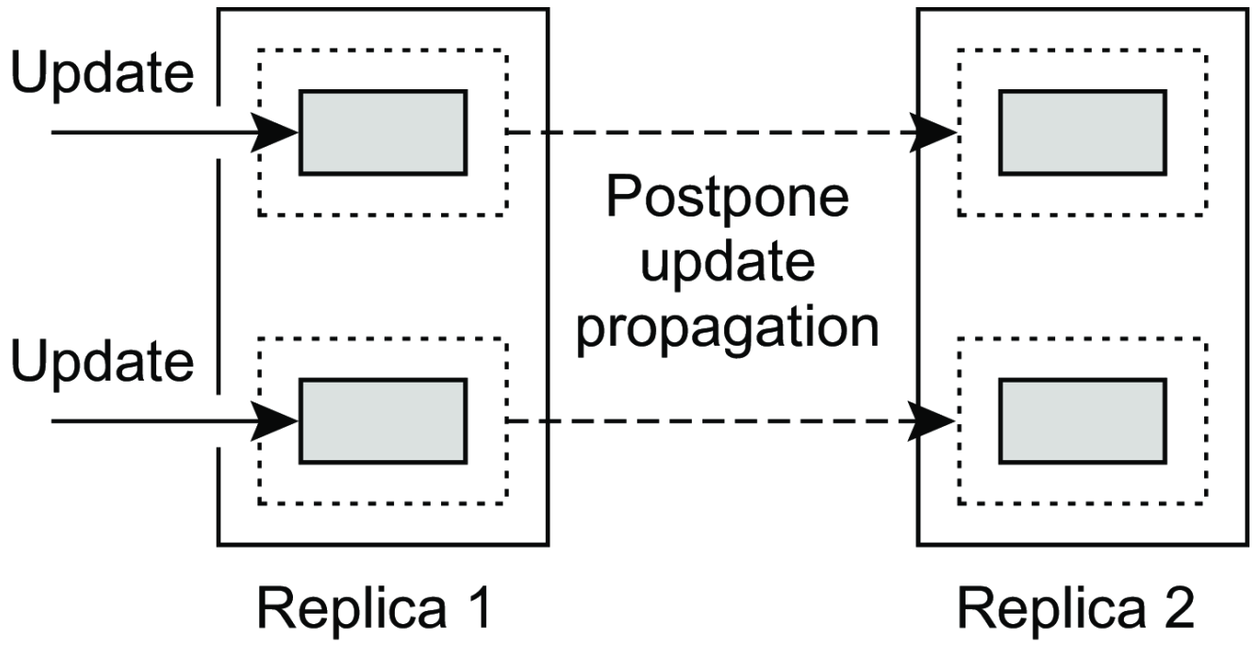
- Granularity of Conits



Coarse Granularity
with possibility of *false sharing*

Consistency Units

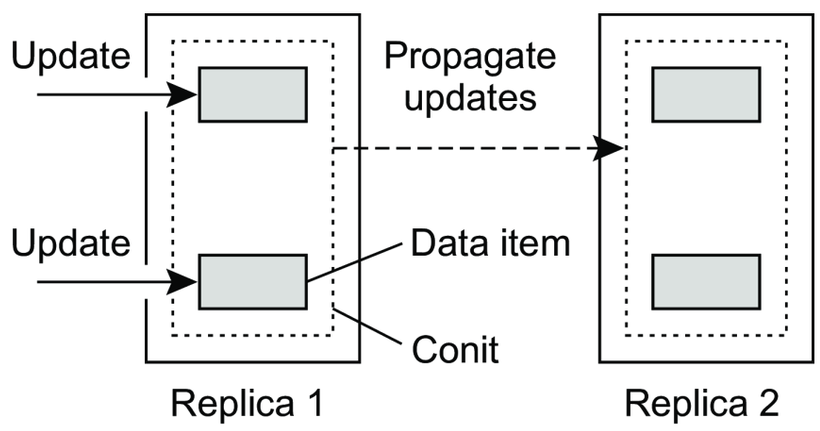
- Granularity of Conits



Fine-grained Conits

Consistency Units

- Coarse granularity:
  - Second data item needs to be updates as well



Consistency Units

- Fine granularity:
  - Need to manage more conits

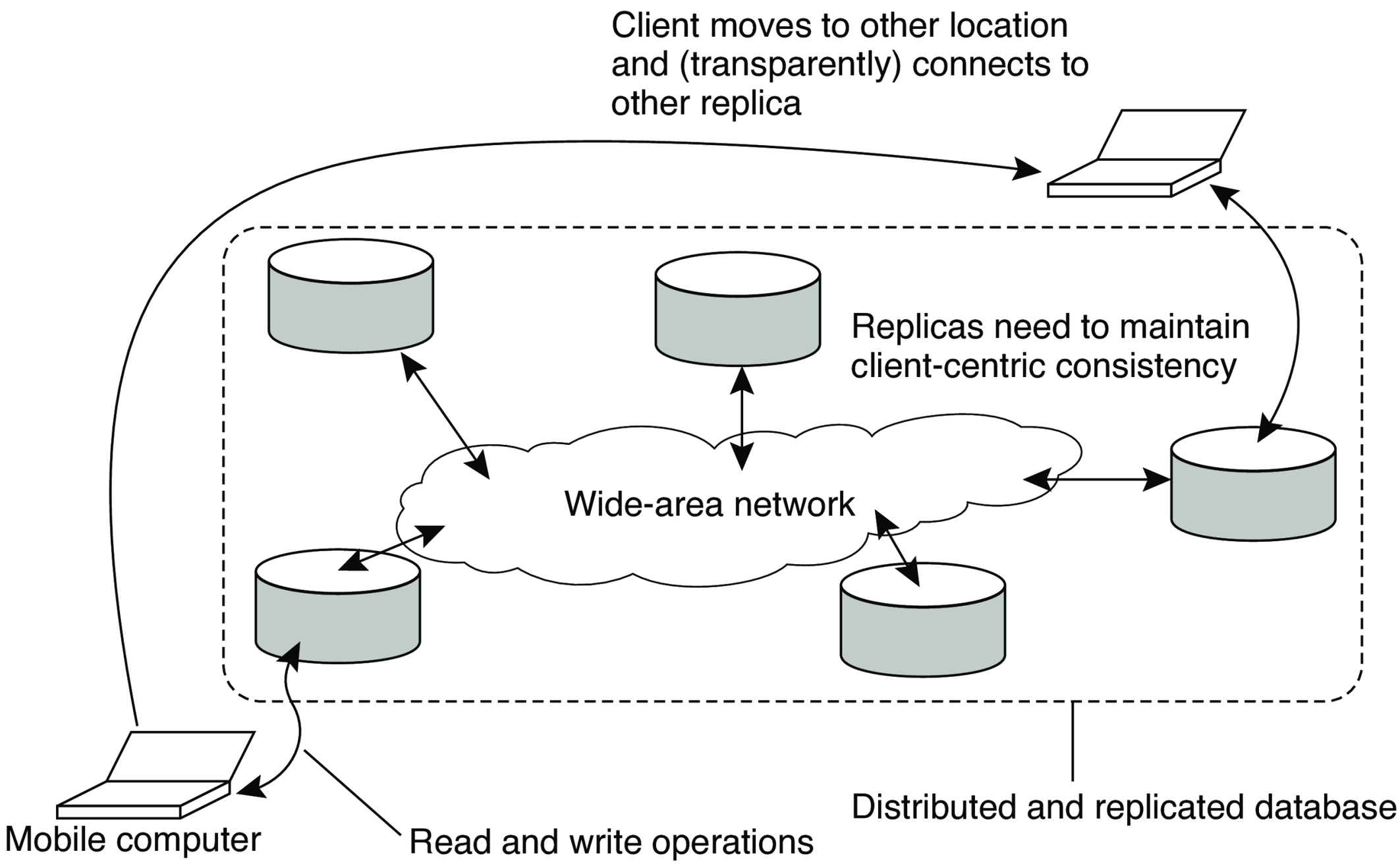
Consistency Units

- For the near future:
  - Protocols for continuous consistency
- Continuous consistency needs to be supported for programming

Client-Centric Consistency Models

- **Data-centric** consistency models:
  - Systemwide consistent view on a data store
- For data stores with mainly read load, can assume eventual consistency
- **Client-centric** consistency models:
  - Allow to hide inconsistencies in a cheap way

Client-Centric Consistency Models



Client-Centric Consistency Models

- Users do not see inconsistencies if they always access the same replica
- But mobile users can access different replicas
- Client-centric consistency models:
  - Specify what can happen to the mobile user

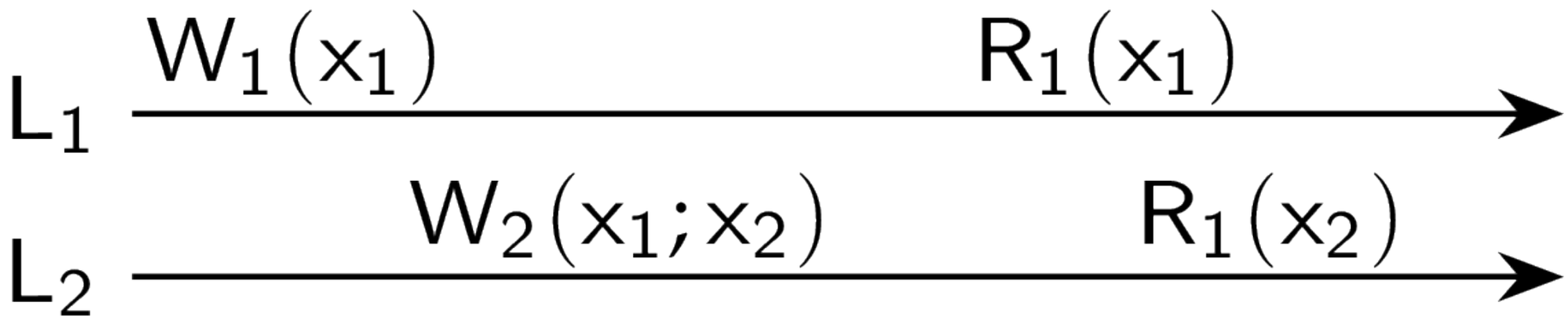
Client-Centric Consistency Models

- Description
  - x_i is a ***version*** of data item x
  - x_i is the result of write operations since initialization: ***Write set*** $WS(s_i)$
- x_j follows from x_i if we add operations to $WS(x_i)$
  - $W(x_i; x_j)$ means x_j follows from x_i
  - $W(x_i \mid x_j)$ means x_i and x_j are unrelated

Client-Centric Consistency Models

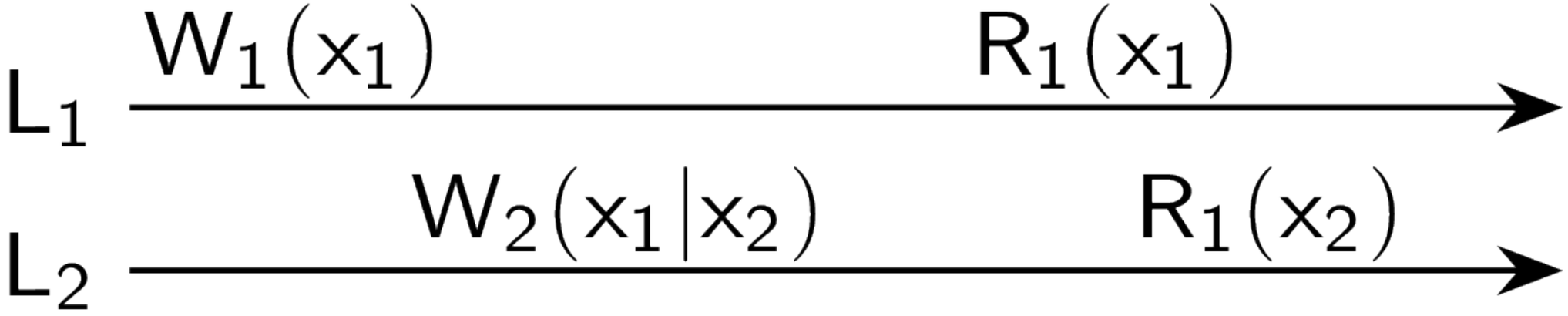
- Monotonic-read consistency
  - If a process reads the value of a data item *x*, any successive read operation on *x* by that process will always return that same value or a more recent value.

Client-Centric Consistency Models



Monotonic Read Consistent

Client-Centric Consistency Models



Monotonic read is violated:

P_1 has read x_1 at local data store L_1 and then read $R_1(x_2)$ at L_2

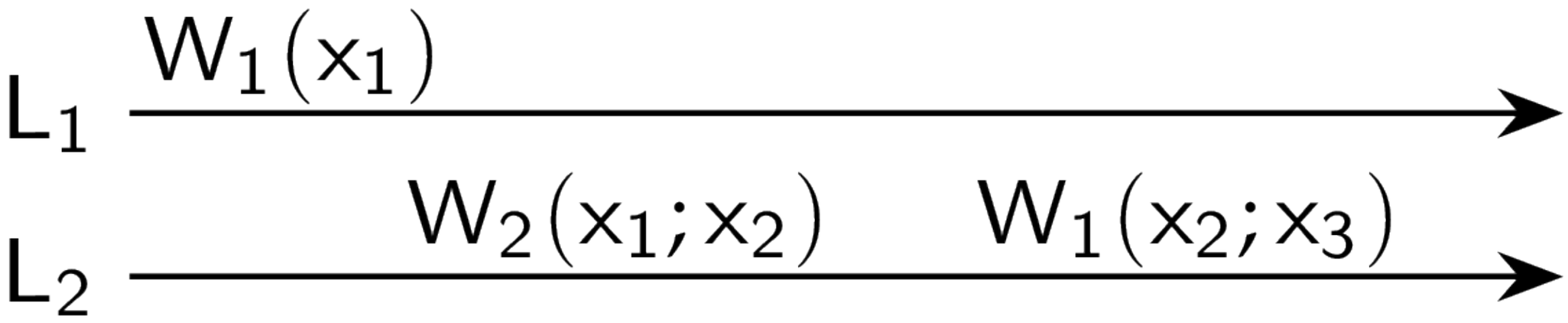
P_2 has a write operation that **does not** follow from x_1

P_1's second read does not include the effect of its write operation!

Client-Centric Consistency Models

- Monotonic write
  - A write operation by a process (user) on a data item *x* is completed before any successive write operation on *x* by the same process (user)

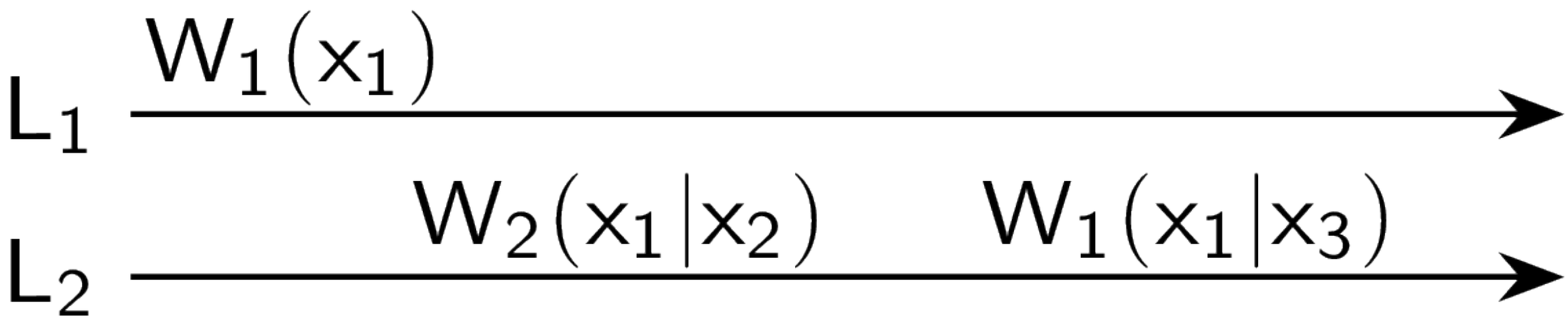
Client-Centric Consistency Models



Monotonic-write consistent store

P_1 performs a write on L_1. This write is not propagated to L_2 yet. Later, it writes to L_2, reading a value that depends on its previous write and then writing a new value.

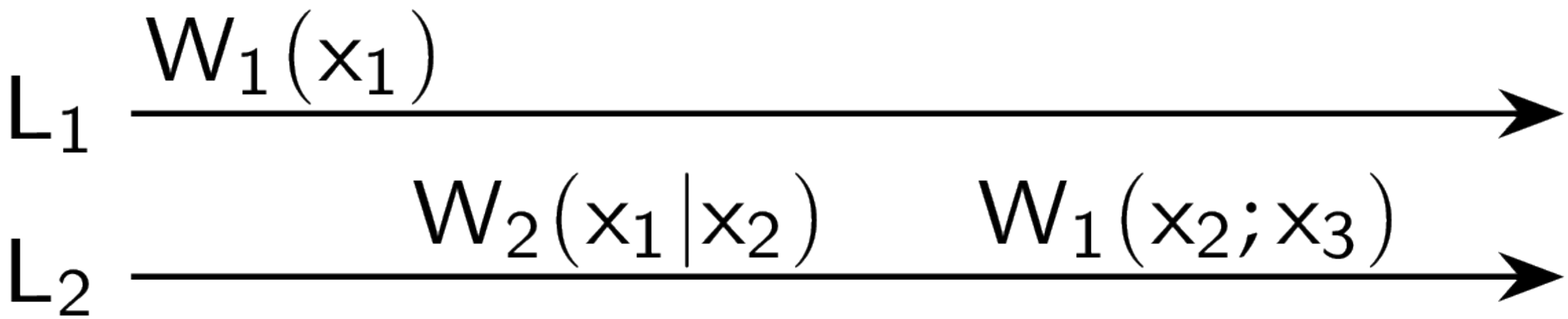
Client-Centric Consistency Models



Monotonic write consistency is violated

Violation as the first write has no effect on the second write of P_1.

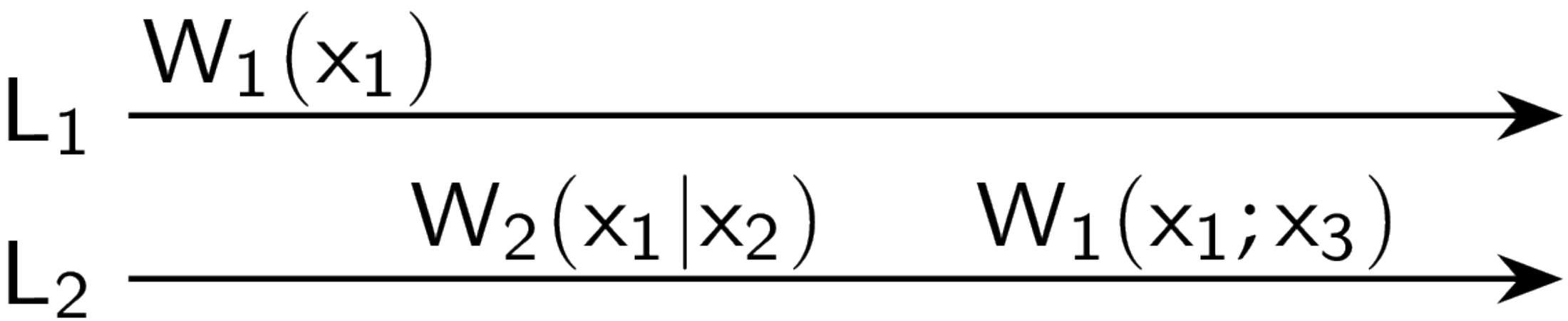
Client-Centric Consistency Models



No monotonic write consistency

Violation since the second write of P_1 does not depend on x_1.

Client-Centric Consistency Models

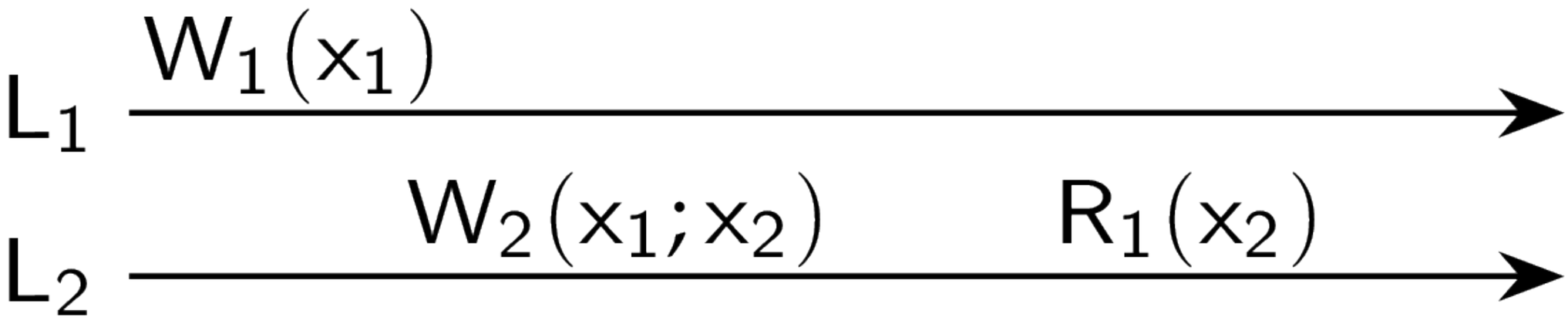


Monotonic write consistent, even though P_1 has overwritten x_2

Client-Centric Consistency Models

- 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.*

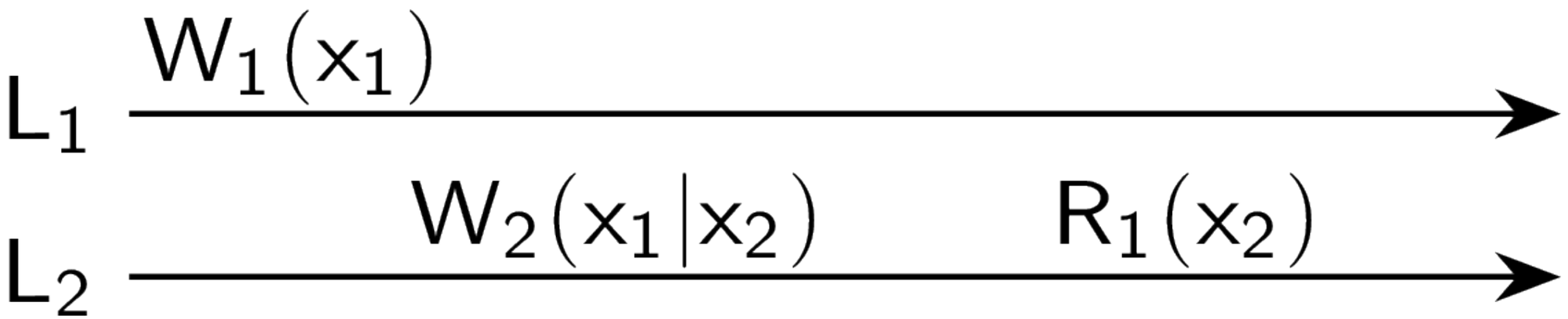
Client-Centric Consistency Models



Read-your-writes consistency is provided

Read your writes does not mean that other processes cannot change your write. The other processes need to depend on your write, though.

Client-Centric Consistency Models



No read-your-writes consistency

Here the effect of the write has been destroyed by another process.

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
  - A write operation is always propagated before the next read operation
  - Counterexample: I update my web-page, but when I read it, I still see the old value

Client-Centric Consistency Models

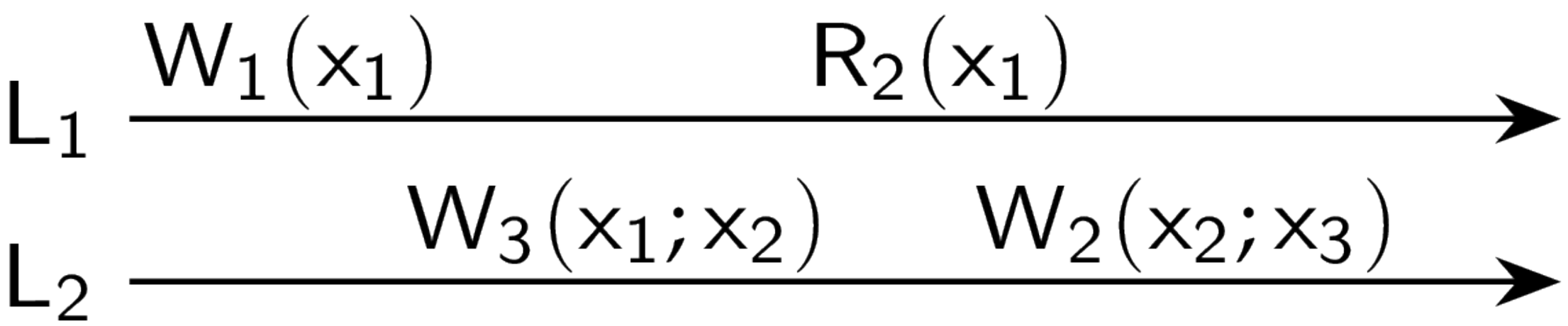
- Write follows reads
  - A write operation by a process on a data item *x* following a previous read operation on *x* by the same process is guaranteed to take place on the same or a more recent value of *x* that was read.

- Any successive write operation by a process on a data item *x* will be performed on a copy of *x* that was shown in the latest read or on a more recent copy

Client-Centric Consistency Models

- Write follows 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.*

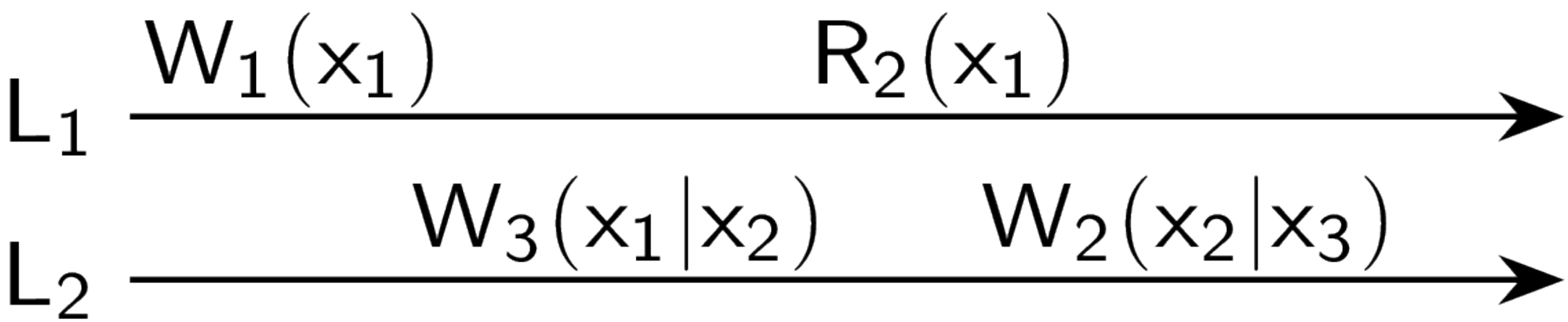
Client-Centric Consistency Models



Writes-follows-reads consistent data store

Process P_1 writes an item. the write is propagated to L_2 and Process P_3 updates the value that P_1 wrote. Process P_2 reads the item, and gets the previous value. Later P_2 writes to the item, with a value known to have depended on x_1.

Client-Centric Consistency Models



Writes-follow-reads is violated

In this scenario, P_1 and P_3 concurrently update x. P_2 reads P_1's version of x, but then updates based on P_3's version.

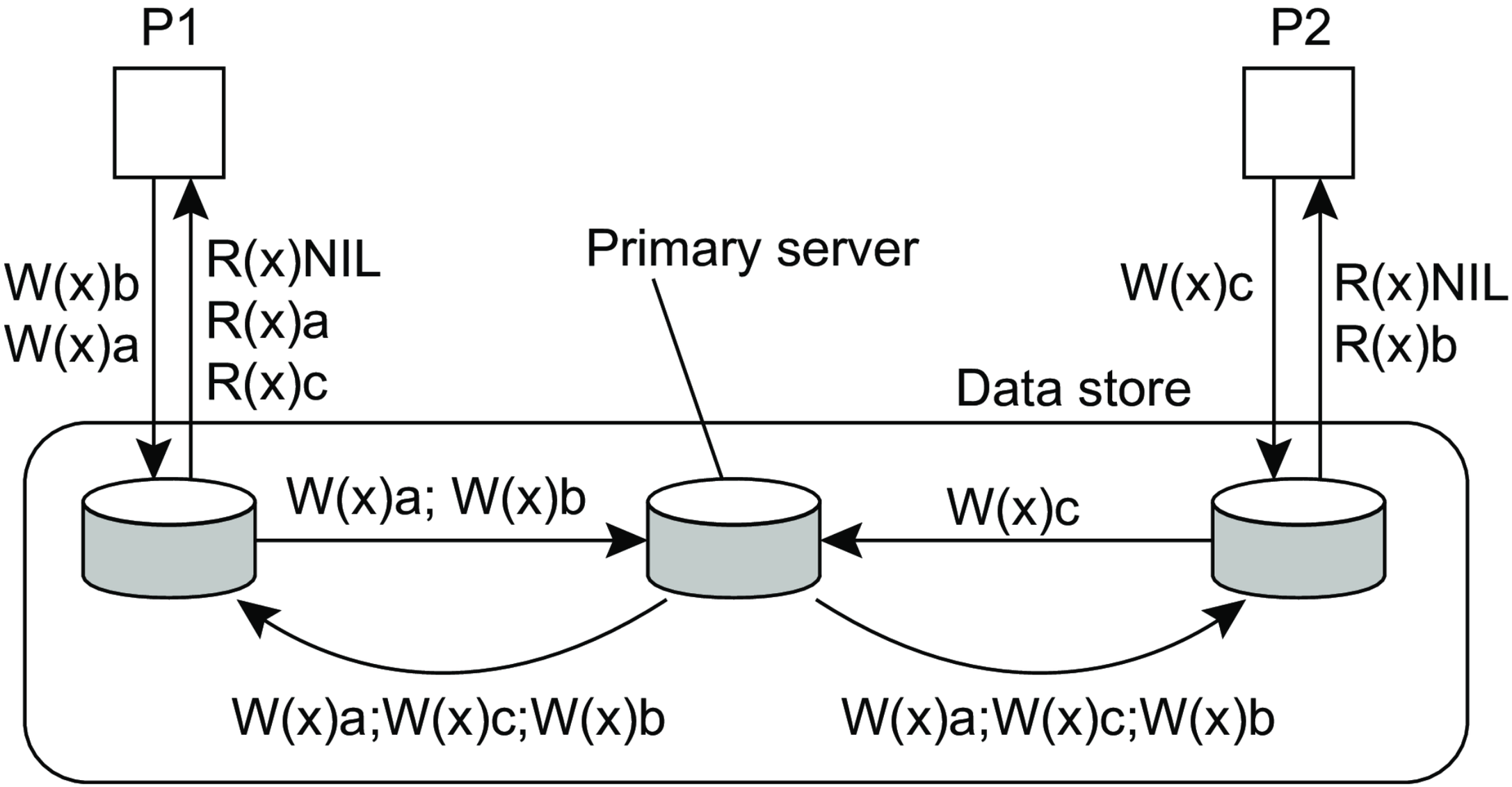
Example: Zookeeper

- Zookeeper **updates** are serializable
  - Monotonic writes is obtained by guaranteeing that the seeming linear ordering of events is consistent with the ordering of events in the program

Example: Zookeeper

- Guarantees monotonic reads
- Does **not** guarantee read-your-writes or writes-follow-read consistency

Example: Zookeeper



Example: Zookeeper

- Zookeeper clients read from one of the replicas
- They send updates to their replica.
- This replica then sends updates to a primary copy
- The primary copy updates all replica lazily

Example: Zookeeper

- Zookeeper combines elements of data- and client-centric consistency

Replica Placement

- Figure out the best k places out of N possible locations
  - Objective:
  - Network related criteria
    - Latency, Bandwidth, Hop count, Logical distance
  - Cost related criteria

Replica Placement

- Figure out the best k places out of N possible locations
  - Objective: Minimize average distance of all clients to the nearest replica
  - Possibility 1: Global optimization
  - Possibility 2: Greedy: Select the best place for one replica. Then choose the next best server.
  - Possibility 3: Determine k clusters.
  - Possibility 4: Map into a d-dimensional space modeling objective criteria

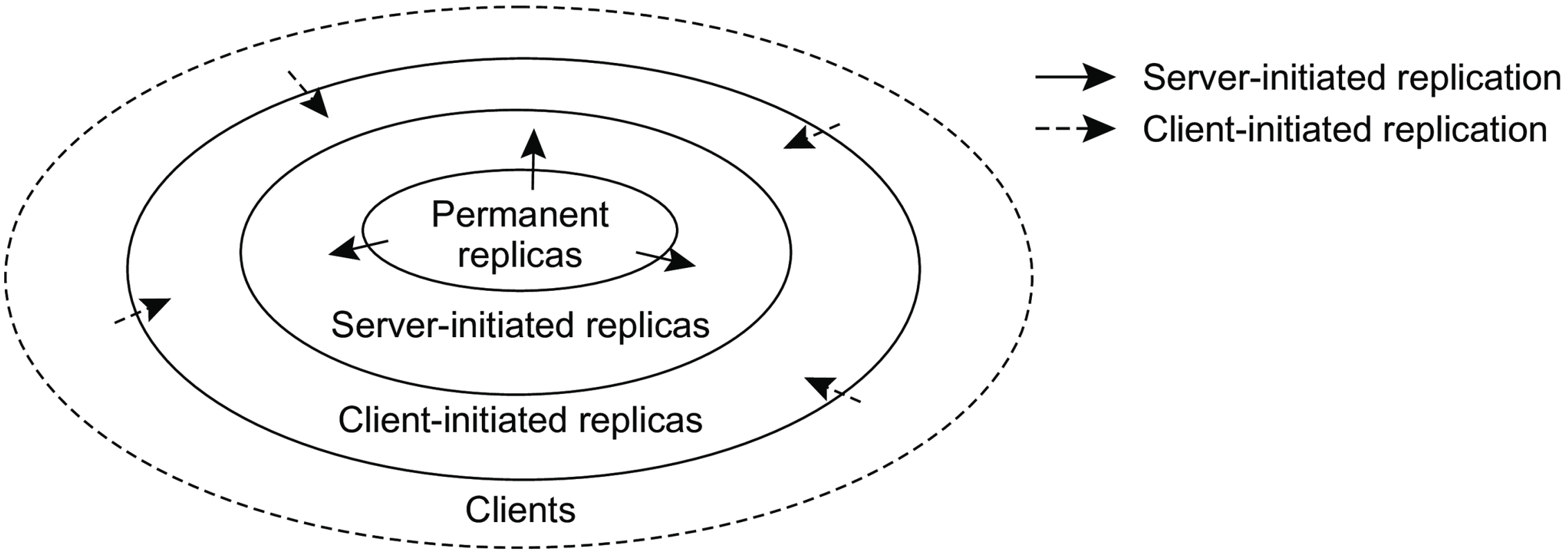
Replica Placement

Main class	**Subclass**
QoS Aware	Optimized QoS
	Bounded QoS
Consistency Aware	Periodic update
	Aperiodic update
	Expiration-based update
	Cache-based update
Energy	
Others	

Content Replication and Placement

- Distinguish different processes
  - A process is capable of hosting a replica of an object or data:
    - Permanent replicas: Process/machine always having a replica
    - Server-initiated replica: Process that can dynamically host a replica on request of another server in the data store
    - Client-initiated replica: Process that can dynamically host a replica on request of a client (client cache)

Content Replication and Placement



Content Replication and Placement

- Permanent replicas
  - Example 1:
    - Several replicas across servers at the same location.
    - Incoming request is routed to one of the servers

Content Replication and Placement

- Permanent replicas
  - Example 2:
    - Mirroring websites
    - Client decide which mirror to access

Content Replication and Placement

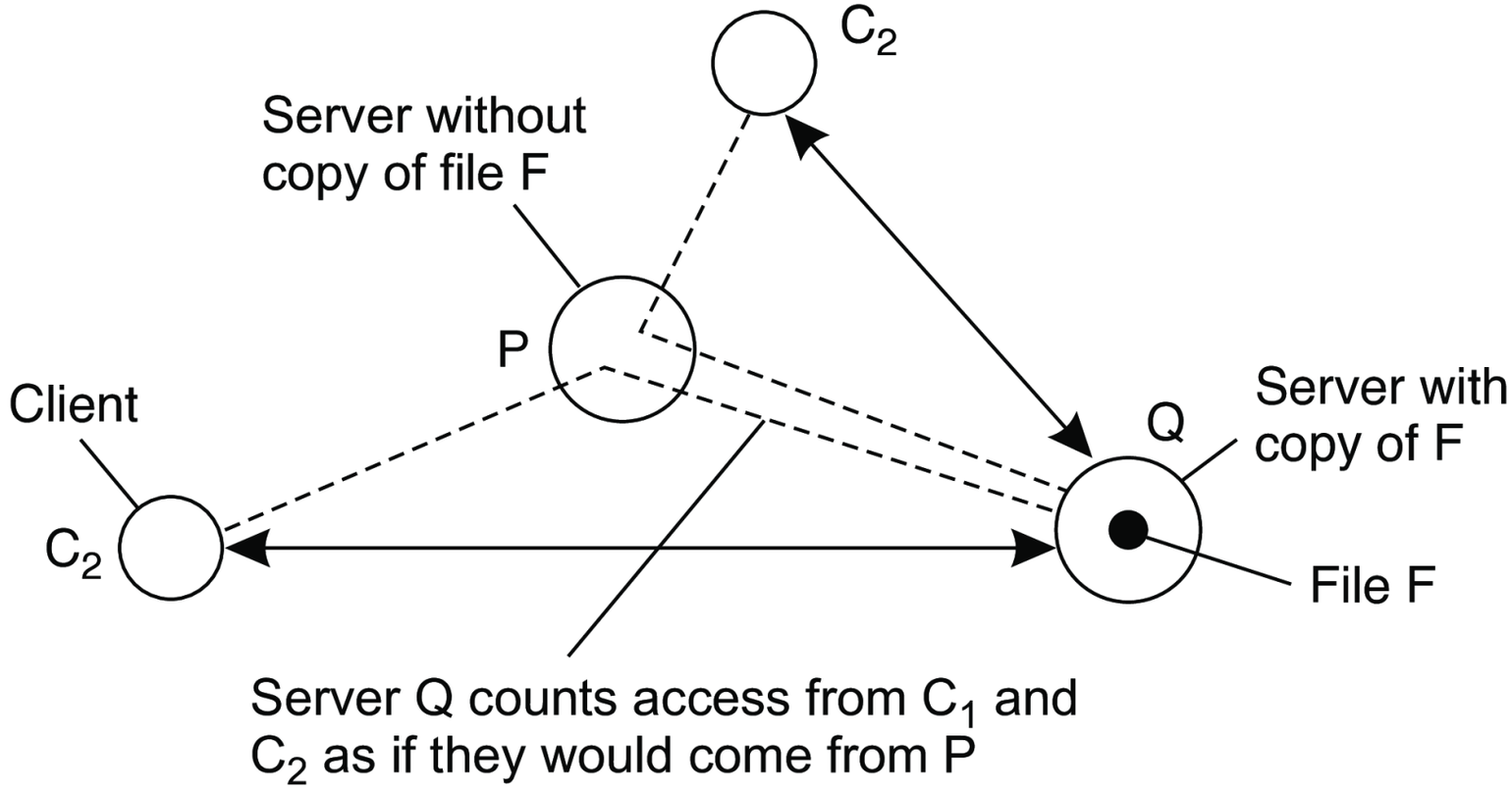
- Permanent replicas
  - Example 3:
    - Distributed Databases

Content Replication and Placement

- Server-initiated replicas
  - Copies of a data store generated for performance
  - Created at the initiative of the data store
  - Example: Content Delivery Networks (CDN)

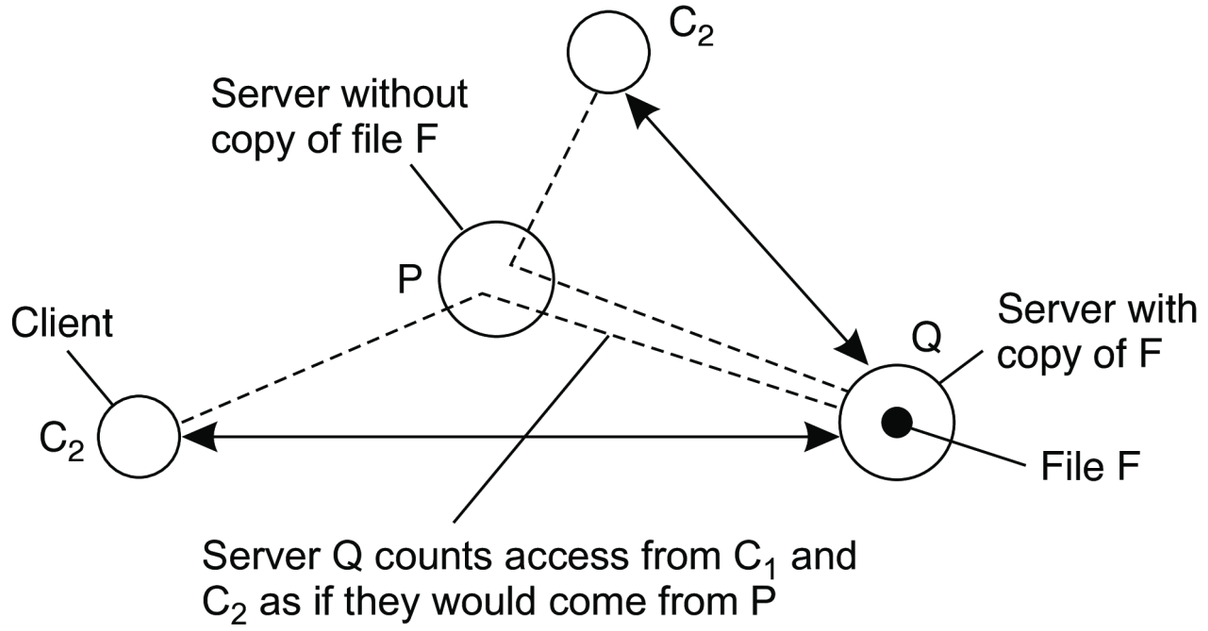
Content Replication and Placement

- Example: Dynamic replication of files for Web Hosting Service
  - Counting access requests from different clients:



Content Replication and Placement

- Keep track of access counts per file, aggregated by considering server closest to requesting clients
- Number of accesses drops below threshold D
  - drop file
- Number of accesses exceeds threshold R
  - replicate file
- Number of access between D and R
  - migrate file



Content Replication and Placement

- Client-initiated replica placement — Client caches
  - If a client ***reads*** frequently data
    - Move data closer to client by caching it

Content Distribution

- Propagate updates
  - Propagate only a notification of an update.
  - Transfer data from one copy to another.
  - Propagate the update operation to other copies.

Content Distribution

- Propagation a notification:
  - Invalidation protocols
    - Use little network bandwidth
    - Best if read-to-write ratio is small

Content Distribution

- Transfer data among replicas
  - Best if read-to-write ratio is high
  - Alternative:
    - log the changes and transfer only those logs to save bandwidth

Content Distribution

- Let the replica perform the updates themselves
  - ***Active replication***
  - Main advantage: Updates often use less bandwidth

Content Distribution

- Pull versus Push Protocols:

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

- Third choice: Leases
  - Promise by the server that it will push updates to the client for a specified time
  - Age-based leases:
    - An object that hasn't changed for a long time, will not change in the near future, so provide a long-lasting lease
  - Renewal-frequency based leases:
    - The more often a client requests a specific object, the longer the expiration time for that client (for that object) will be
  - State-based leases:
    - The more loaded a server is, the shorter the expiration times become

Content Distribution

- Unicasting versus Multicasting

Managing Replicating Objects

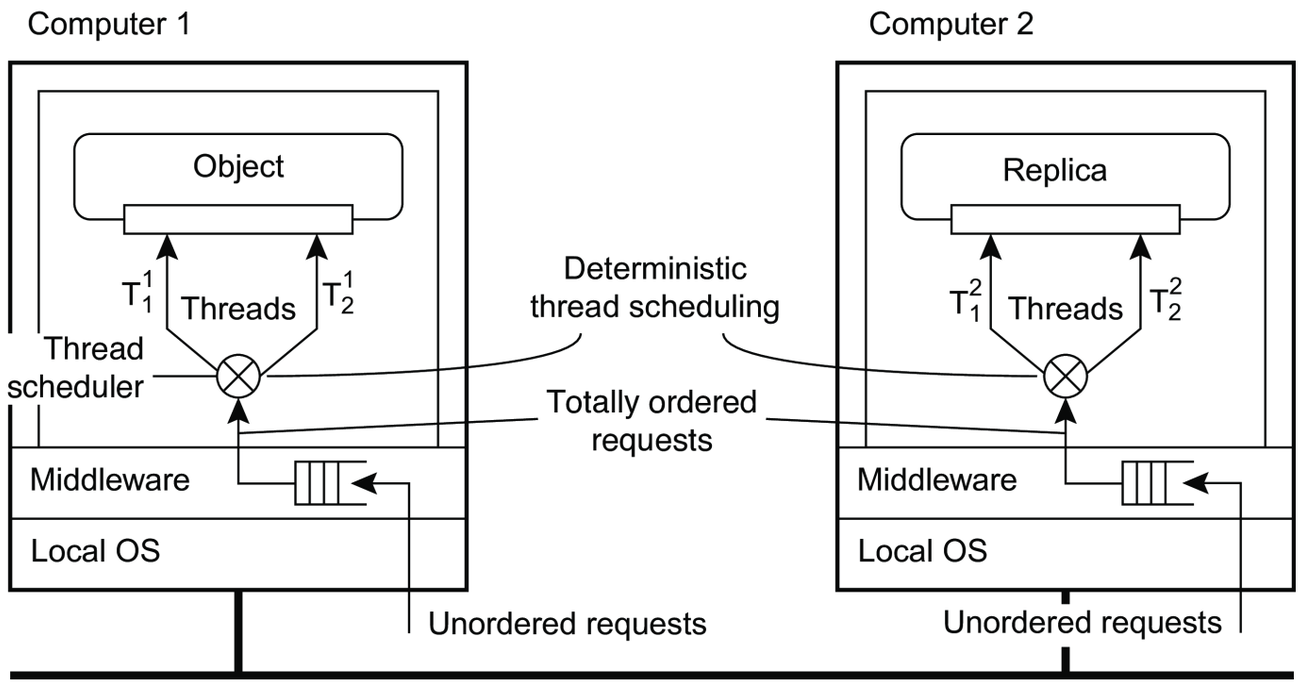
- Data-centric consistency usually uses entry consistency
  - Use locks or similar synchronization variables
- Entry consistency is difficult when objects are replicated
  - Serializable: Only one method access an object at a time
  - Need to insure that all changes to the replicated object are the same

Managing Replicating Objects

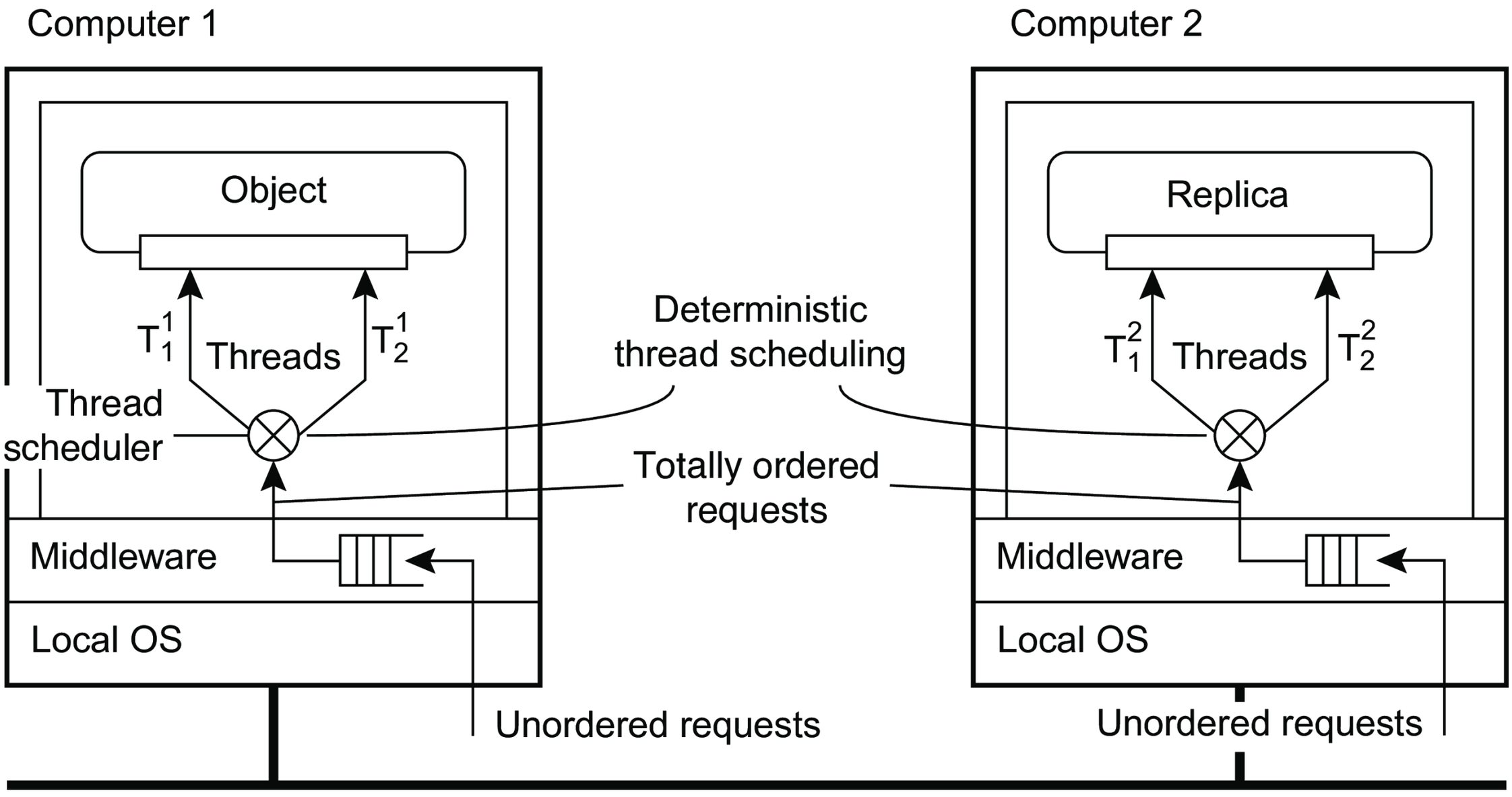
- Who is responsible for managing replication
  - Rarely, the replicas themselves
  - Usually, middleware

Managing Replicating Objects

- Method invocation at replicas needs to be serialized
- But the threads used for accessing replicas also need to be deterministic.



Managing Replicating Objects

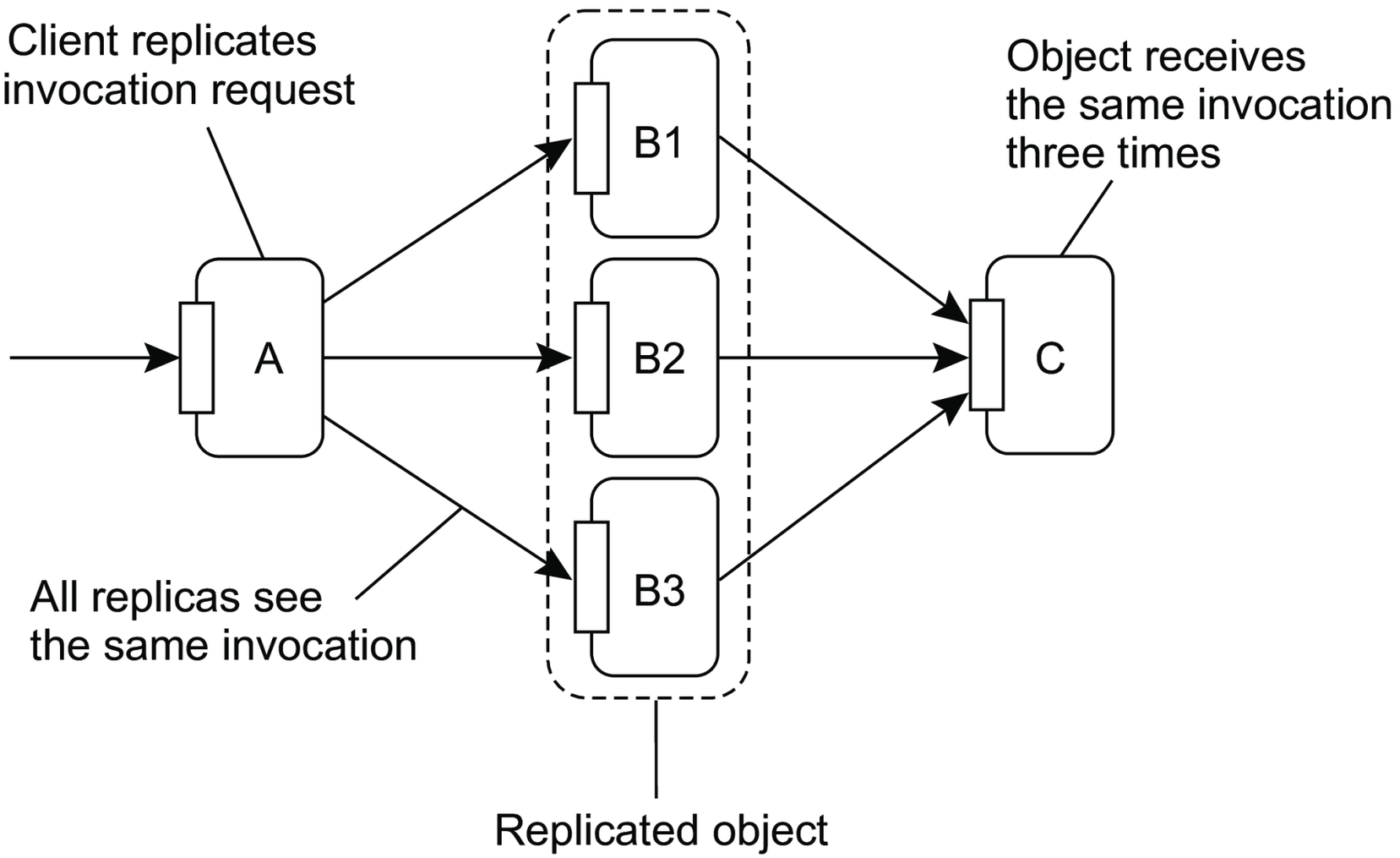


Managing Replicating Objects

- Possibilities:
  1. Threads sharing the same lock are scheduled in the same order on every replica
     - ***Deterministic Thread Scheduling***
     - Scheduling of threads done at a primary copy
  2. Identify application level information in order to identify locks used for serialization

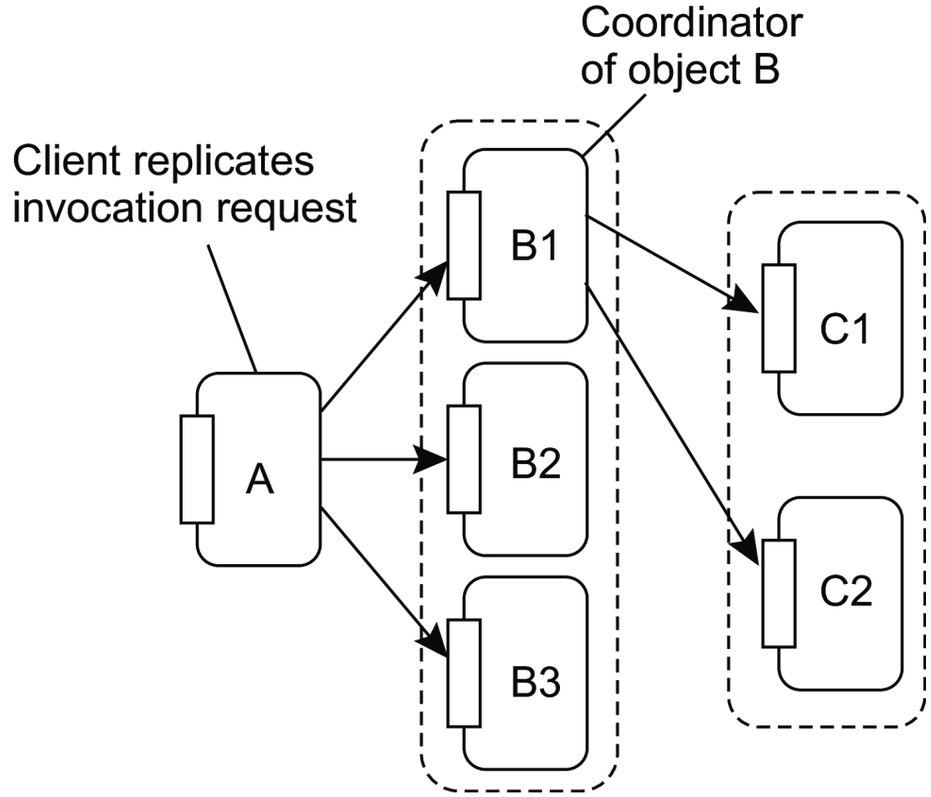
Managing Replicating Objects

- ***Replicad Invocations***
-



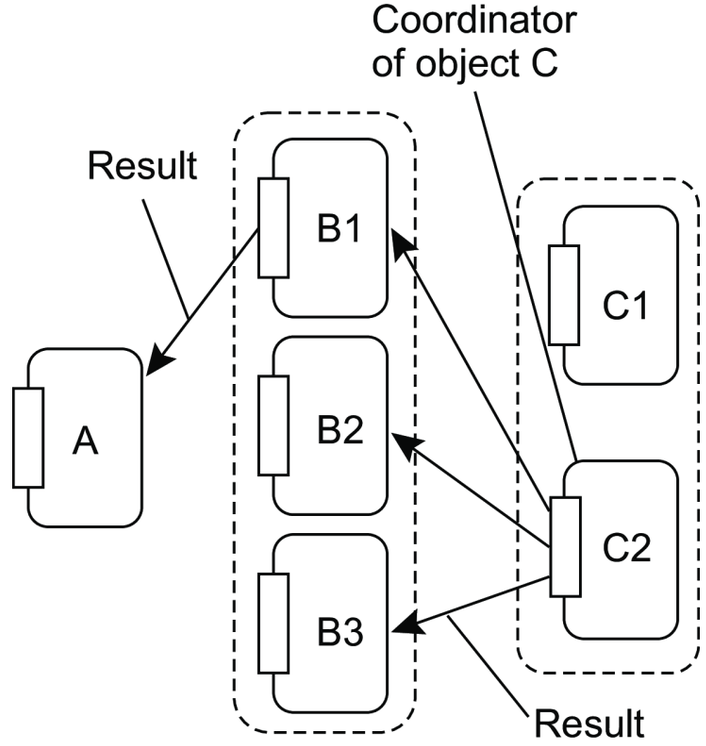
Managing Replicating Objects

- Replicated Invocations:
  1. Forbid it
  2. Replication-aware communication:



Managing Replicating Objects

- Replicated Invocations:
  1. Forbid it
  2. Replication-aware communication:



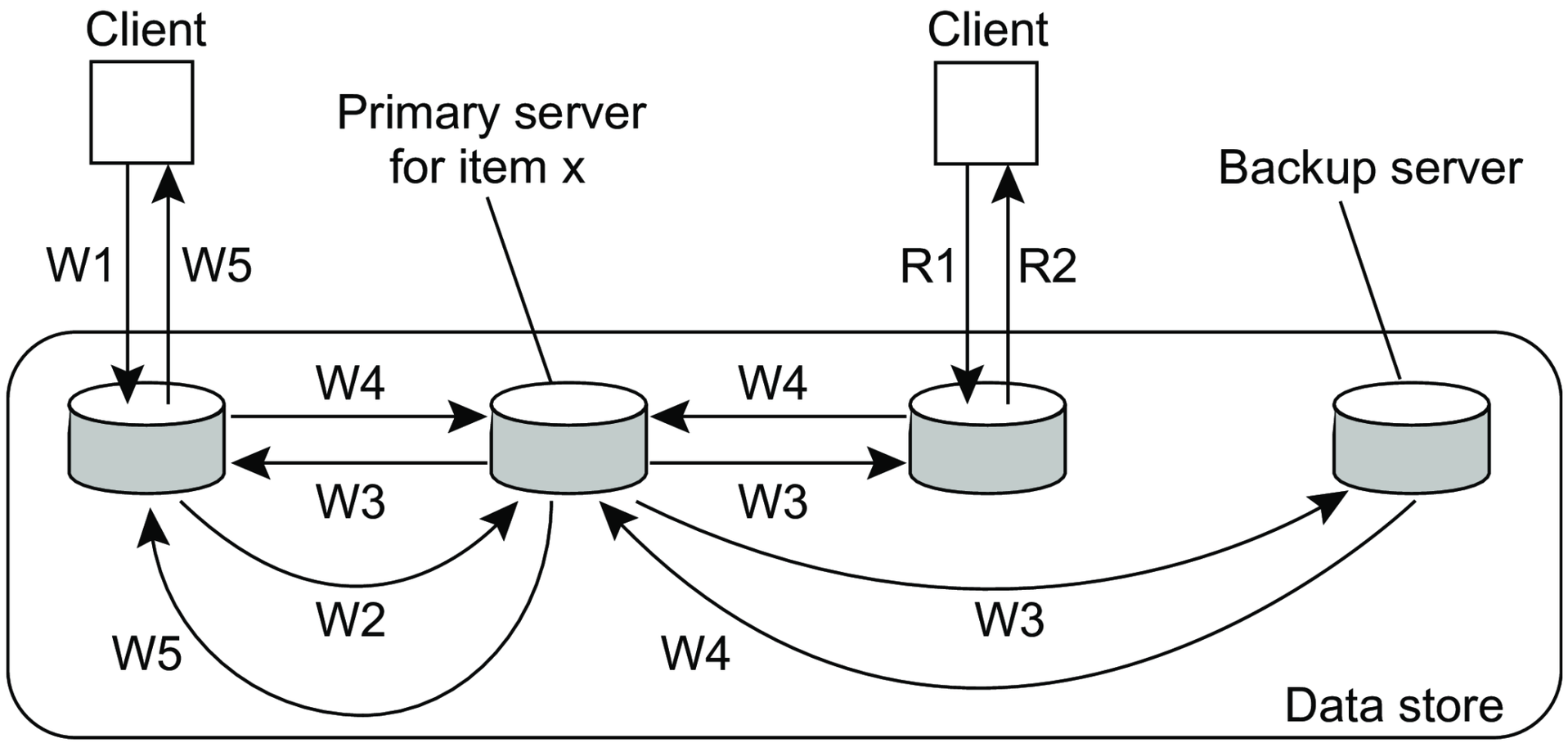
Consistency Protocols
Sequential Consistency

- Sequential Consistency: Primary based protocols
  - Each data item x in the data store has an associated primary, which is responsible for coordinating write operations on x

Consistency Protocols
Sequential Consistency

- Primary backup protocols
  - All write operations are done at the primary server for *x*
  - All reads are done locally
  - Traditionally applied in distributed databases and file systems that require a high degree of fault tolerance. Replicas are often placed on the same LAN.

Consistency Protocols
Sequential Consistency



W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

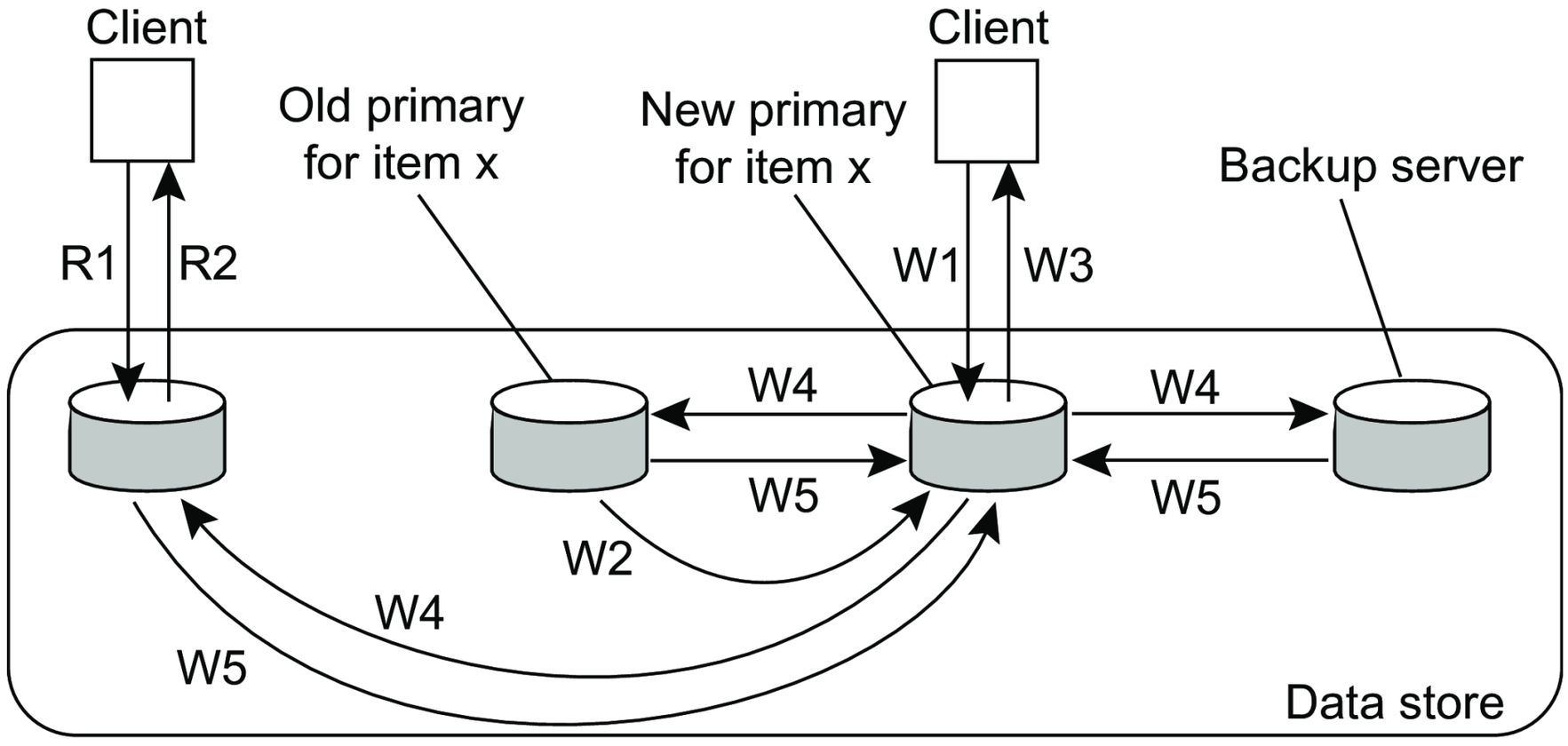
Consistency Protocols
Sequential Consistency

- Primary-backup protocol with local writes
  - Primary copy migrates between processes that wish to perform a write operation
  - Multiple successive write operations can be carried out locally
  - Read operations can be carried out locally
  - Updates need to be propagated in a non-blocking manner

Consistency Protocols
Sequential Consistency

- Primary-backup protocol with local writes
  - Example:
    - Mobile computer becomes the primary copy of all data item it wants to change
    - Mobile computer is disconnected
    - Others do not see the updates until after the computer returns

Consistency Protocols
Sequential Consistency



W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

Consistency Protocols
Sequential Consistency

- Non-blocking local-write primary-based protocols:
  - Used for distributed file systems
  - A fixed central server through which all write operations take place (remote-write primary backup)
  - But: Server temporarily allows one of the servers to perform a series of local updates
  - Afterwards, the updates are propagated to the central server

Consistency Protocols
Sequential Consistency

- Replicated write protocols
  - Write operations are carried out at multiple replicas
  - Schemes where an operation is forwarded to all the replicas
  - Schemes using voting

Consistency Protocols
Sequential Consistency

- Active replication can use a **sequencer**
  - a central coordinator
  - assigns unique sequence numbers and forwards operations to all replica
  - operations are carried out in the order of their sequence number

Consistency Protocols
Sequential Consistency

- Sequencer
  - Use a group of sequencers
    - Use multicast update operations between sequencers
  - Order updates using Lamport's total-ordering mechanism

Consistency Protocols
Sequential Consistency

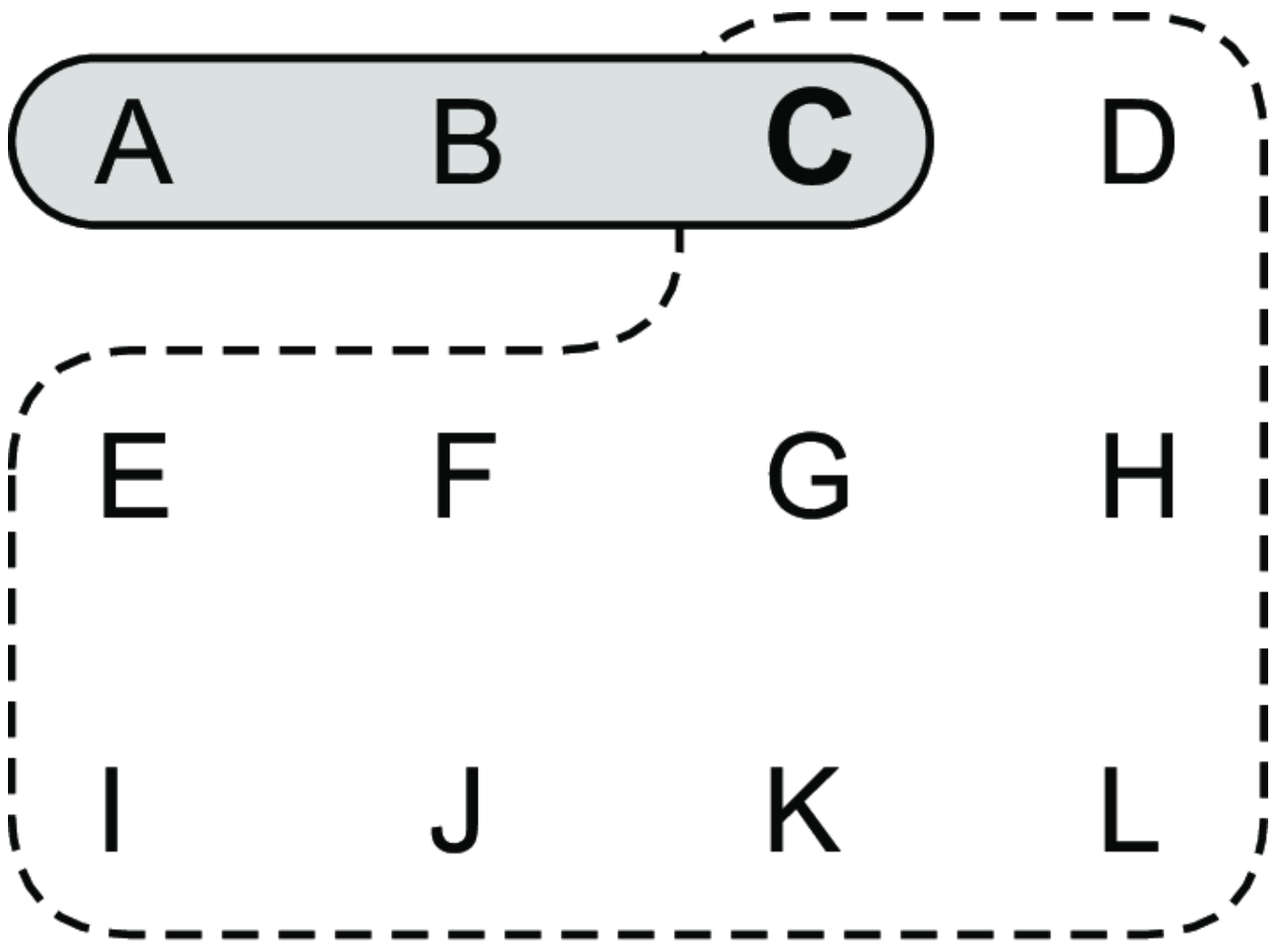
- Using Quora
  - Clients need to gather permission from other servers in order to either read or write
  - Assume N servers that can vote
    - Usually, but not always the servers where the replica reside
    - N_R votes needed to read, N_W votes needed to read

Consistency Protocols
Sequential Consistency

- Using quora
  - Prevent read-write conflicts
    - $N_R + N_W > N$
  - Prevent write-write conflicts
    - $N_W + N_W > N$

Consistency Protocols
Sequential Consistency

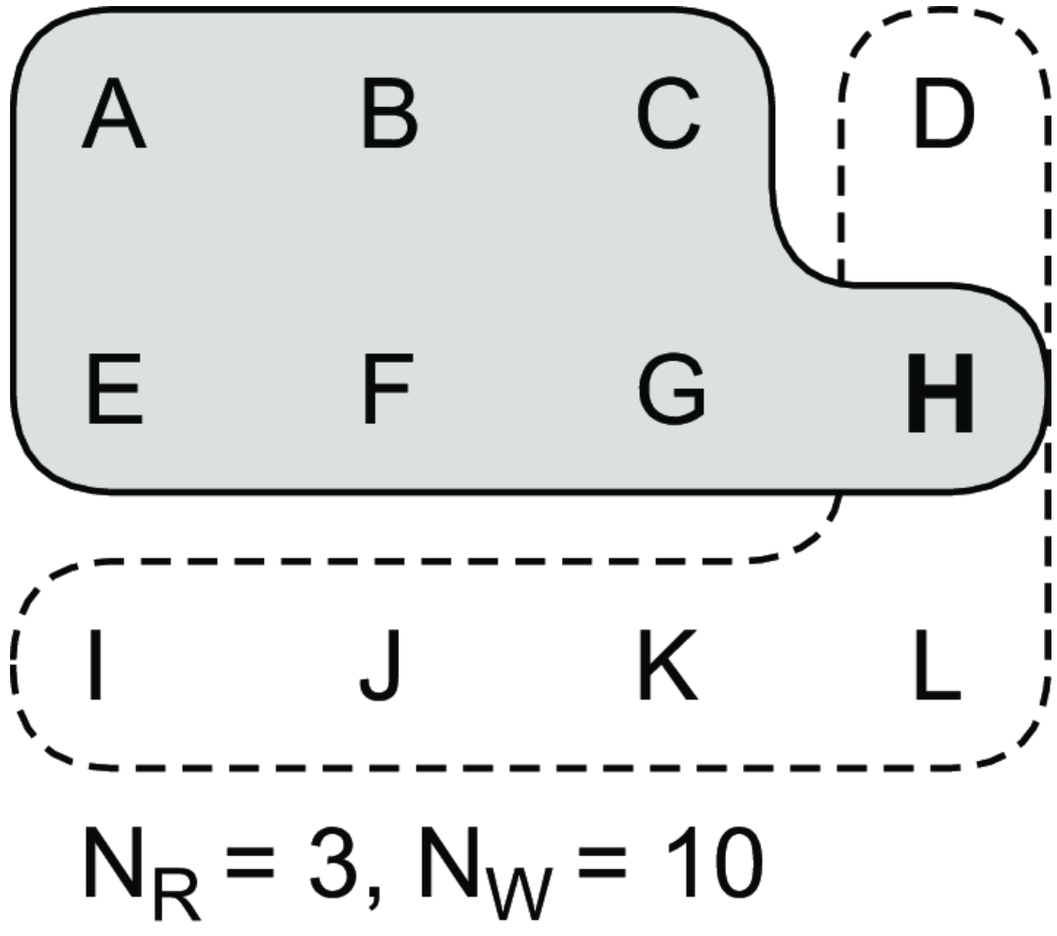
- Example
  - Read goes through



$N_R = 3, N_W = 10$

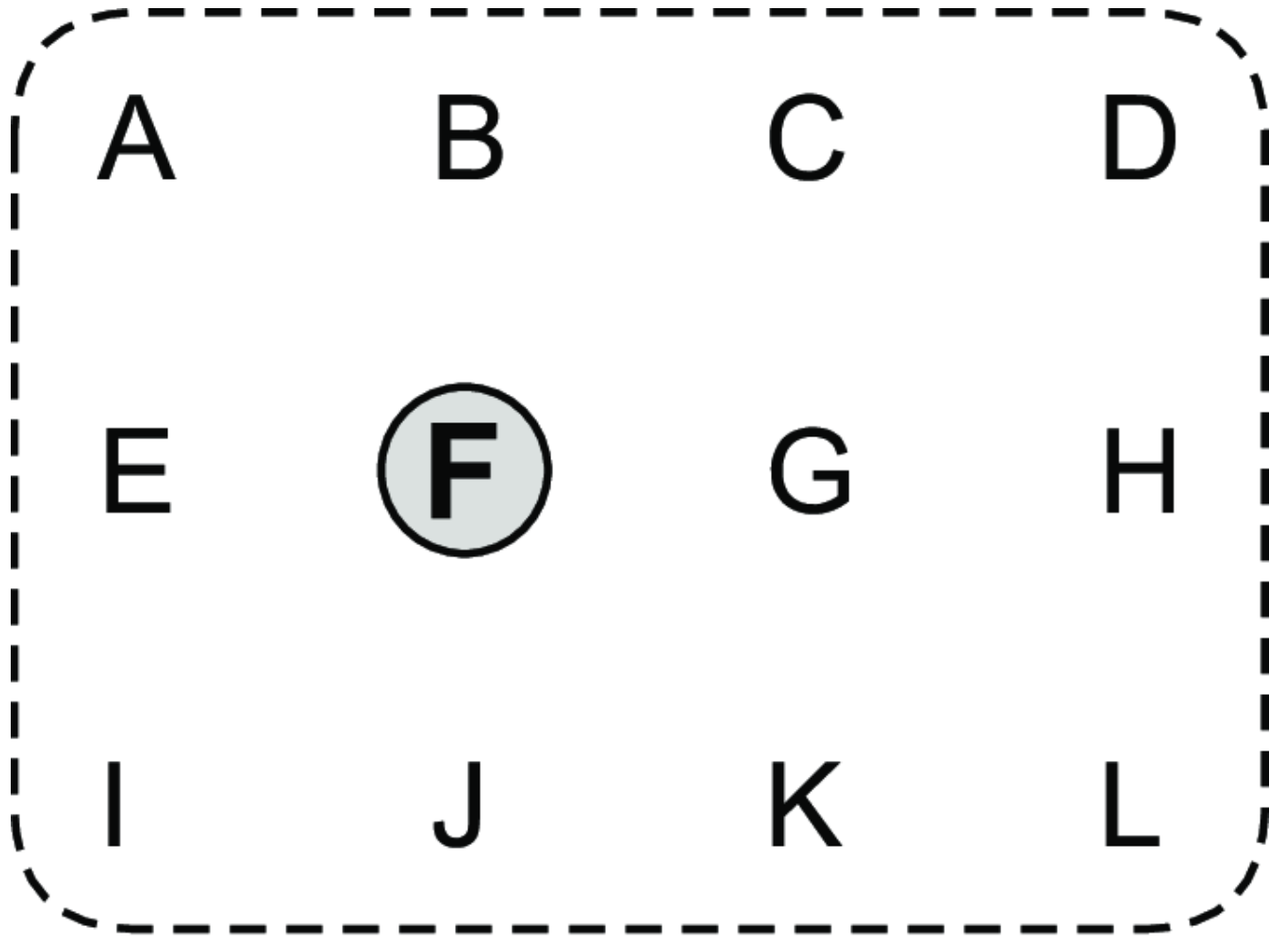
Consistency Protocols
Sequential Consistency

- Example
  - Bad choice of parameters
  - Can read a stale value



Consistency Protocols
Sequential Consistency

- ROWA
  - Read one, write all



$N_R = 1, N_W = 12$

Consistency Protocols
Sequential Consistency

- Quora protocols have developed:
  - Witnesses: sites that do not have a replica
  - Adjustments in case of failures

Consistency Protocols
Cache Consistency

- Cache-coherence protocols
  - Solutions can be based on hardware
    - Example: Snooping in multiprocessor systems
    - Example: Broadcasting

Consistency Protocols
Cache Consistency

- Caching solutions differ by ***coherence detection strategy***
  - E.g. a compiler performs an analysis to detect access to data that can lead to inconsistencies and then puts in additional code to prevent them
  - E.g. in a distributed system check contents with server

Consistency Protocols
Cache Consistency

- Caching solutions differ by ***coherence enforcement strategy***
  - Invalidation vs. propagation of updates

Implementing Client Centric Consistency

- Naïve Implementation
  - Each write operation W is assigned a globally unique identifier by its origin server.
  - For each client, we keep track of two sets:
    - Read set: the (identifiers of the) writes relevant for that client's read operations
    - Write set: the (identifiers of the) client's write operations.

Implementing Client Centric Consistency

- Naïve Implementation
  - ***Monotonic Read***
    - When client C wants to read at server S, C passes its read set. S can pull in any updates before executing the read operation, after which the read set is updated.
  - Notice: Servers will need to log write information

Implementing Client Centric Consistency

- Naïve Implementation
  - **Monotonic Write**
    - When client C wants to write at server S, C passes its write set. S can pull in any updates, executes them in the correct order, and then executes the write operation, after which the write set is updated.

Implementing Client Centric Consistency

- Naïve Implementation
  - **Read-your-writes**
    - When client C wants to read at server S, C passes its write set. S can pull in any updates before executing the read operation, after which the read set is updated.

Implementing Client Centric Consistency

- Naïve Implementation
  - **Writes-follow-reads**
    - bring the selected server up to date with the write operations in the client's **read** set
    - add the identifier of the write operation to the **write** set along with the identifiers in the read set

Implementing Client Centric Consistency

- Better Solution needed:
  - Read and write sets can be very large
  - First, break behavior into *session*
    - Sessions are started and terminated by an application e.g. by the app starting and finishing itself

Implementing Client Centric Consistency

- Improve the presentation of read and write set data
- Use vector clocks
  - Writes of a process at a site are always executed in the order by the process
  - Whenever a server accepts a new write operation W, it assigns it a globally unique identifier along with a time-stamp $\tau(W)$.
    - Time stamps increase with operations

Implementing Client Centric Consistency

- Using two vector clocks
  - $\mathrm{WVC}_i[j]$ time-stamp of the most recent write operation originating from S_j that has been processed by S_i
  - Read and write sets are represented by vector time-stamps
    - $\mathrm{SVC}[j] = \max(\tau(W) \mid \mathit{origin}(W) = S_j)$
    - represents the latest write operations that have been seen by the applications

Implementing Client Centric Consistency

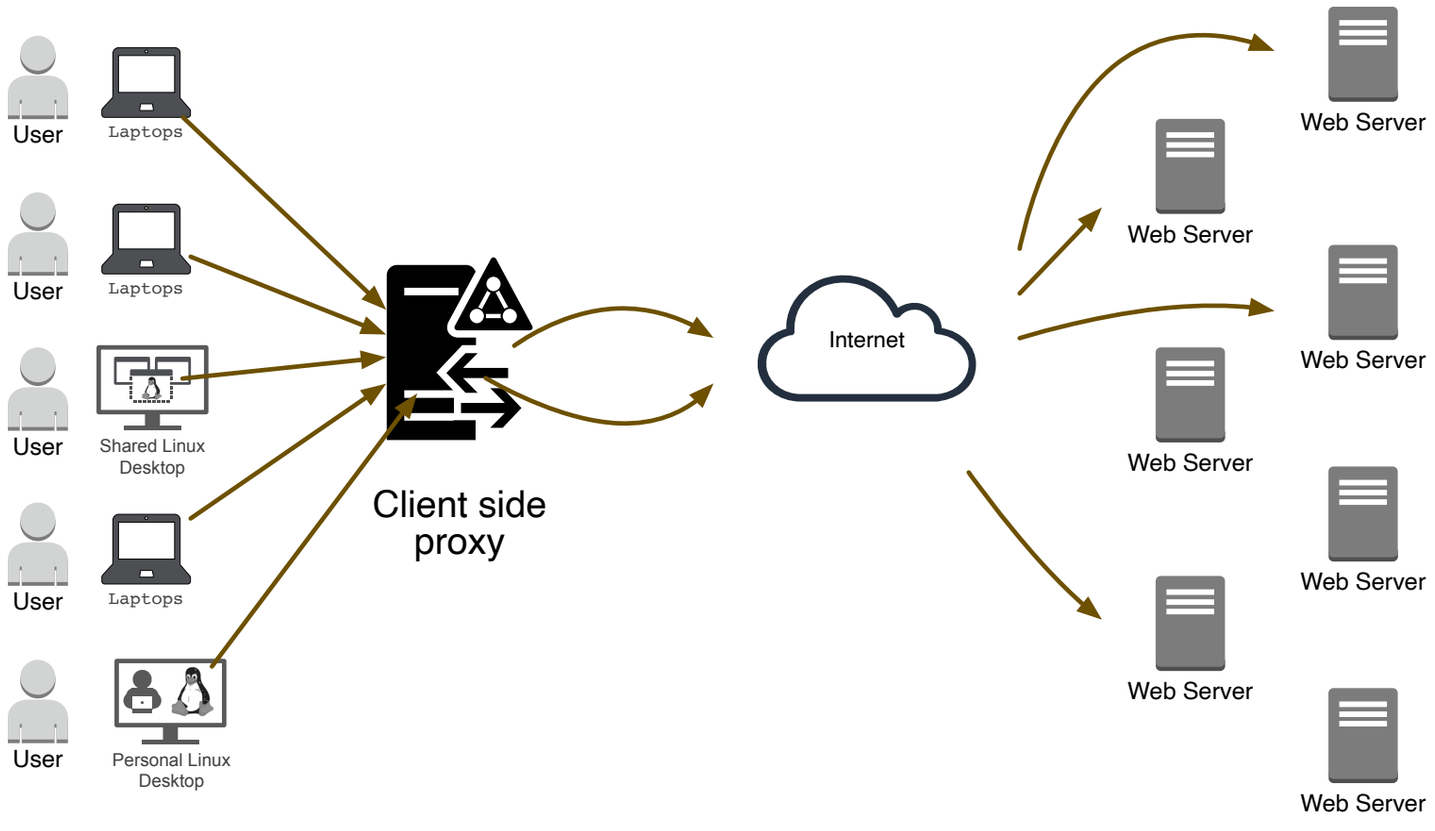
- Using two vector clocks
  - Assume a client logs into Server S_j
  - It passes SVC to S_j
  - If $\text{SVC}_j > \text{WVC}_i[j]$:
    - Then S_i has not seen all the writes from S_j that the client has seen
    - Depending on the consistency model, S_i may now have to fetch these writes
    - Once these are performed:
      - $\text{SVC}[j] \leftarrow \max(\text{SVC}[j], \text{WVC}_i[j])$

Implementing Client Centric Consistency

- Resumé
  - Vector clocks can be used to compress information on complex operations at different sizes

Caching and Replication in the Web

- Client-side caching in the Web
  - Browsers have a simple caching facility
  - Client's site can run a Web proxy
    - The Web proxy acts as a shared cache

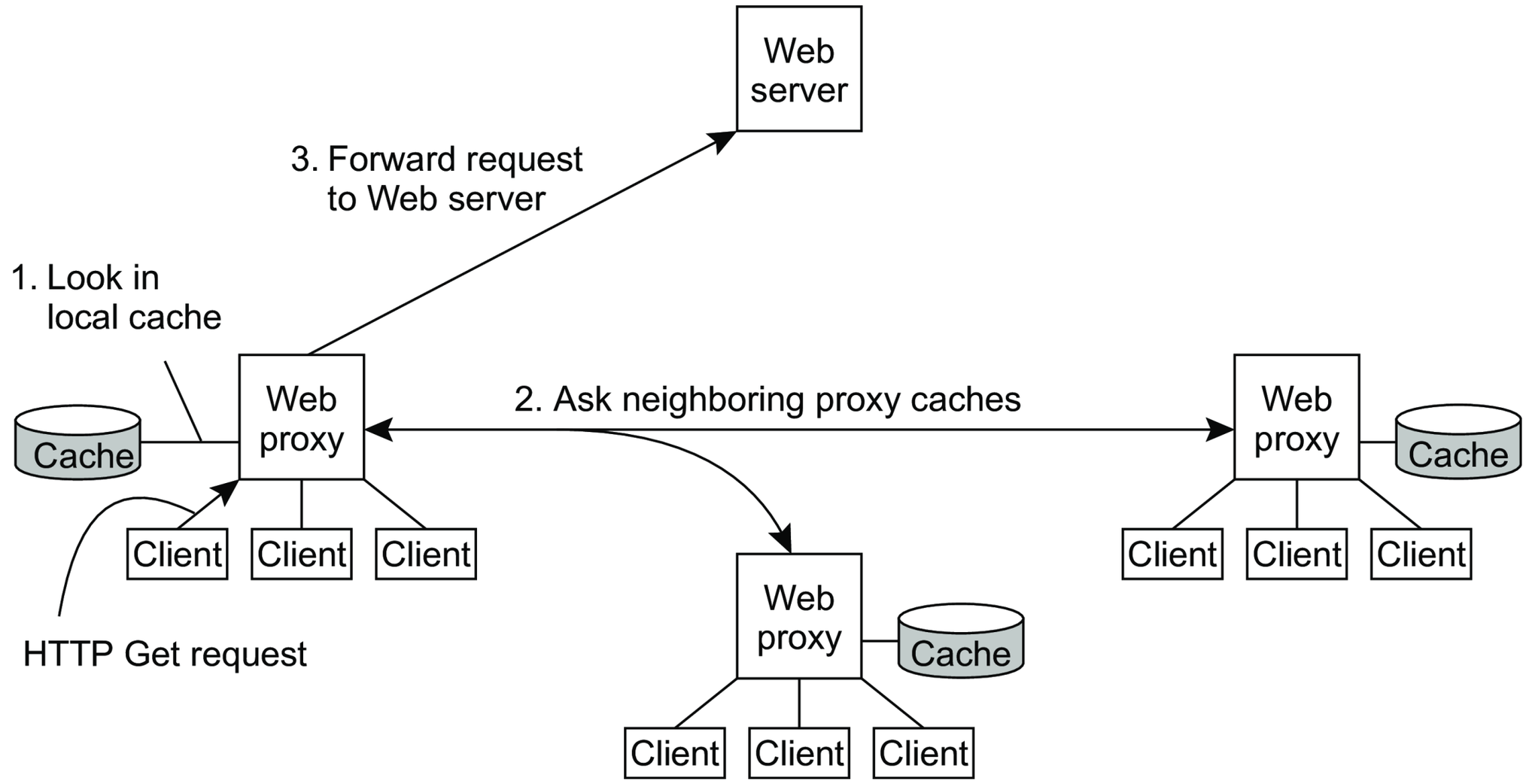


Caching and Replication in the Web

- In addition, many websites have forward proxies

Caching and Replication in the Web

- Cooperative Caching



Caching and Replication in the Web

- Cooperative caching:
  - Idea: Use high-bandwidth / low-latency links to ask neighboring proxy caches for content instead of going to the web-site
- Traditional caching is called hierarchical
  - Trade-offs are very dependent on the setting

Caching and Replication in the Web

- Cache consistency models:
  - Strong consistency: Client Side Proxy always contacts the web-server with a If-Modified-Since HTTP header
  - Weaker consistency: Squid Web proxy has an expiration time that is adjusted based on recency of updates:

$$T_{\text{expire}} = \alpha(T_{\text{cached}} - T_{\text{last-modified}}) + T_{\text{cached}}$$

    - $\alpha = 0.2$ from experience
    - Also uses max and min expiration times

Caching and Replication in the Web

- Alternative to pull-based schemes:
  - Push-based schemes with invalidation
  - Leases

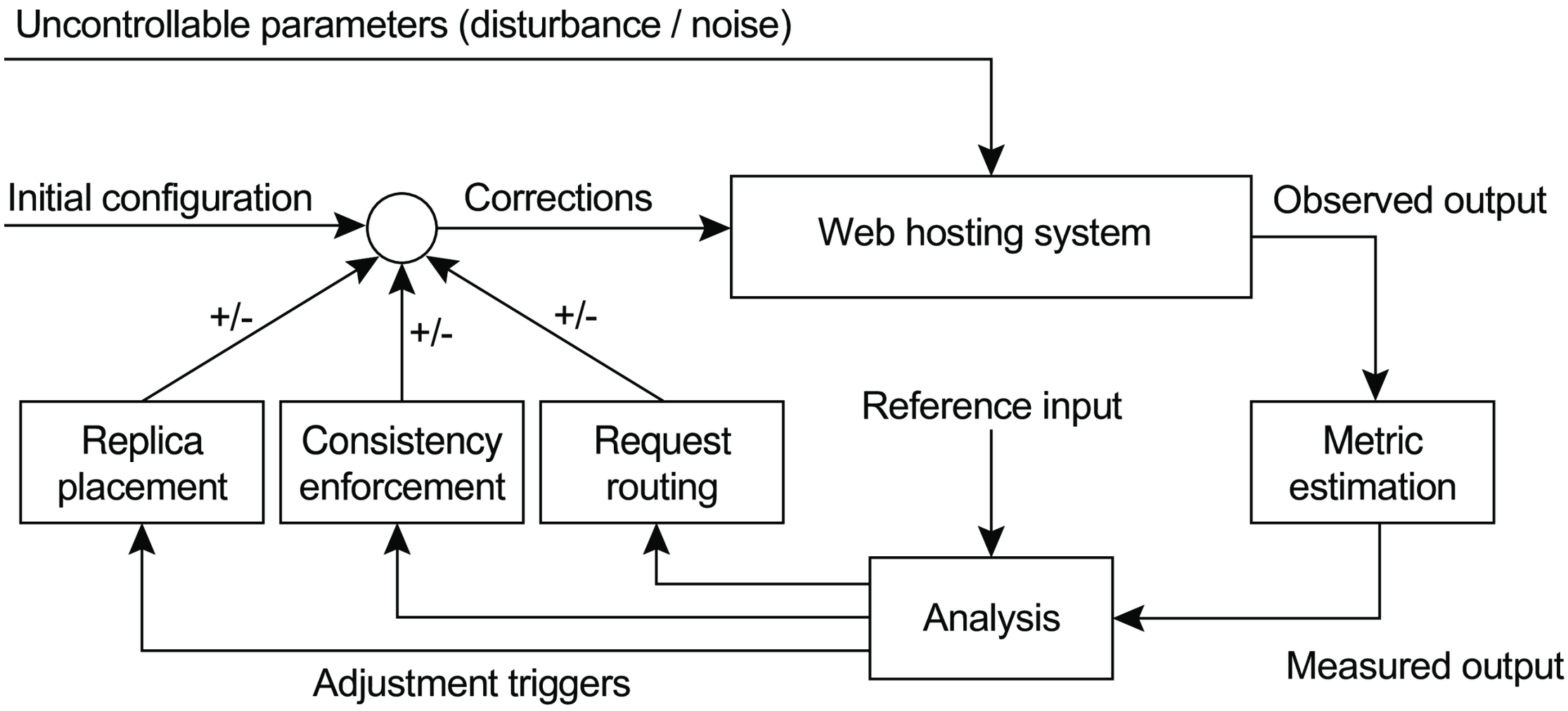
Caching and Replication in the Web

- Many different schemes to manage contents of a cache:
  - If a cache is full, which item to evict?
  - Simple schemes seem to work well:
    - LRU, Oldest, …
  - Could use shadow caches to adjust to the optimal replacement policy

Caching and Replication in the Web

- Content Delivery Networks (CDN)
  - Availability of web sites is of prime importance
  - CDNs become big: Akamei has 365,000 servers in 2022
  - CDN needs to deal with:
    - replica placement, consistency, client-request routing

Caching and Replication in the Web



Caching and Replication in the Web

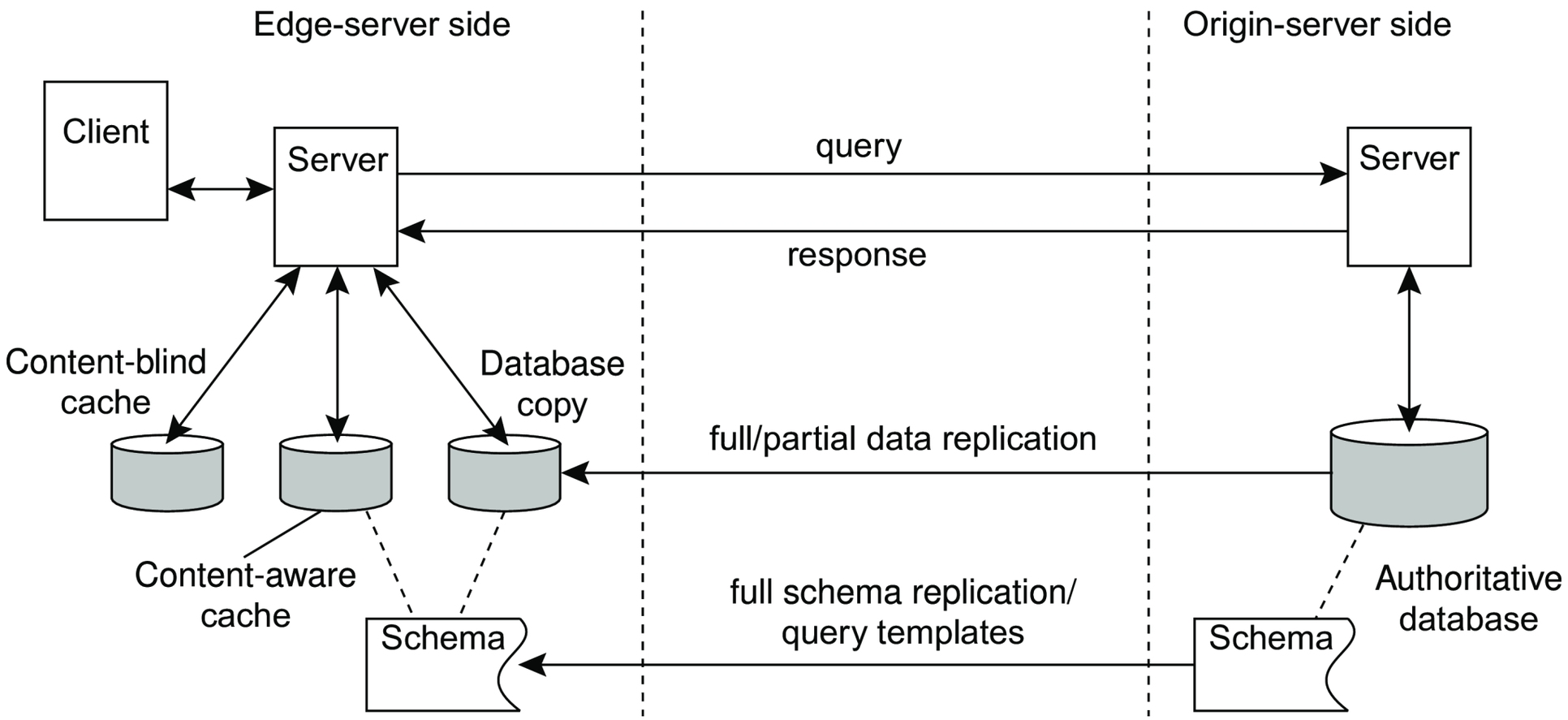
- Can measure:
  - Latency
  - Available bandwidth (for large documents, streams)
- Number of hops between nodes
  - Difficult because of things such as **Multi-Protocol Label Switching (MPLS)**
  - MPLS circumvents network-level routing by using virtual circuit techniques

Caching and Replication in the Web

- Can measure:
  - Network usage metrics
  - Consistency metrics
  - Financial metrics

Caching and Replication in the Web

- Dealing with ***dynamic content***
  - Example: Edge-server: Push data to the edge



Caching and Replication in the Web

- Dealing with ***dynamic content***
  - Partial replication: Only a subset of data is replicated
  - **Content-aware caches**:
    - Edge server stores only data needed for its typical load
      - Edge server makes a **query-containment check** to decide whether it or the website should handle a request

Caching and Replication in the Web

- To avoid the complications of content-aware caches, use content-blind caching
  - E.g. queries are hashed
  - An edge server checks the query hashes to find out whether it has dealt with the query recently
    - In this case it uses the content-blind cache
    - Otherwise, it fetches the result from the web-server and updates the cache

Summary

- Replication is done for
  - Reliability
  - Performance
- Replication causes a consistency problem
  - Strict serializability (databases) is costly
  - Relax consistency
    - Continuous consistency: Set bounds for divergence
    - Weaker consistency models

Summary

- Weaker consistency models
  - A consistency model is a contract between the distributed system and the application developer
  - Data-centric models
  - Client-centric models

Summary

- Caching is a form of replication
  - Widely used in very different environments
  - Techniques depend on the environment:
    - E.g. multi-processors can do snooping