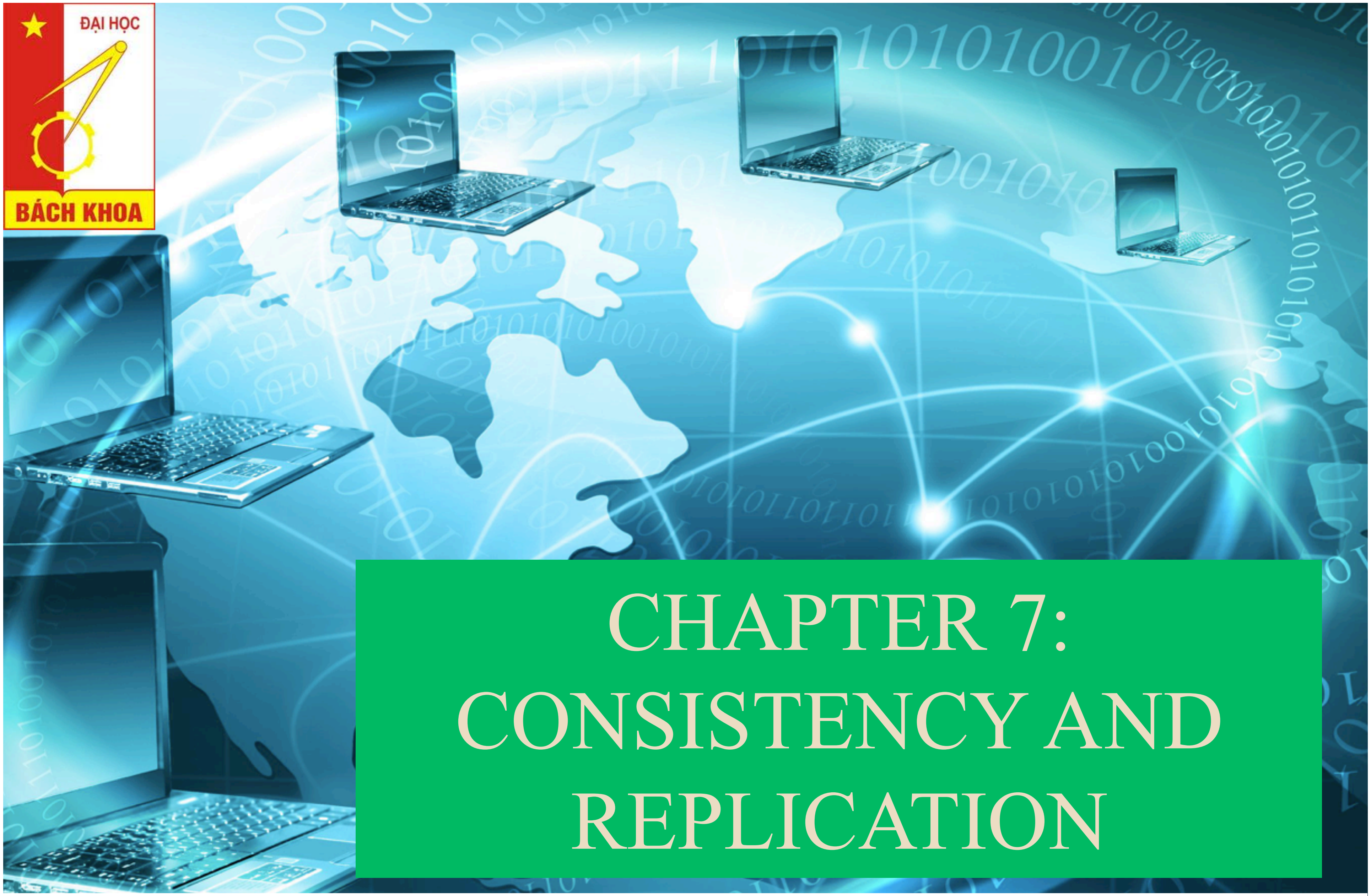
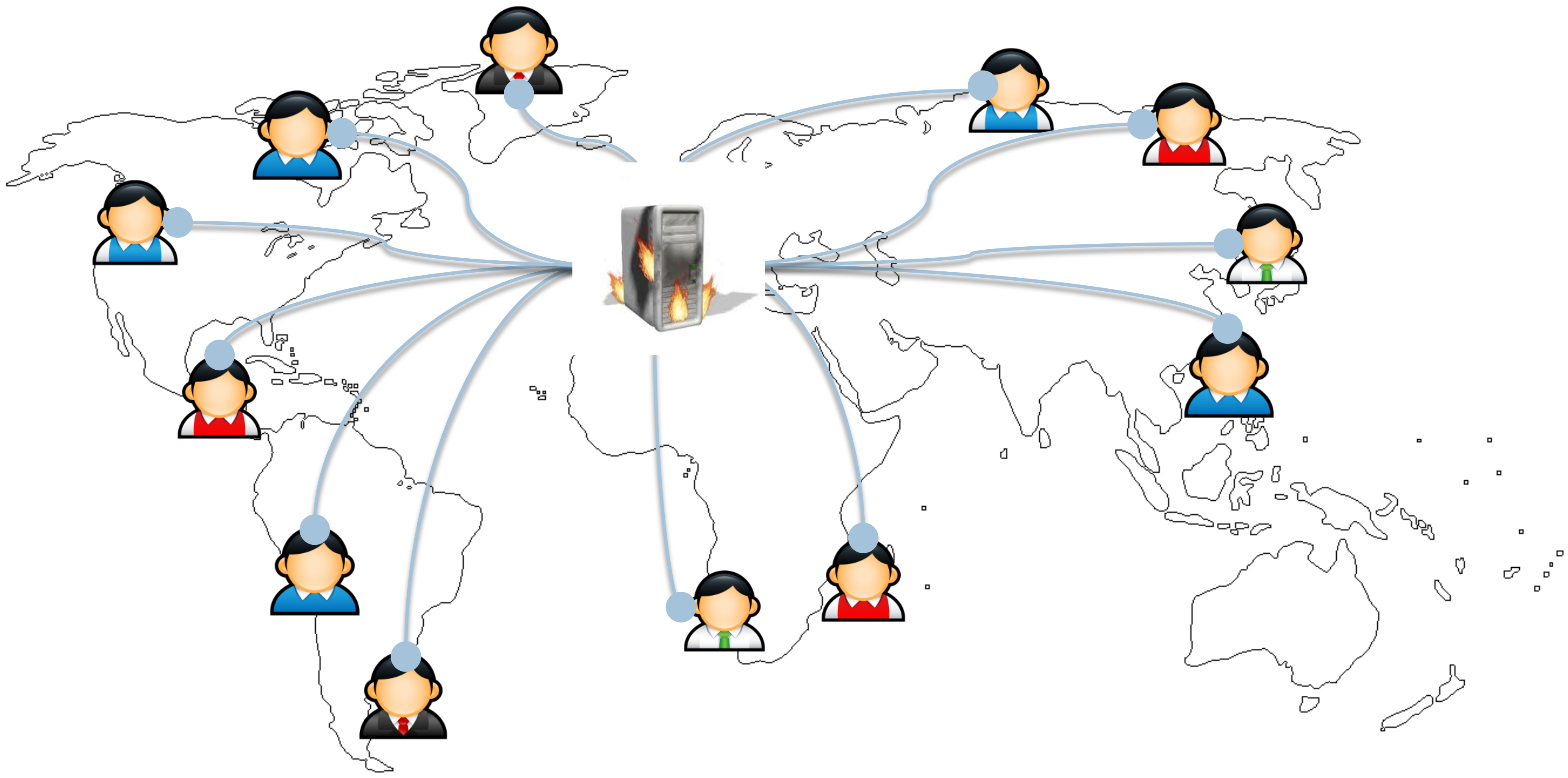


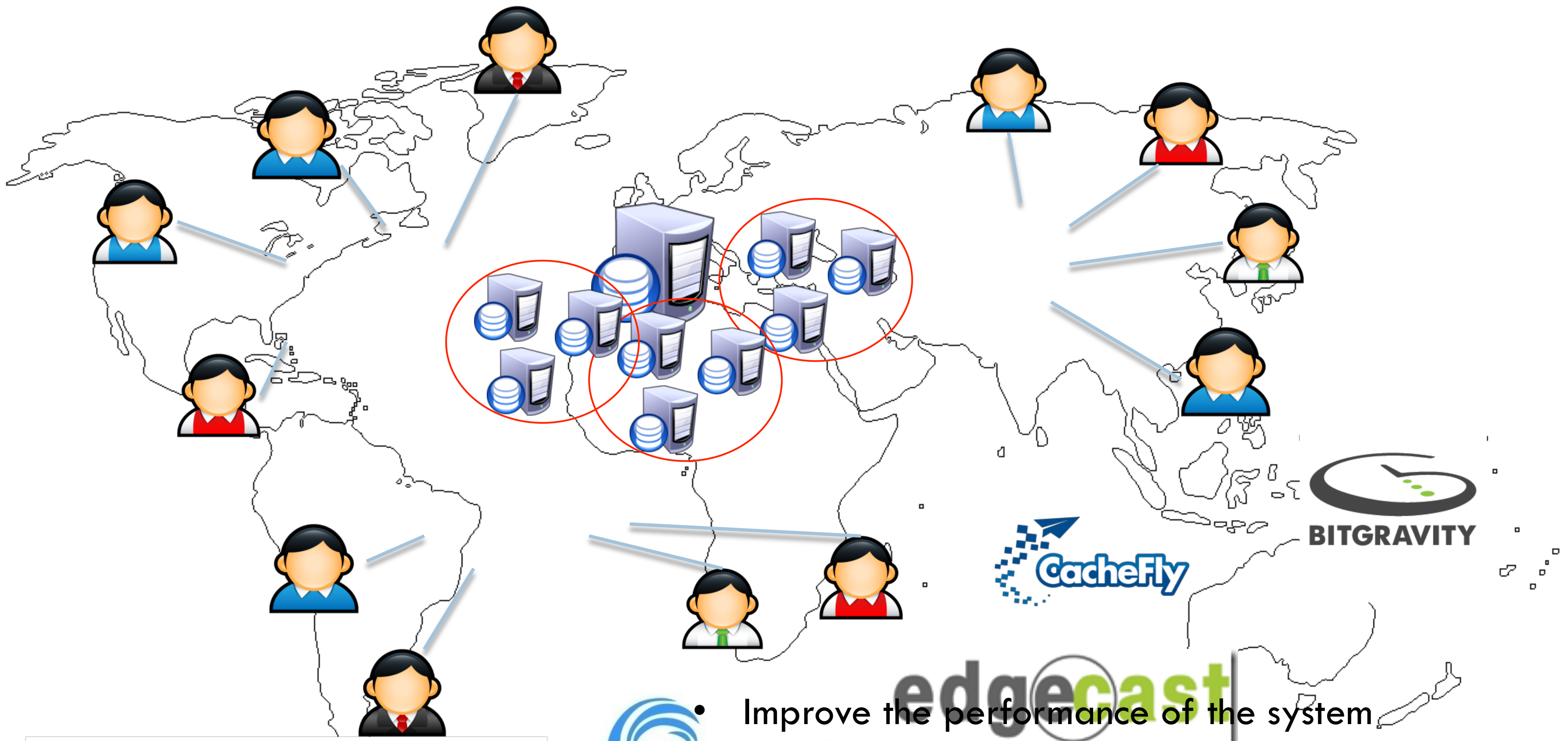
CHAPTER 7: CONSISTENCY AND REPLICATION

Dr. Trần Hải Anh

Problems

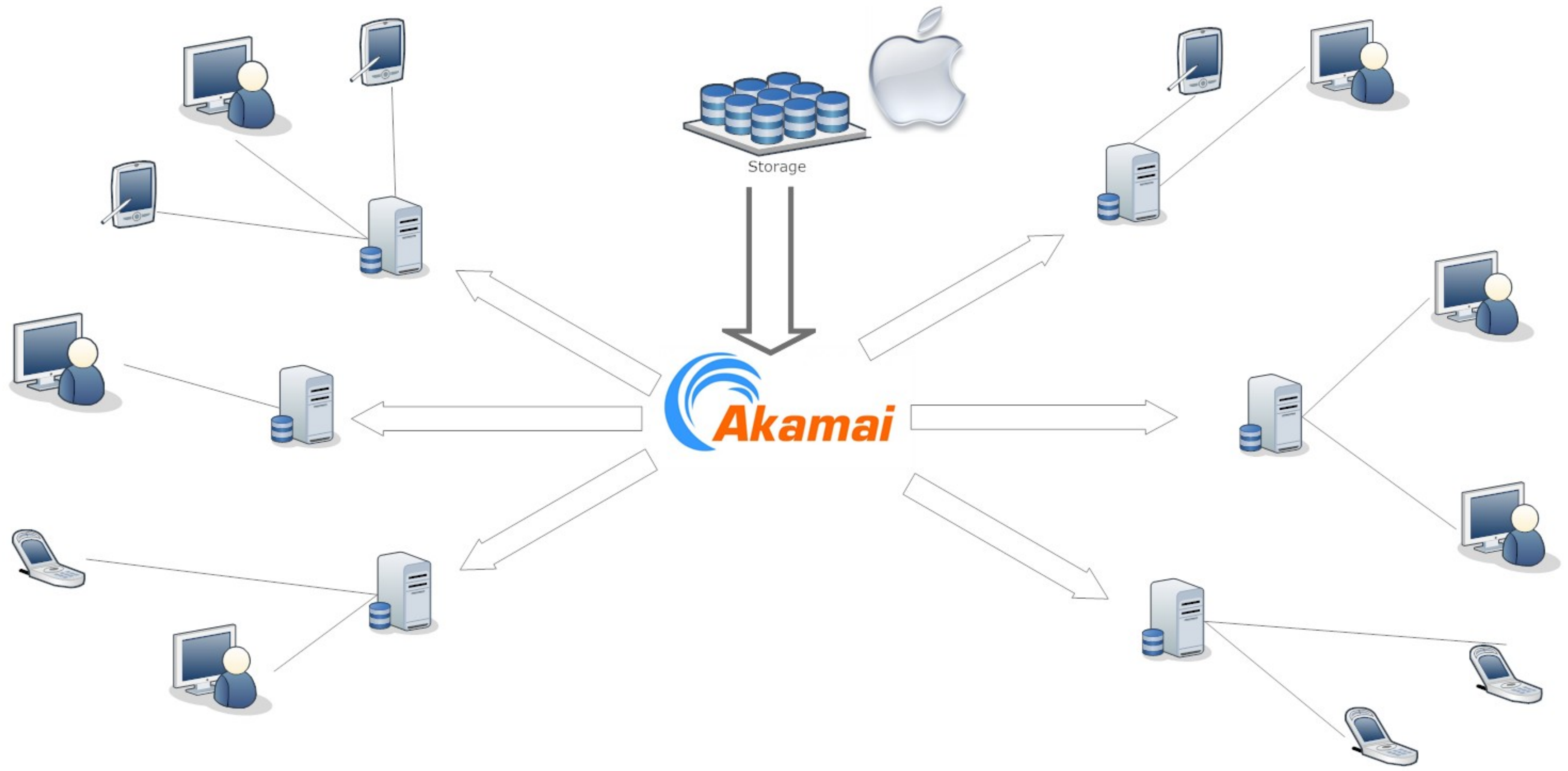


Content Delivery Network



- Improve the performance of the system
- Reduce network load
- Provide better fault tolerance

AKAMAI



Outline

1. Introduction

1.1. Why do we need replication

- Reliability
- Performance
- Scalability (?)

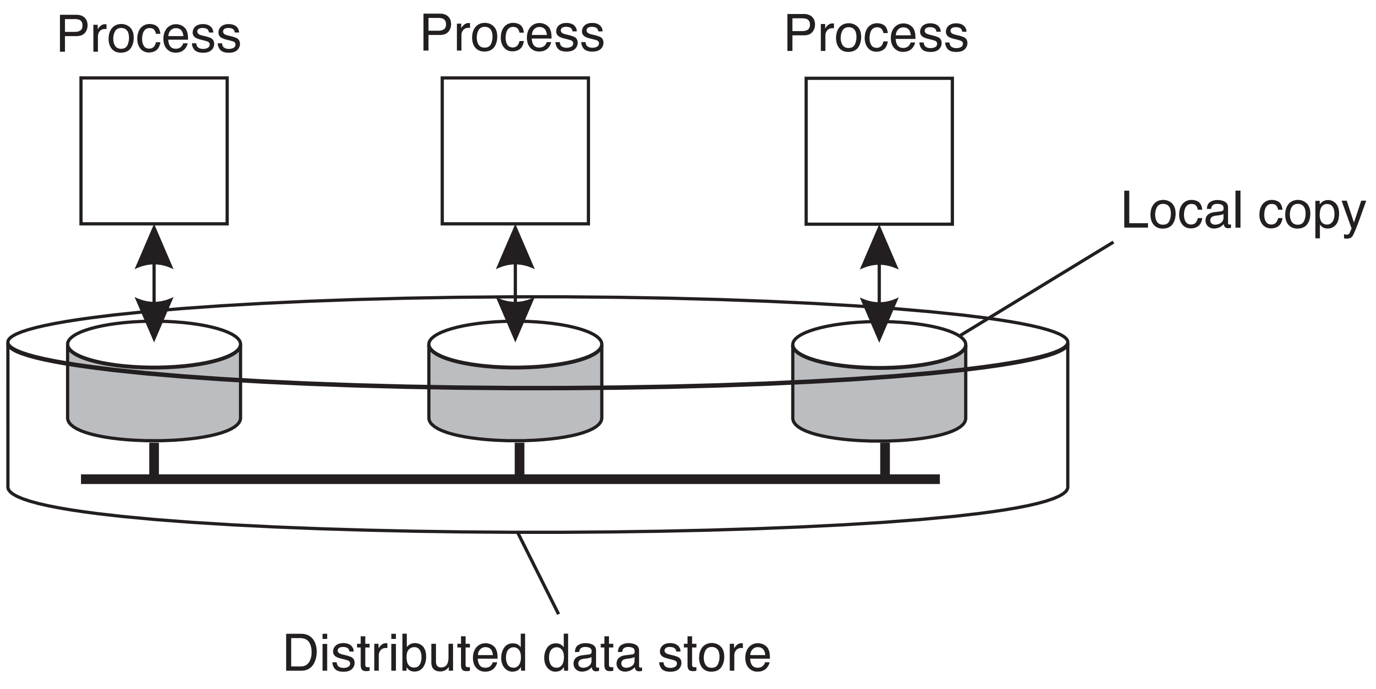
→ Consistency

1.2. Consistency

- Consistency of replicated data
  - Impossible to propagate the updates immediately
  - When? How?
- Strong consistency and Weak consistency
- Trade-off between consistency and performance

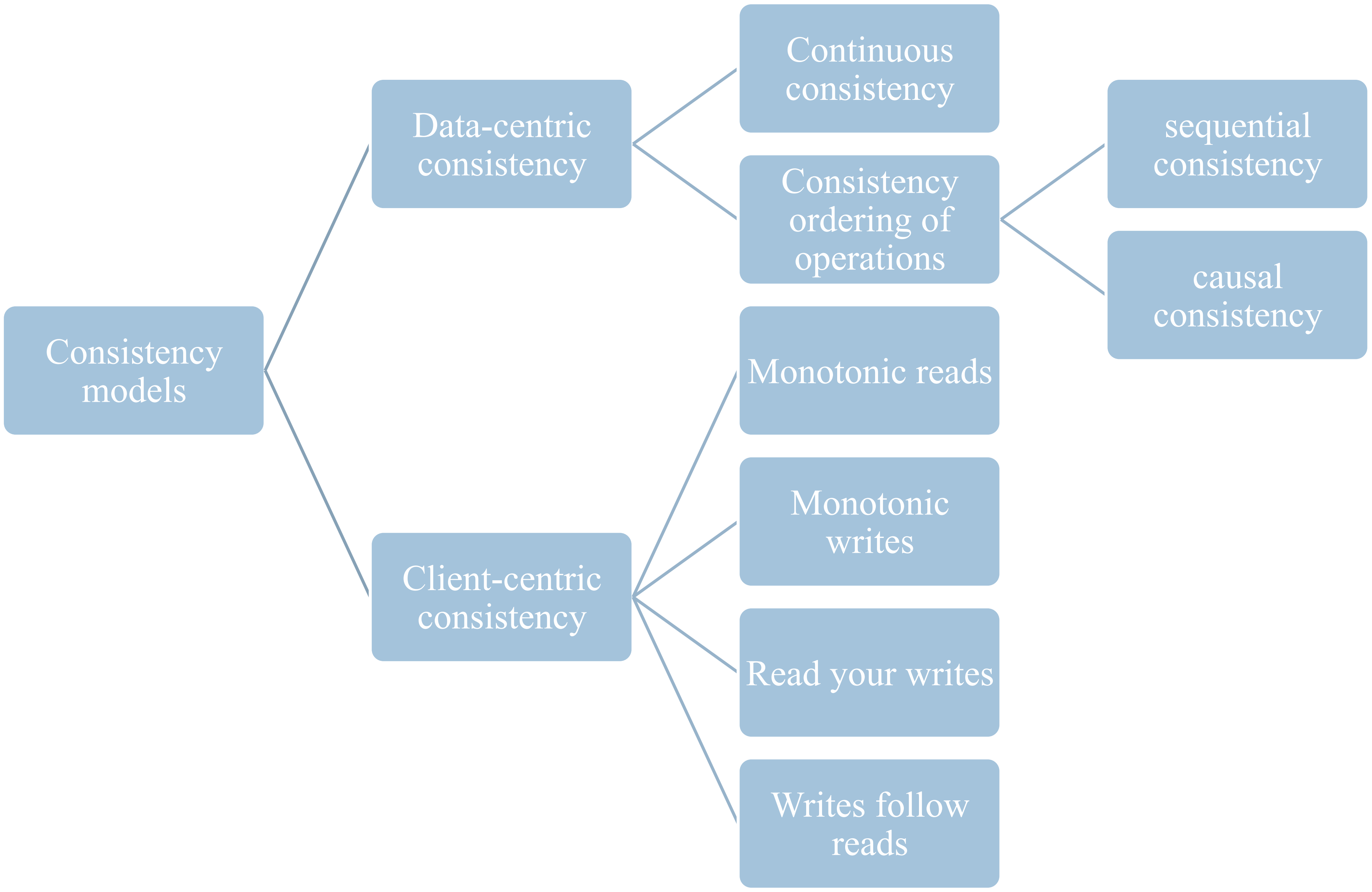
2. Data-centric consistency models

2.1. Distributed data store



Consistency model

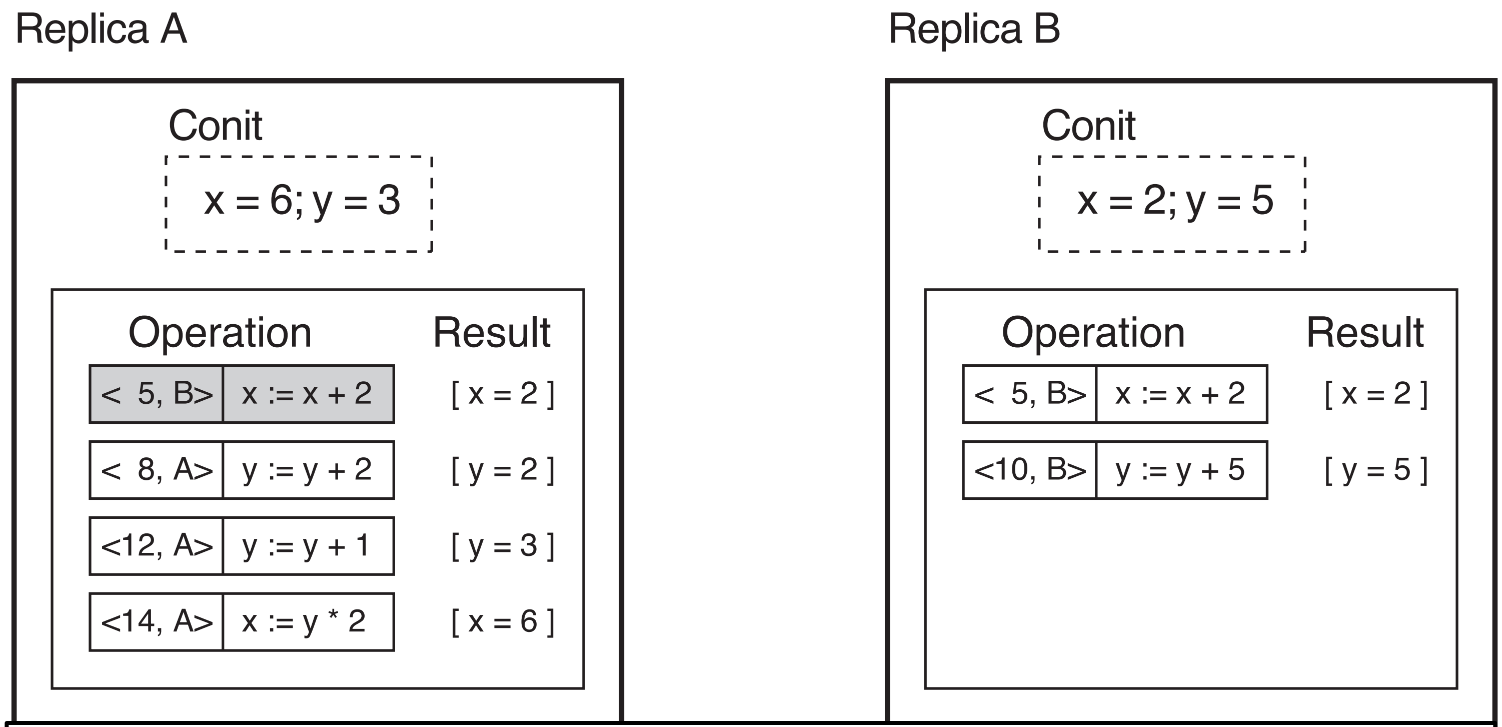
- A contract between processes and the data store.
- If processes agree to obey certain rules, the store promises to work correctly.
- Range of consistency models
- Major restrictions → easy
- Minor restrictions → difficult


Consistency models
Data-centric consistency
Client-centric consistency
Continuous consistency
Consistency ordering of operations
sequential consistency
causal consistency
Monotonic reads
Monotonic writes
Read your writes
Writes follow reads

2.2. Continuous consistency

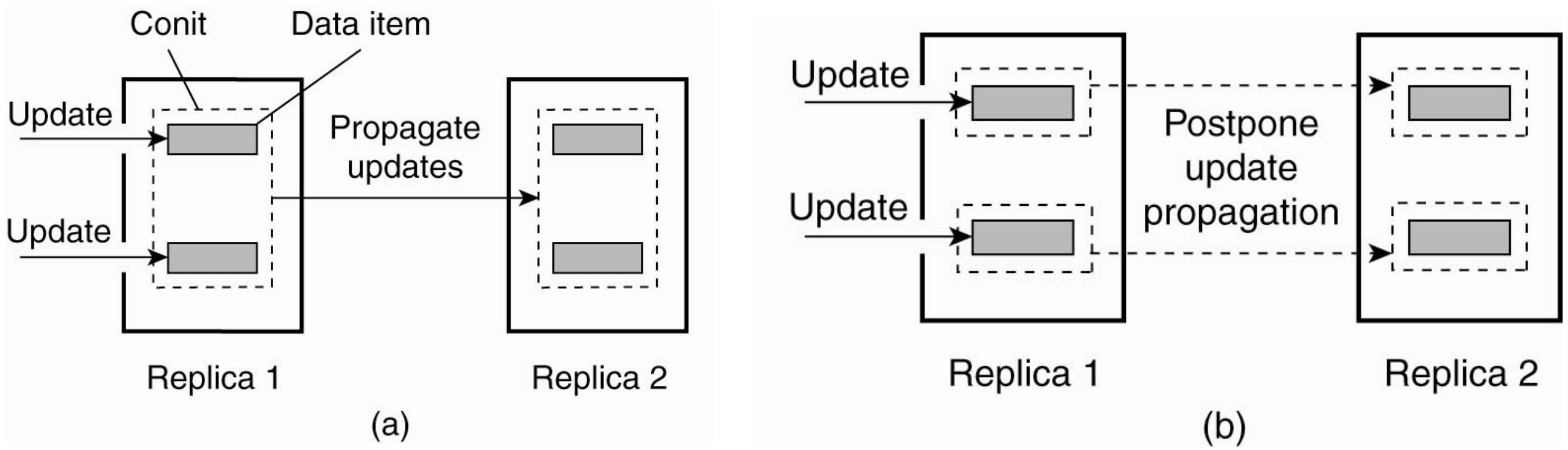
- Factors for defining inconsistencies:
  - Deviation in numerical values
  - Deviation in staleness (the last time a replica was updated)
  - Deviation of ordering of update operations
- When the deviation exceeds a given value, Middleware will perform replication operations to bring the deviation back to the limit.

2.3. Conit (consistency unit)

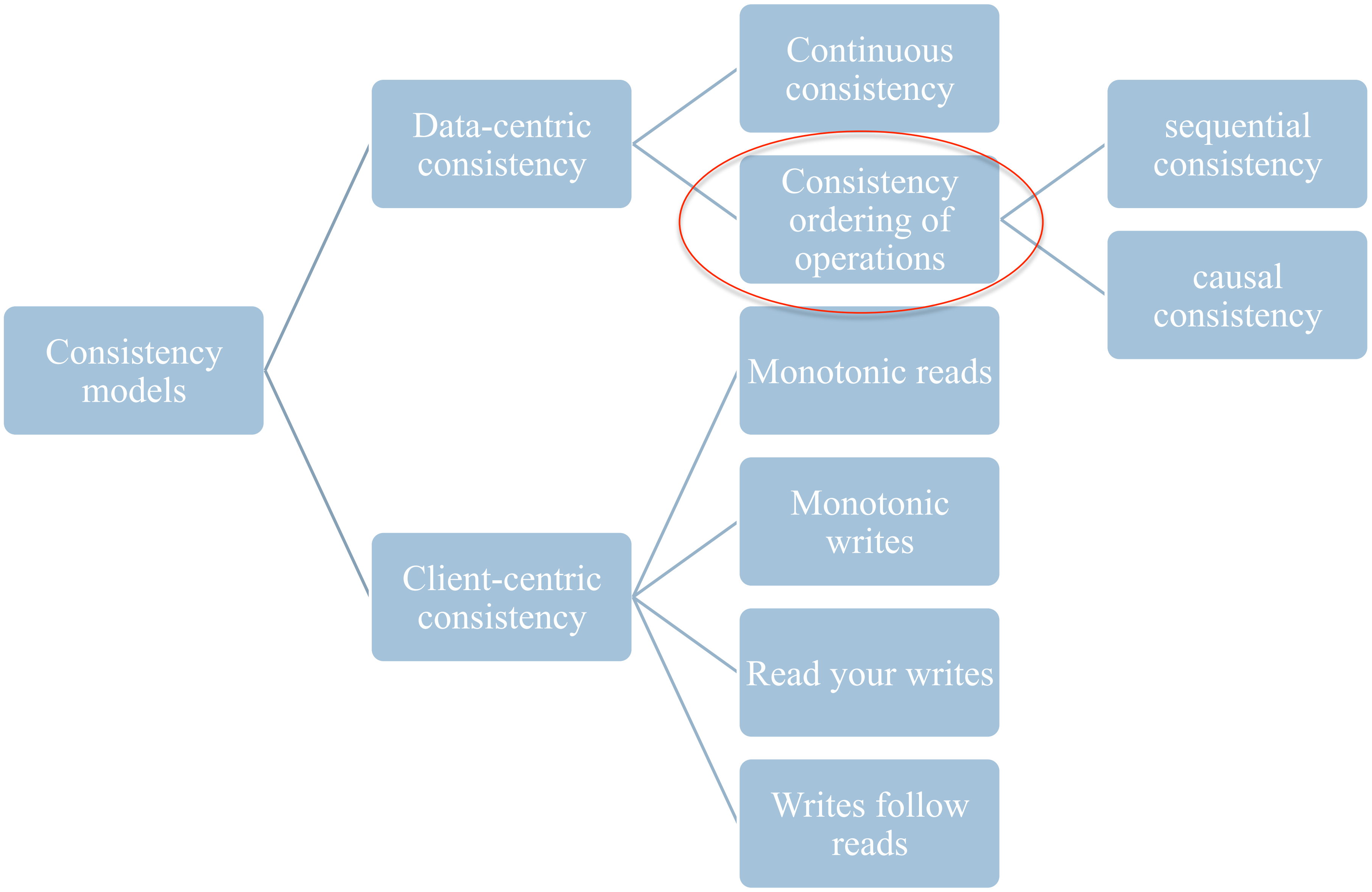


Time:?
Oder:?
Value:?

Size of conit



- A conit represents a lot of data → bring replica sooner in an inconsistent state
- Conit very small → overhead related to managing the conit

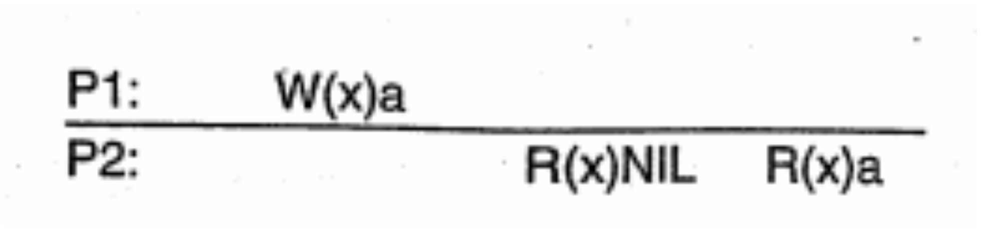

Consistency models
Data-centric consistency
Client-centric consistency
Continuous consistency
Consistency ordering of operations
sequential consistency
causal consistency
Monotonic reads
Monotonic writes
Read your writes
Writes follow reads

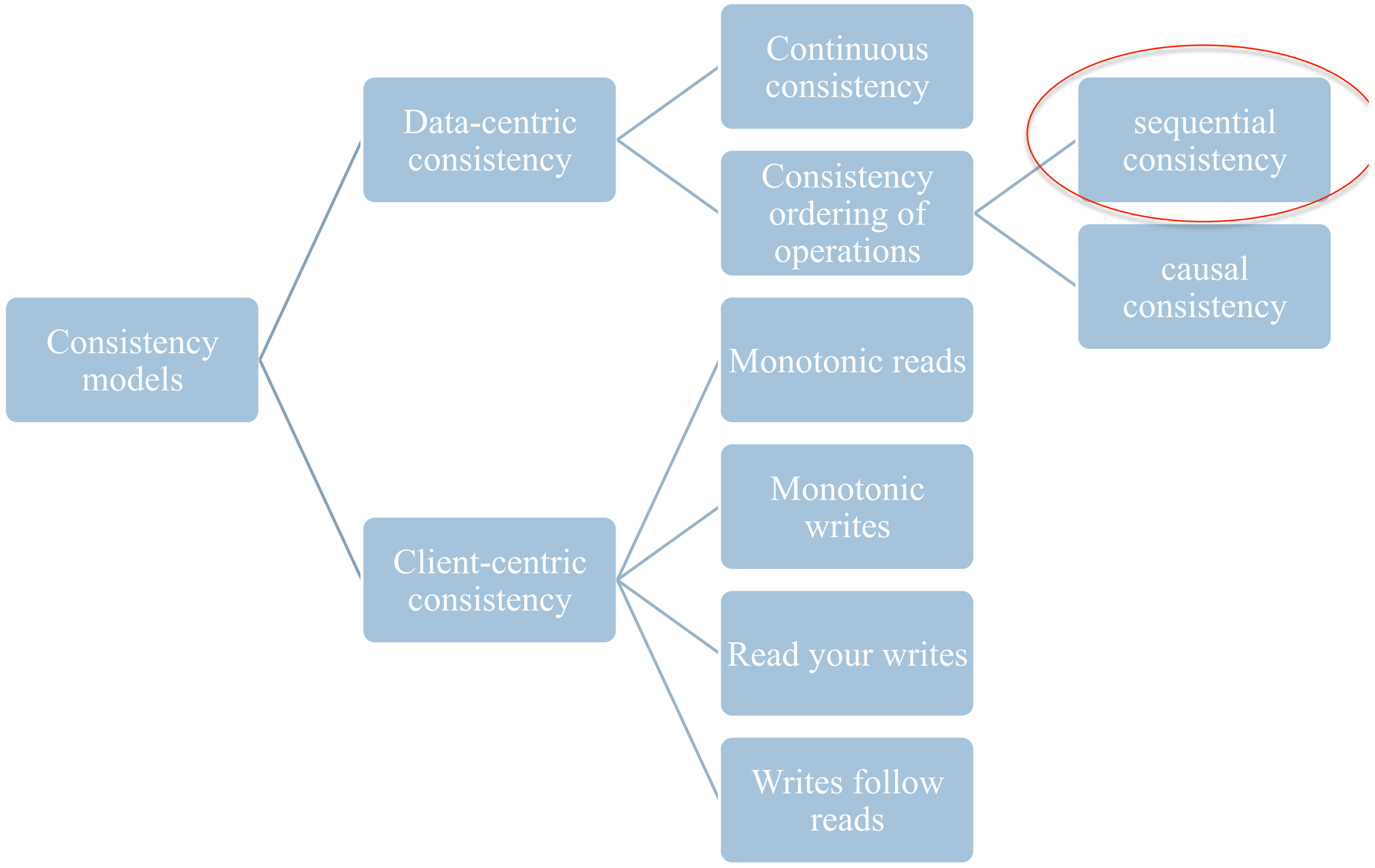
2.4. Consistent Ordering of Operations

- Concurrent programming
- Parallel and distributed computing
- Express the semantics of concurrent accesses when shared resources are replicated
- Deal with consistently ordering operations on shared, replicated data.

Special notation

- Operations on data item *x*
  - Reading: (Ri(x)b)
  - Writing: (Wi(x)a)
  - Initial value of data item is NIL

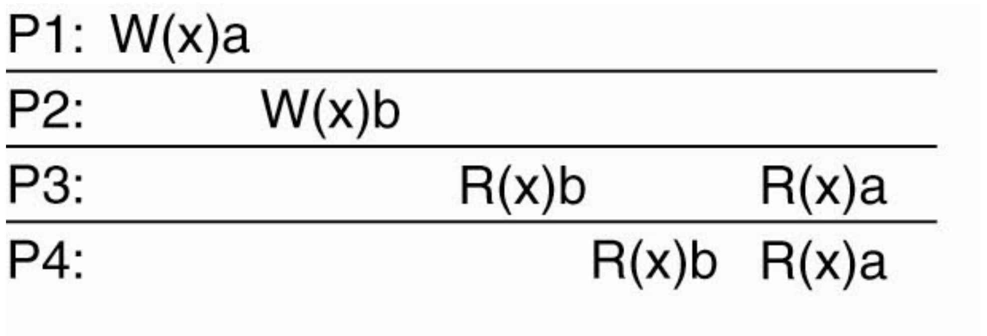



Consistency models
Data-centric consistency
Client-centric consistency
Continuous consistency
Consistency ordering of operations
sequential consistency
causal consistency
Monotonic reads
Monotonic writes
Read your writes
Writes follow reads

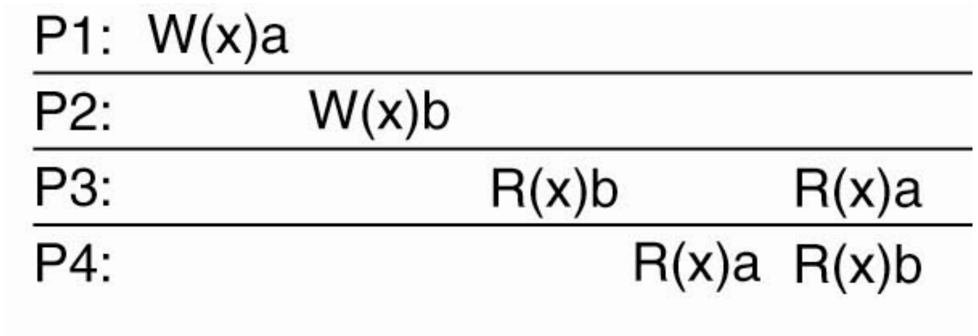
Sequential consistency

- When processes run concurrently on different machines, any valid interleaving of read and write operations is acceptable behavior
- All processes see the same interleaving of operations
- *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 this sequence in the order specified by its program.*

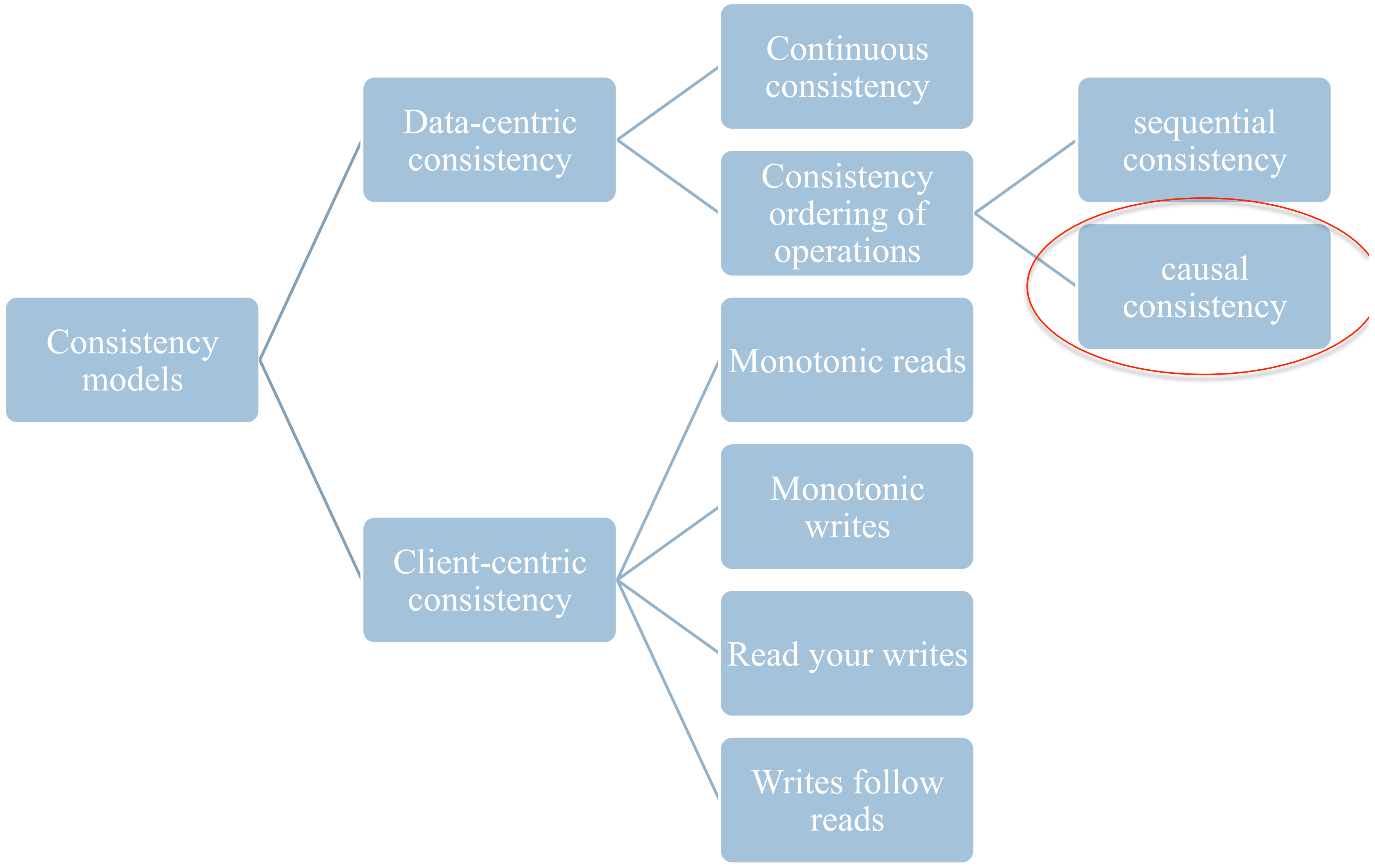
Example


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

(a)


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

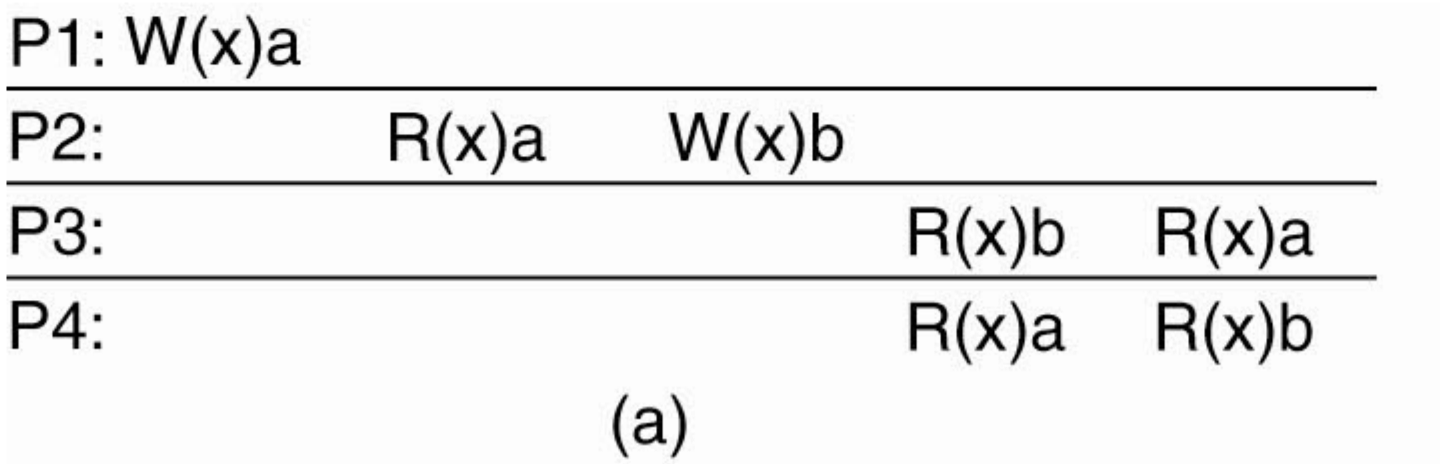
(b)


Consistency models
Data-centric consistency
Client-centric consistency
Continuous consistency
Consistency ordering of operations
sequential consistency
causal consistency
Monotonic reads
Monotonic writes
Read your writes
Writes follow reads

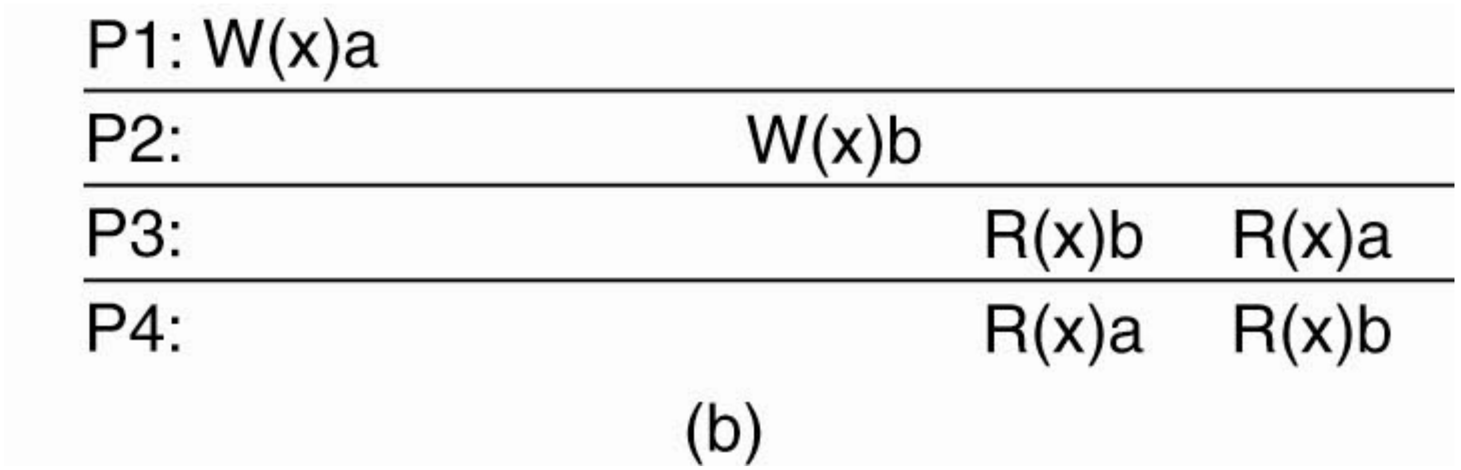
Causal consistency

- A distinction between events that are potentially causally related and those are not.
- *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.*

Causal consistency (cont.)



(a)



(b)

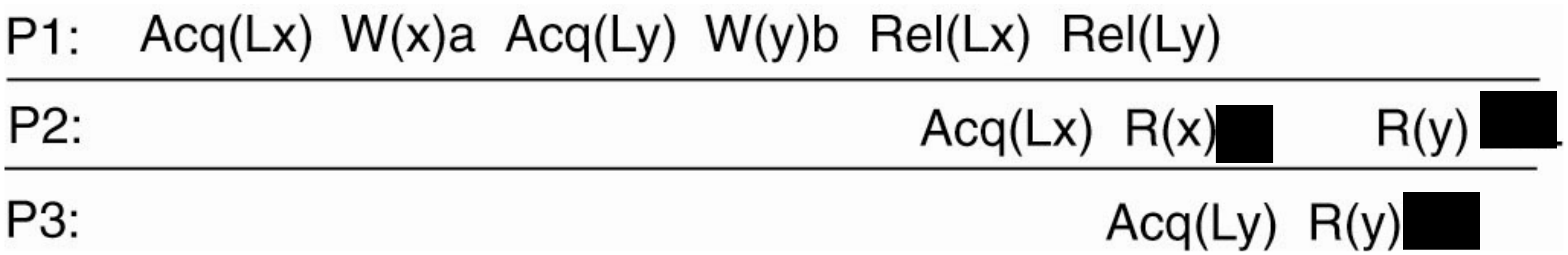
Grouping operations

- Sequential and causal consistency are defined at the level of read and write operations → appropriate for the hardware level (shared memory multiprocessor systems) → did not match the granularity as provided by applications.
- At application level: *read* and *write* operations are bracketed by the pair: ENTER_CS and LEAVE_CS

3 conditions

- A process does an acquire only after all the guarded shared data have been brought up to date.
- Before updating a shared data item, a process must enter a critical section.
- If a process wants to enter a critical region, it must check with the owner of the synchronization variable guarding to fetch the most recent copies

Example



P1: Acq(Lx) W(x)a Acq(Ly) W(y)b Rel(Lx) Rel(Ly)

P2: Acq(Lx) R(x)[redacted] R(y)[redacted]

P3: Acq(Ly) R(y)[redacted]

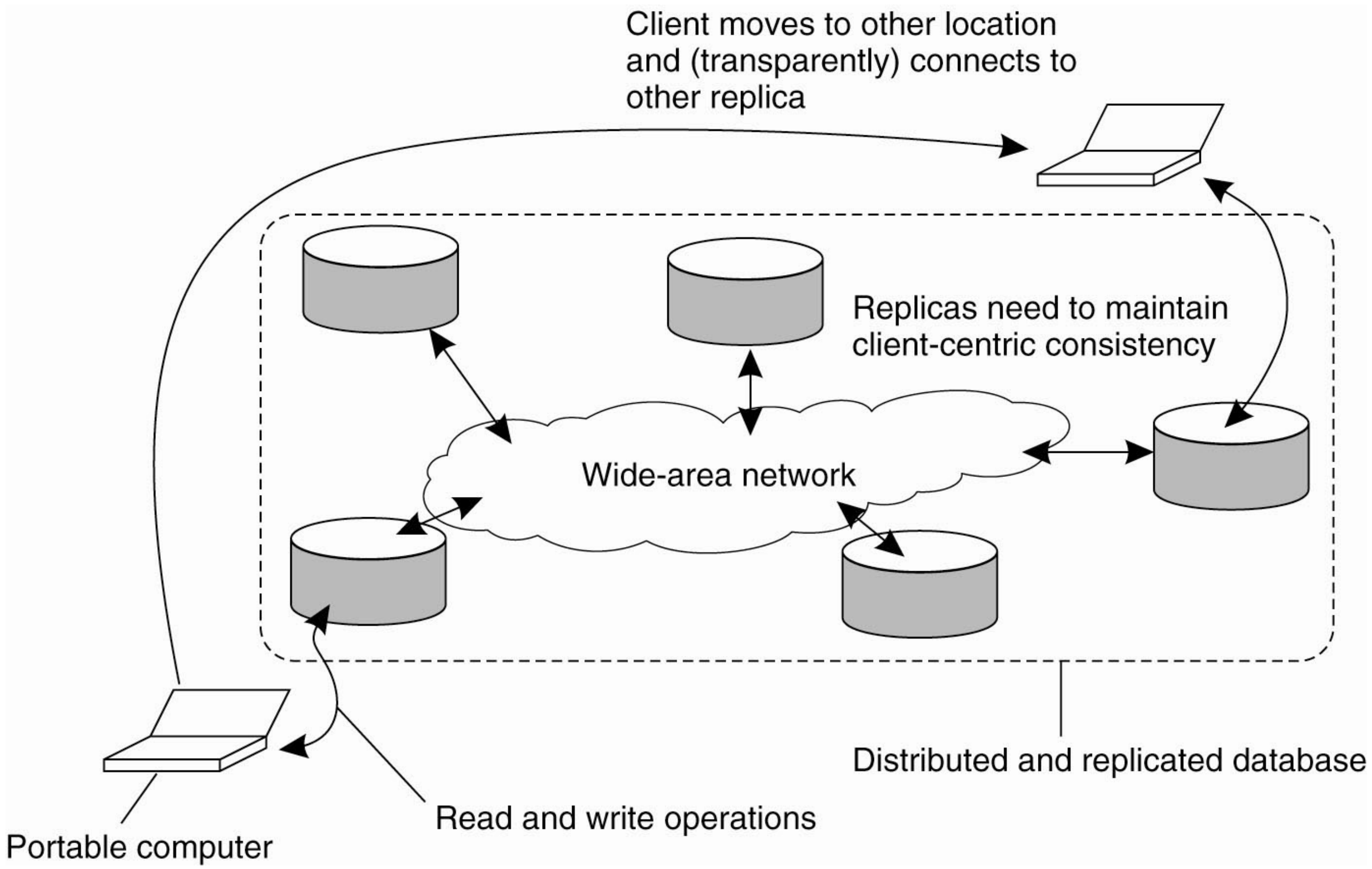
3. Client-centric consistency

3.1. Eventual consistency
3.2. Monotonic reads
3.3. Monotonic writes
3.4. Read your writes
3.5. Writes follow reads

3.1. Eventual Consistency

- Consider two services: DNS, WWW
- Very little number of writes (updates), huge number of reads
- No write-write conflict, only the read-write conflicts.
- These systems tolerate a relatively high degree of inconsistency
- If no updates take place for a long time, all replicas will gradually become consistent.

Problem of Eventual Consistency


Client moves to other location and (transparently) connects to other replica
Replicas need to maintain client-centric consistency
Wide-area network
Distributed and replicated database
Read and write operations
Portable computer

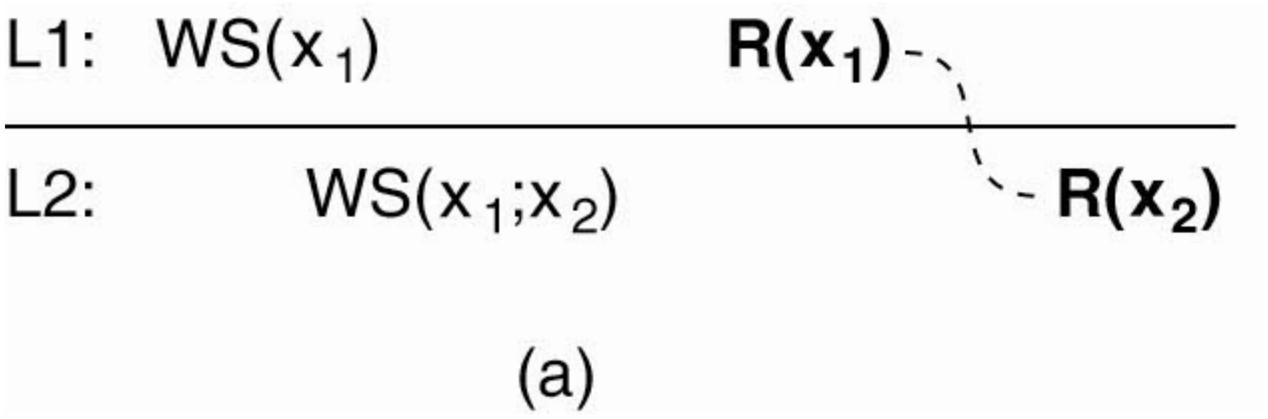
Client-centric consistency

- Provide guarantees for a single client concerning the consistency of accesses to a data store by that client.
- No guarantees for concurrent accesses by different clients.
- 4 types:
  - Monotonic reads
  - Monotonic writes
  - Read your writes
  - Writes follow reads

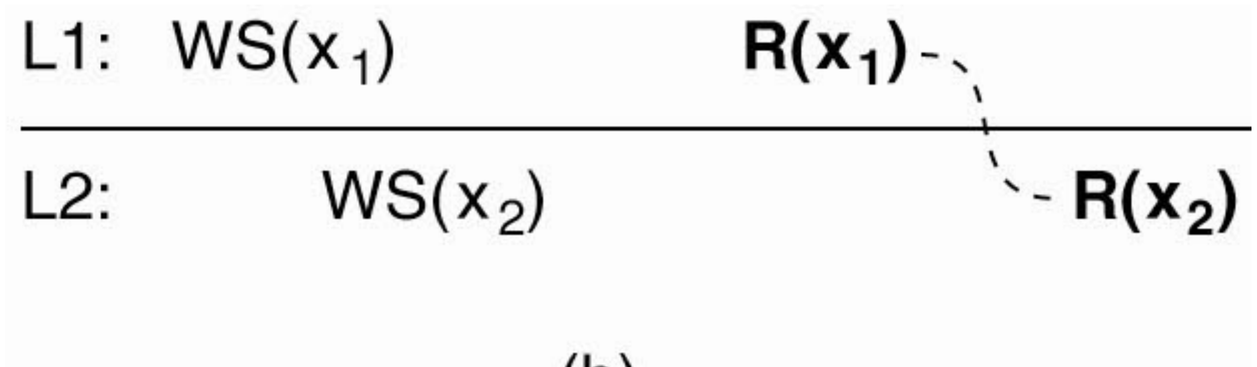
Notations

- L_i: ith local copy
- $x_i[t]$: data item x at L_i, time t
- $WS(x_i[t])$: writes operation at Li that took place since initialization
- $WS(x_i[t_1]; x_j[t_2])$: All operations $WS(x_i[t_1])$ have been delivered to L_j, before t_2

3.2. Monotonic reads

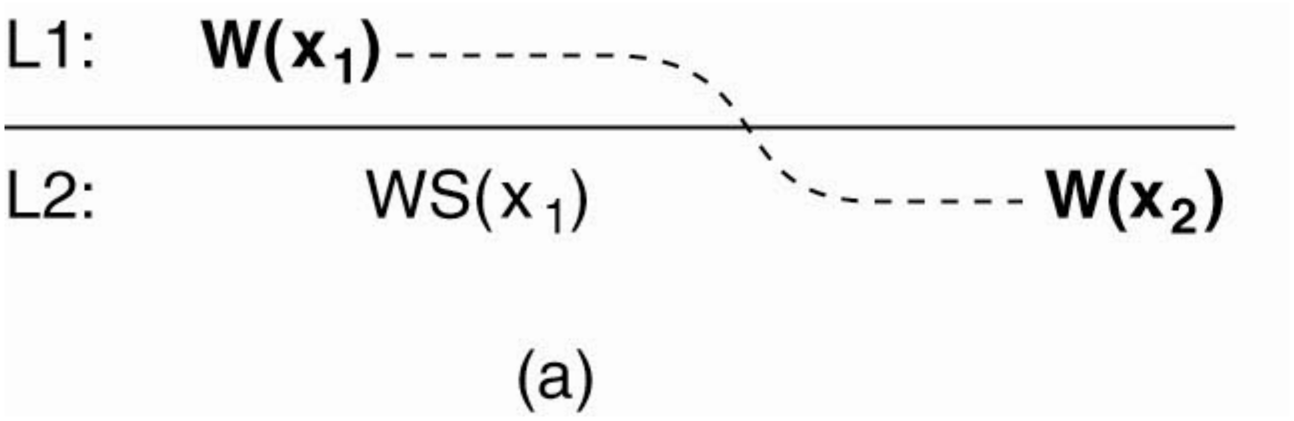


(a)

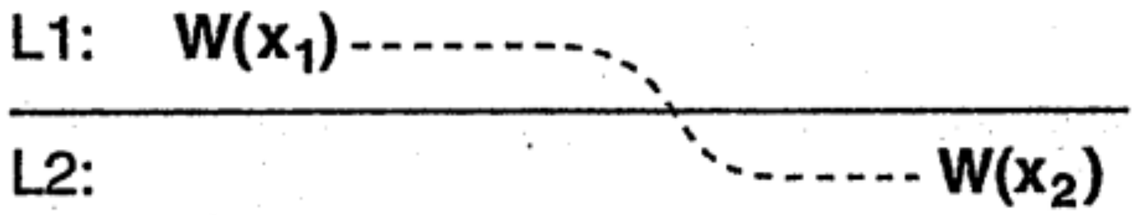


(b)

3.3. Monotonic writes

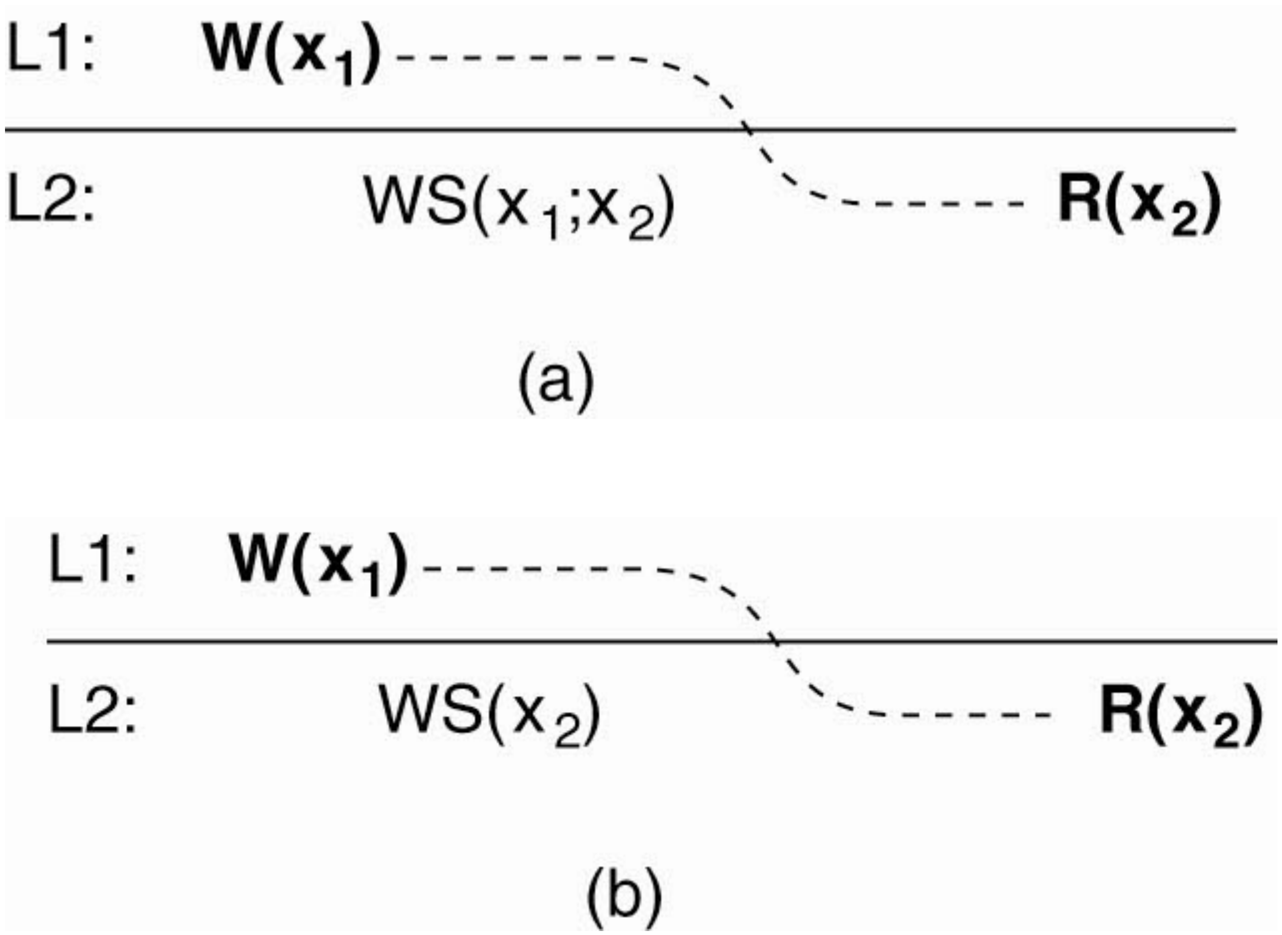


(a)

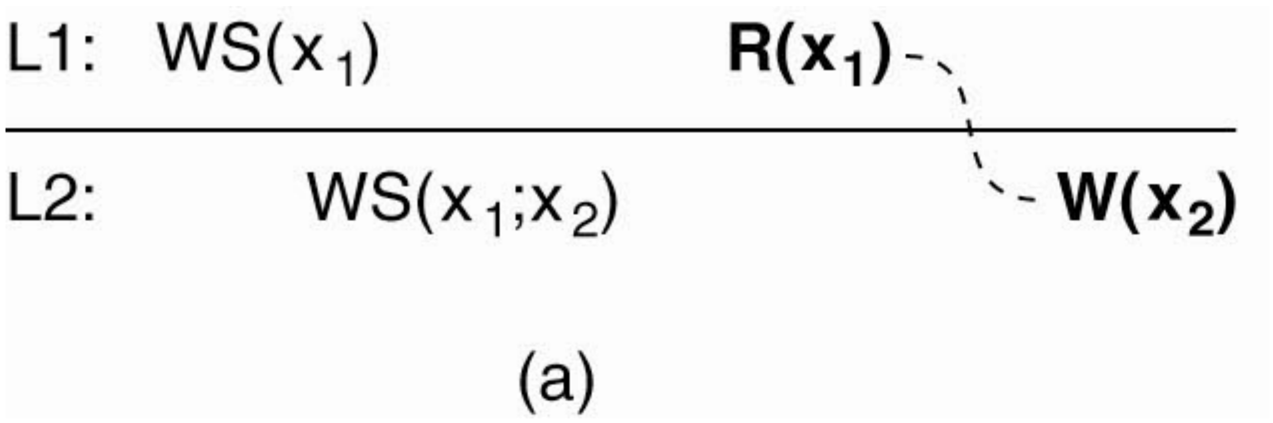


(b)

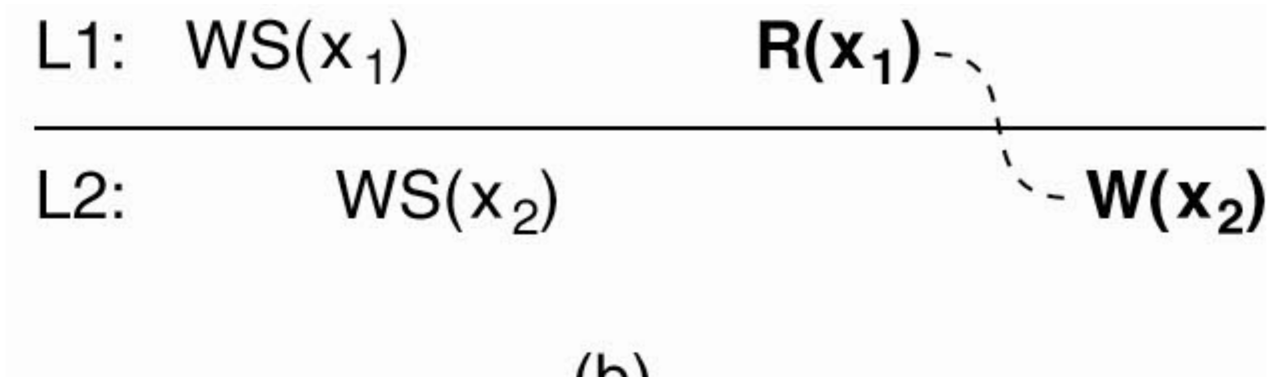
3.4. Read your writes



3.5. Writes follow reads



(a)



(b)

4. Replica management

4.1. Replica server placement

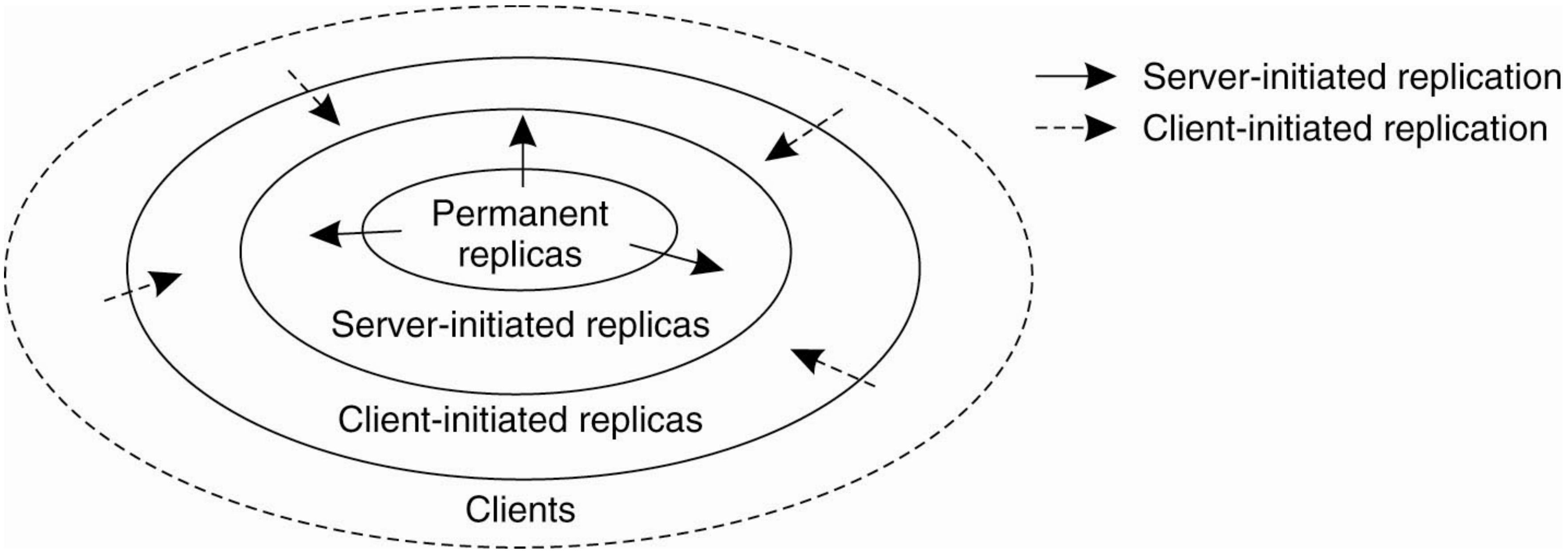
4.2. Content replication and placement

4.3. Content distribution

4.1. Replica server placement

- Problem
  - N locations for replica placement
  - Determine K out of N locations
- Solution 1
    - Distance between clients and locations
    - Select one server at a time
- Solution 2: Ignoring the position of clients
  - Take the topology of the Internet
  - Sort the ASes
    - Place the server on the router with the largest number of Network interfaces
  - Continue with the sorted list

4.2. Content replication and placement


Server-initiated replication
Client-initiated replication
Permanent replicas
Server-initiated replicas
Client-initiated replicas
Clients

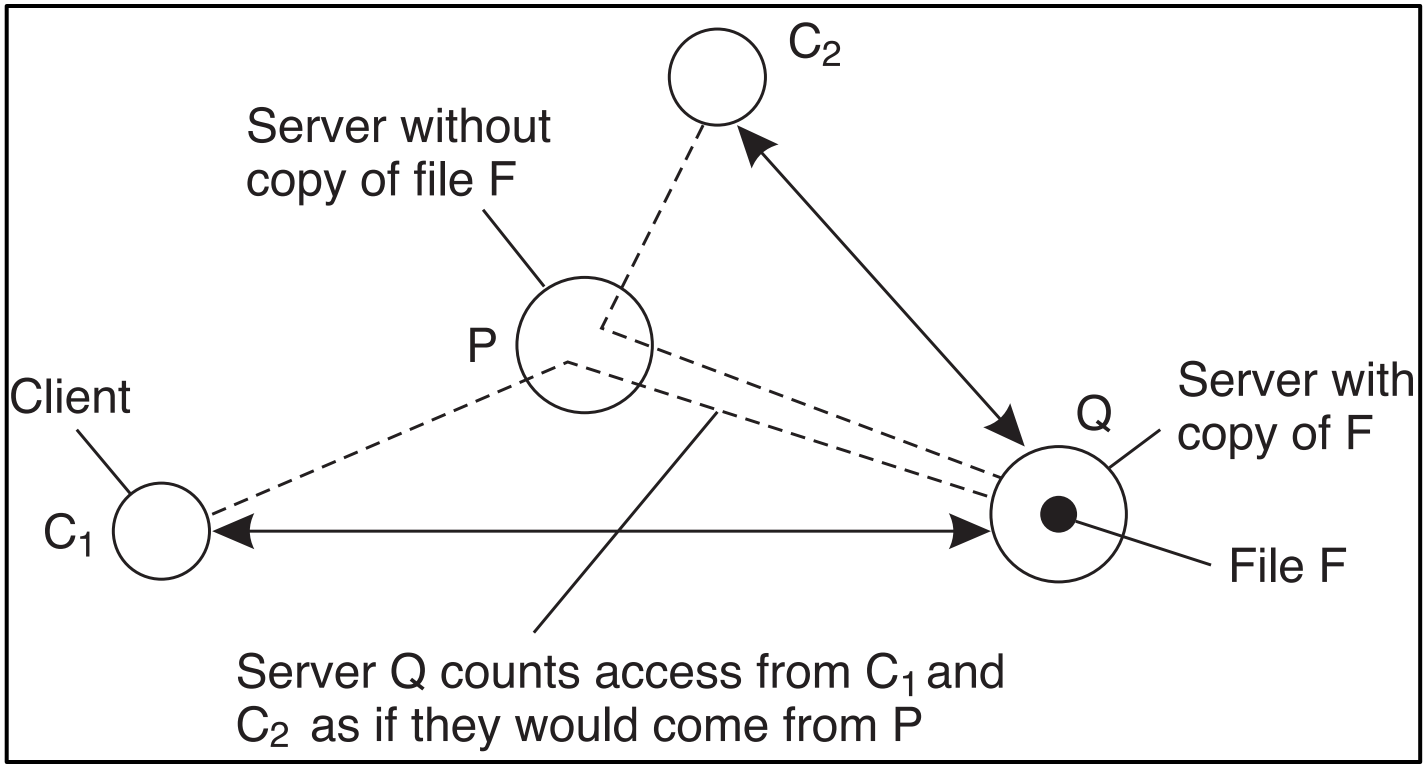
Permanent replicas

- The initial set of replicas
- The number of replica is small
- First kind of distribution
  - Data is replicated across a limited number of servers
  - For each request, it is forwarded to one of the servers (eg. using Round-robin strategy).
- 2nd kind of distribution: mirroring
  - Client simply chooses one of the various mirror sites.
- Shared-nothing architecture

Server-initiated Replicas

- Server is active
  - The number of requests increased suddenly
  - Activate other replicas
- Reduce load for replicas
- Update the data to a new replica closer to the client

Server-initiated Replicas


C_2
Server without copy of file F
P
Client
C_1
Q
Server with copy of F
File F
Server Q counts access from C_1 and C_2 as if they would come from P

Client-initiated Replicas

- Caching
  - Client manages the cache management, decides to update the cache
  - Erase
  - Write
  - Policy caching
- Can share caches between clients

4.3. Content Distribution

- State vs. Operations
- Pull vs. Push
- Unicast vs. Multicast

State vs. Operations

- Solutions for updating data:
  - Propagate only a notification of an update
    - Use little network bandwidth.
    - Read-to-write ratio is small
  - Transferring the modified data
    - Read-to-write ratio is high
  - Send update operation (active replication)

Pull/Push

- Push: server after updating notification data for all clients
  - Replica activated by server
  - Ensure high consistency
  - Weak interaction (eg when client or replica needs to update data)
  - The server should have a list of all connected clients
- Pull: client when need data will ask server
  - Usually used for client caches
  - Suitable for high writes-reads ratio
  - Increased access time (with cache miss)
- Mixed

Uni vs. multicast

- Multicasting:
  - Appropriate in case 1 replica wants to promote updates to (N-1) other copies in a data store
  - More efficient and economical than sending (N-1) times
  - Appropriate for the push-based approach
  - Not suitable if destination nodes belong to a LAN
- Unicasting:
  - Appropriate for pull-based

5. Consistency protocols

5.1. Continuous consistency
5.2. Primary-based protocols
5.3. Replicated write
5.4. Cache coherence

5.1. Continuous consistency

- Bounding numerical deviation
- Bounding staleness deviation
- Bounding ordering deviation

Bounding numerical deviation

- Single data item x.
- Each write W(x) has an associated weight that represents the numerical value by which x is updated
- The write's origin: origin(W(x))
- each server Si keeps log Li of writes that are performed on its own local copy of x.
- TW[i,j] is the writes executed by Si that originated from Sj

$$TW[i,j] = \sum \{weight(W) \mid origin(W) = S_j \; \& \; W \in L_i\}$$

  - TW[k,k] : aggregated writes submitted to Sk

Bounding numerical deviation

- Actual value of x

$$v(t) = v(0) + \sum_{k=1}^{N} TW[k,k]$$

$$v_i = v(0) + \sum_{k=1}^{N} TW[i,k]$$

$$v_i \leq v(t)$$

  - The threshold:

$$v(t) - v_i \leq \delta_i$$

Bounding Staleness Deviation

- Can use local time of processes to evaluate
  - Server S_k has vector clock RVC_k
  - if $RVC_k[i]=T(i)$ => S_k has seen all operations on S_i at T(i)
  - T(i) : local time of server *I*
  - When $T(k)-RVC_k[i]>$ delta => eliminate operations having $T>RVC_k[i]$

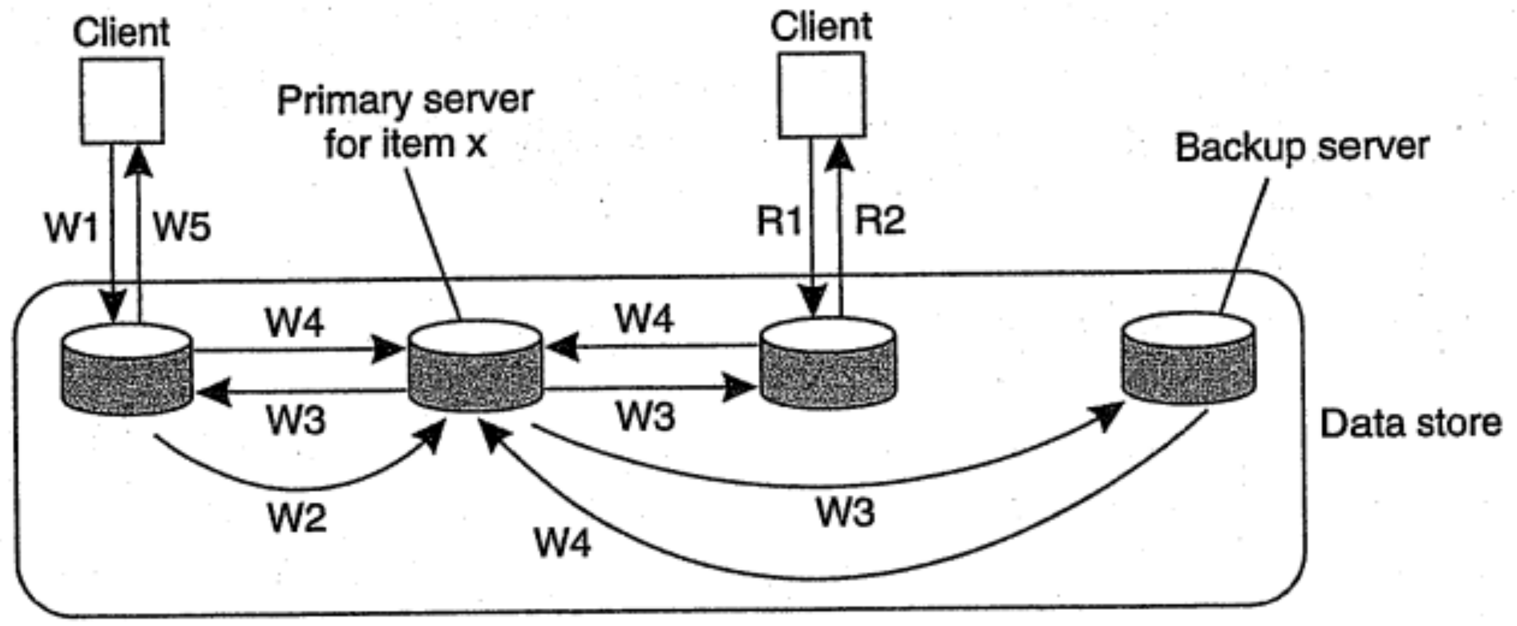
Bounding ordering deviation

- Each replica has a write queue
- Global order should be considered
- The largest number of write operations are in the queue
- When this number exceeds, the server will stop the execution and will negotiate with other servers in order

5.2. Primary-based protocols

- Consistency model => complex
- Developers need simpler models
- Each data item has a primary that is responsible for manipulating operations on that data items
- Fixed-primary (remote-write protocol)
- Local-primary (local write protocol)

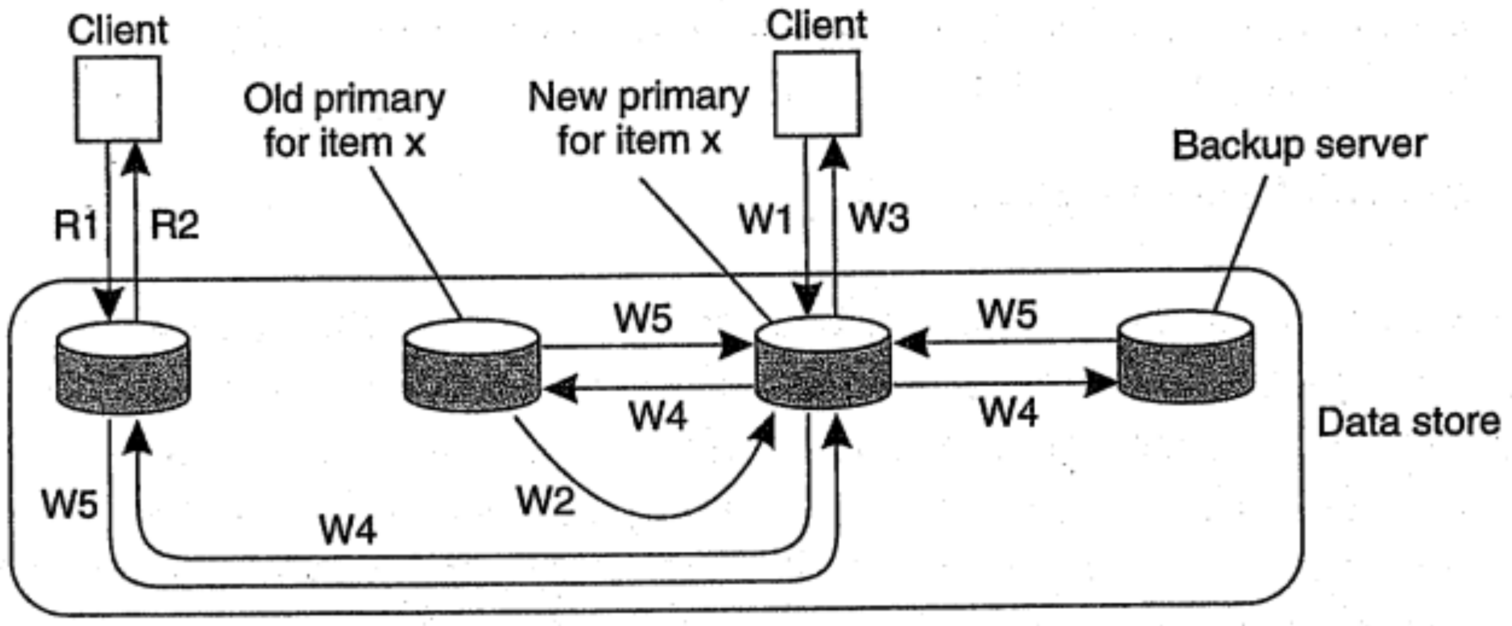
Remote-write protocol



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

Local-write protocol



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

5.3. Replicated-write protocols

1. Active replication
2. Quorum-based protocol

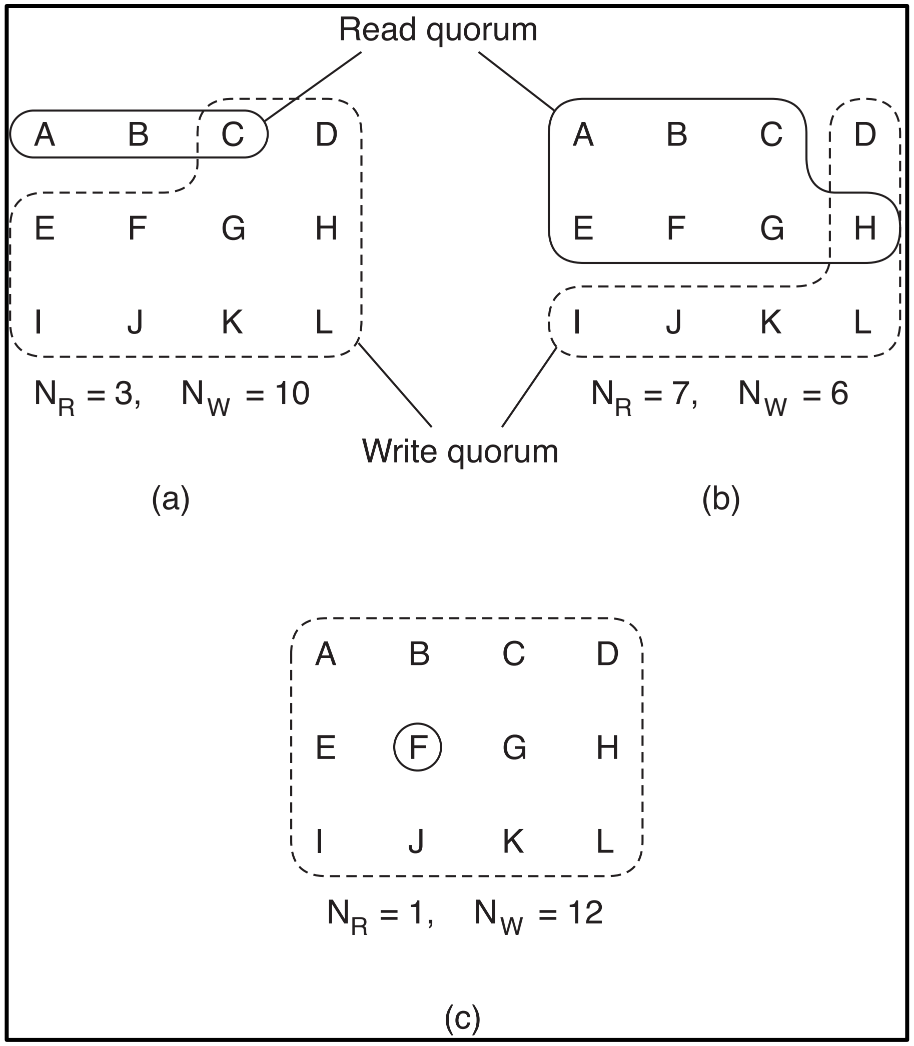
5.3.1. Active replication

- A process is responsible for propagating the update operation to all replicas
- Need a total ordered mechanism
  - logiccal synchronization of Lamport
  - Sequencer

5.3.2. Quorum-based protocol

- For strong consistency => need to update all replicas
- After updating at a costly cost => not all replicas are read => wasted
- Is there a reduction in the number of replicas that need updating?
- When reading the data
  - Risk of reading the old data
  - Read more data in some other replicas => Select the copy with the latest data
- Write Quorum & Read Quorum
  - $N_R + N_W > N$
  - $N_W > N/2$

Example of quorum


Read quorum
A B C D
E F G H
I J K L
$N_R = 3$, $N_W = 10$
A B C D
E F G H
I J K L
$N_R = 7$, $N_W = 6$
Write quorum
(a)
(b)
A B C D
E F G H
I J K L
$N_R = 1$, $N_W = 12$
(c)