Sistem Terdistribusi 10

Replication & Consistency

Masalah yang terjadi pada Sistem

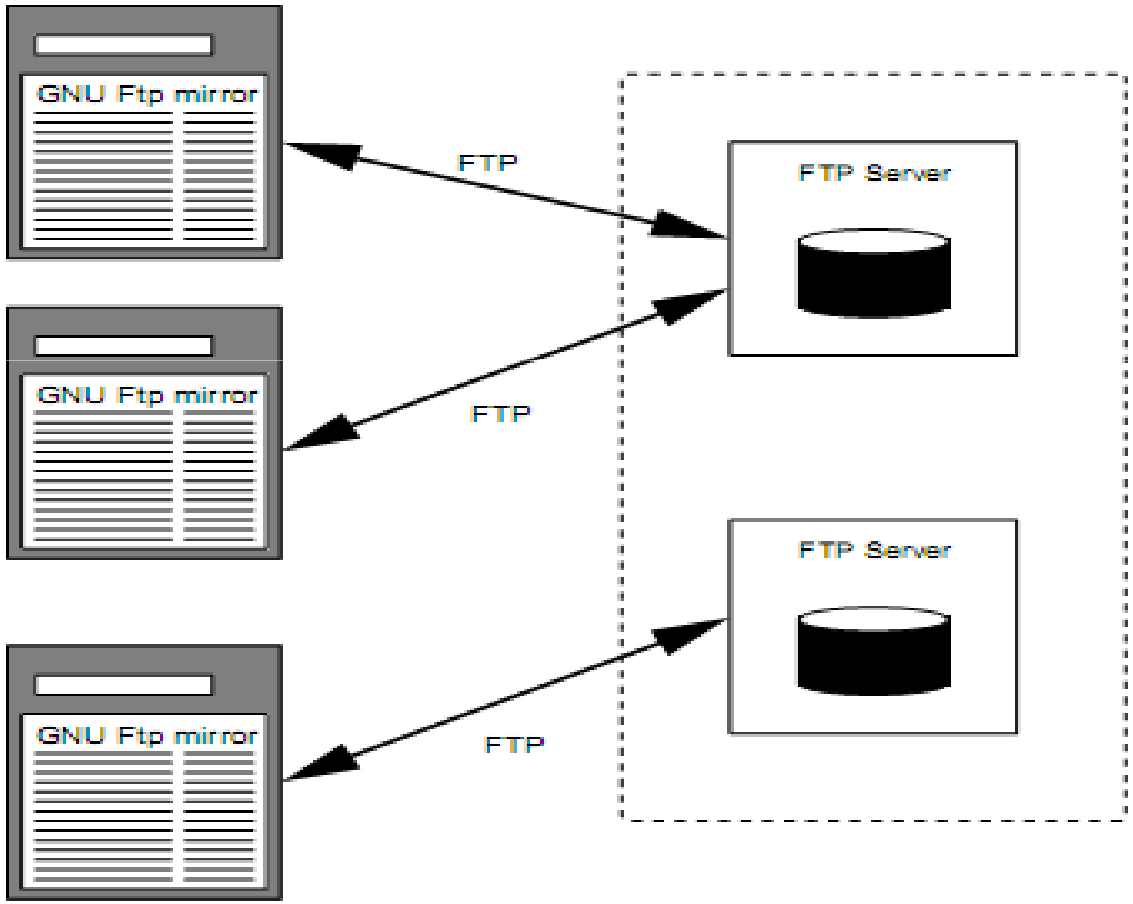
- Bagaimana agar sistem reliable?
- Bagaimana agar sistem memiliki performance yang baik?
- Bagaimana agar sistem dapat diakses dengan mudah kapan saja?

- Solusi?
  - **REPLICATION**

Replication

- Process **sharing information** to ensure **consistency** between **redundant resources** such as software/hardware components to improve **reliability, fault-tolerance,** or **accessibility**
- Make **copies** of services on **multiple machines**

Server Replication



Why Replication?

- **Reliability**: dapat diandalkan
- **Performance**: performanya tetap baik
- **Scalability**: dapat diperluas
- **Availability**: tetap dapat diakses

Reasons detail

- **Reliability**:
  - If a replica **crashes**, system can **continue working** by switching to other replicas.
  - Avoid corrupted data:
    - can protect against a single, failing write operation.
- **Improving Performance**
  - Important for distributed systems over large geographical areas.
  - **Divide** the work over a number of servers.
  - Place data in the proximity of clients.

Reasons detail

- **Scalability**:
  - Data dapat ditambahkan hingga besar
  - Walau data besar, namun data tetap dapat ditampung karena data dapat dipecah dan direplikasi
  - Sistem dapat **diperluas**
- **Availability**:
  - Karena data tersebar dengan replikasi, maka data akan **selalu ada** jika diakses
  - Masing-masing replika dapat saling menggantikan jika terjadi kerusakan

Example: DNS

- **DNS (Domain Name Service)** allows owner of a domain to **replicate** name database

- Same **two** reasons:
  - Data dapat dibagi-bagi agar lebih dekat ke client
  - Sistem memiliki backup data sehingga dapat diandalkan

- Also need
  - Scaling technique

Issue-issue

- **Updates**
  - Consistency?
    - Whenever a copy is modified, it becomes different from the rest.
  - Sinkronisasi dan Locking?
- **Replica placement**
  - How many?
  - Where?
- **Redirection / Routing**
  - Which replica should be used by client?

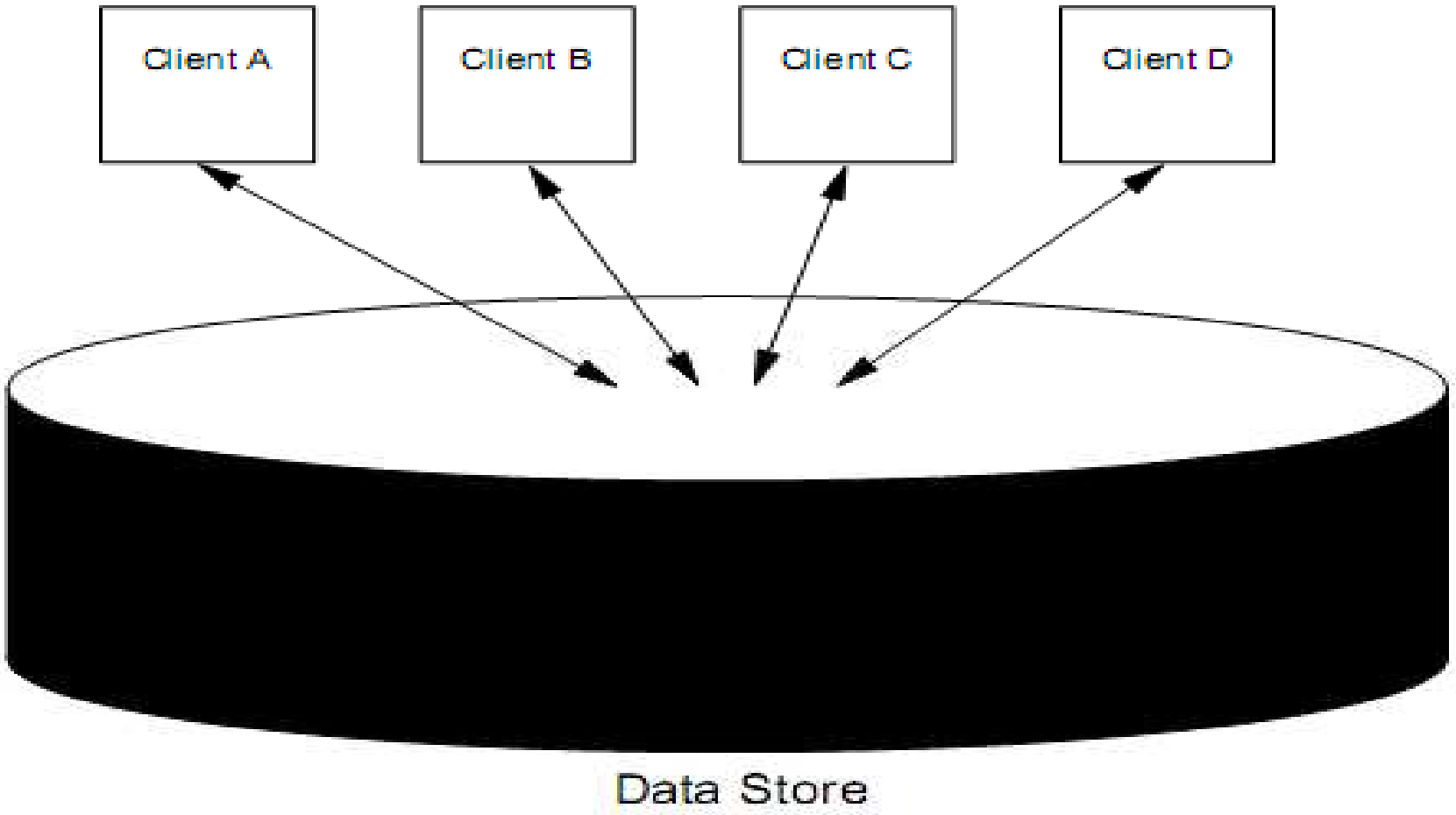
Replication

- **Data centric**
  - Focus on data in all replicas
  - Try to keep data consistent across replicas
- **Client centric**
  - Focus on single client
  - Only maintain consistency for each client separately

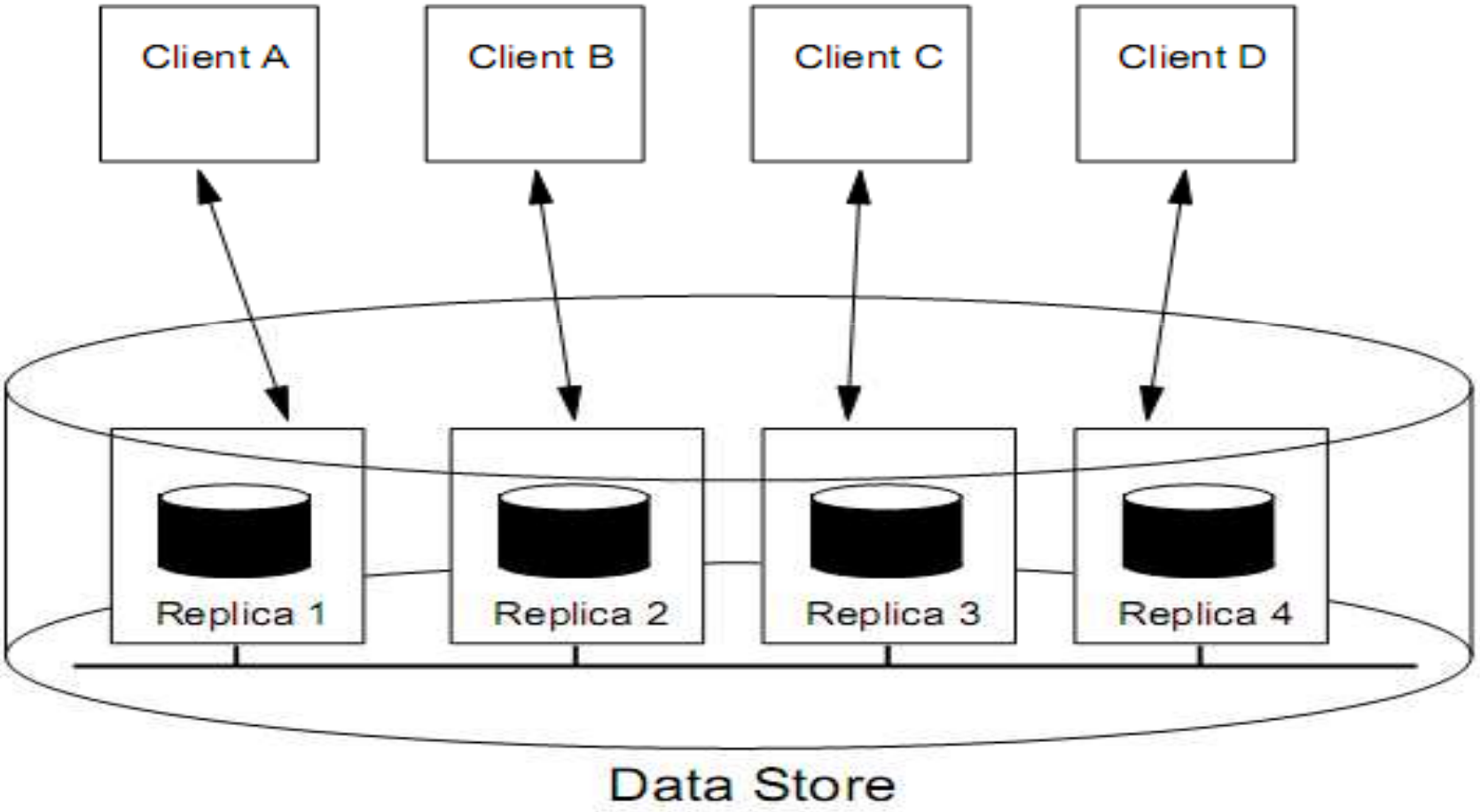
General Approach

- Update a **single item** in local replica
  - Atomically
  - Time-stamp – e.g., logical clock
- Replica **propagates** update to all of its other replicas
  - Periodically (polling)
- Receiving replica ***merges*** update with its own copy
  - Conflicting updates resolved arbitrarily to **latest time-stamp**

Model



The truth


Client A
Client B
Client C
Client D
Replica 1
Replica 2
Replica 3
Replica 4
Data Store

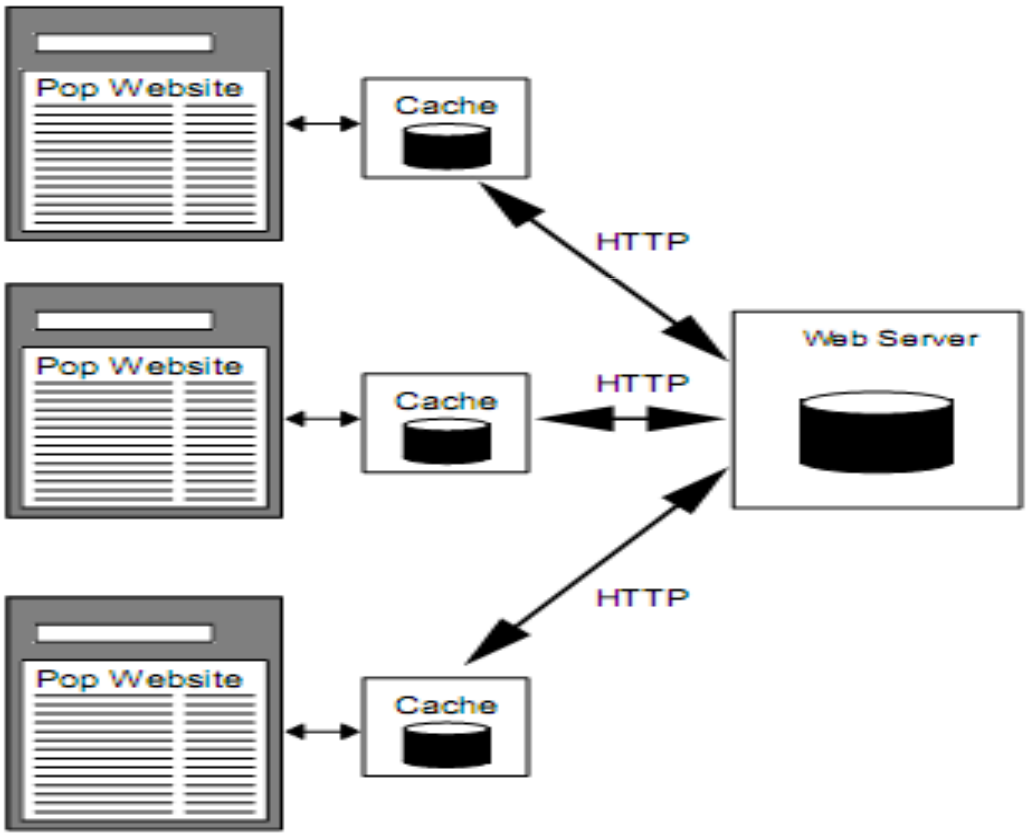
Rules

- **Why replicate?**
  - **Reliability**
    - Avoid single points of failure
  - **Performance**
    - Scalability in numbers and geographic area
- **Why not replicate?**
  - Replication transparency
    - **Consistency issues**
  - Updates are **costly**
    - Availability may suffer if not careful

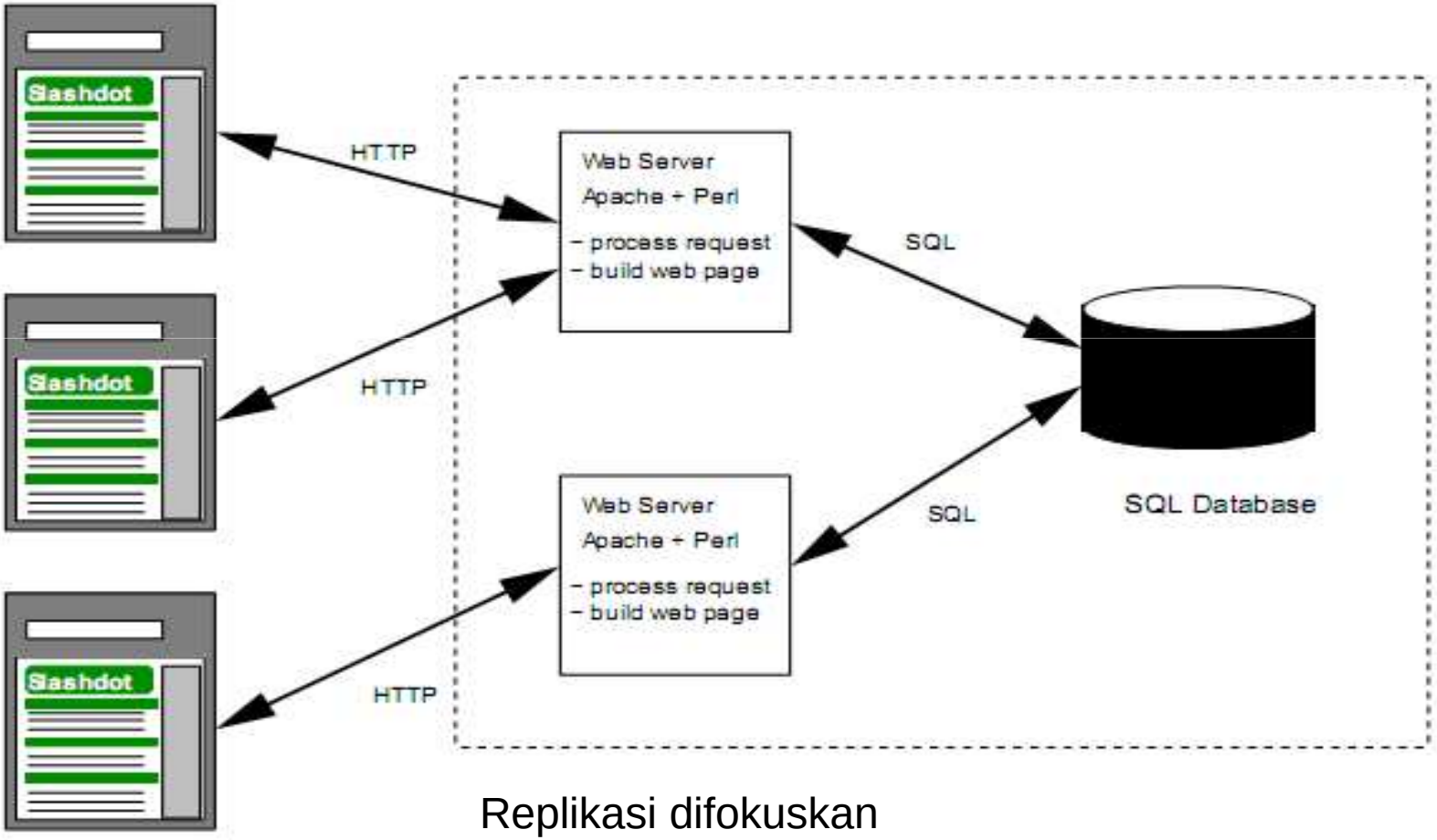
Caching

Data Replication (Caching):

Penyimpanan data2 yang sering diakses di tempat penyimpanan sementara

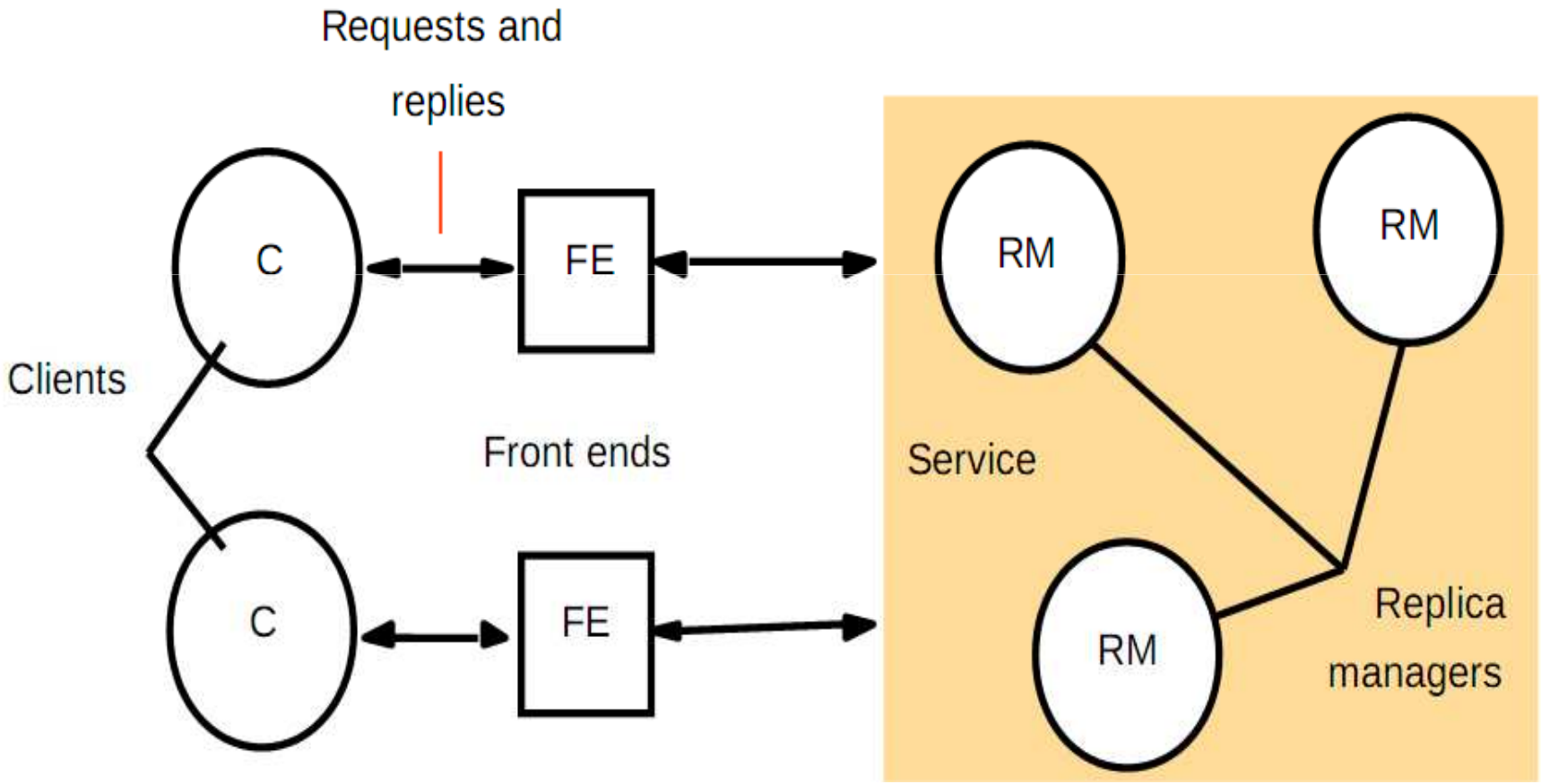


Control Replication

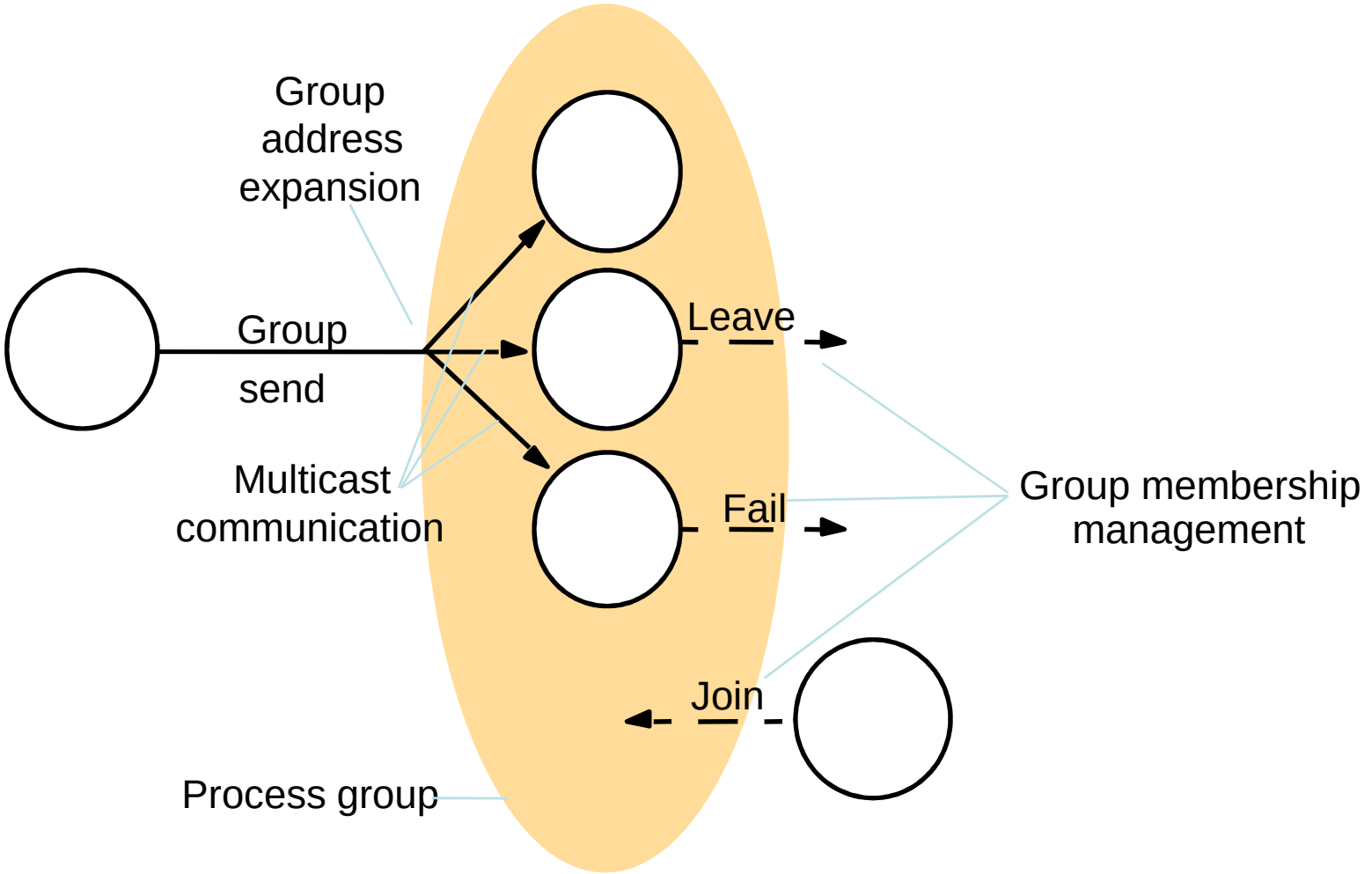


Replikasi difokuskan
pada software server

Arsitektur Replikasi

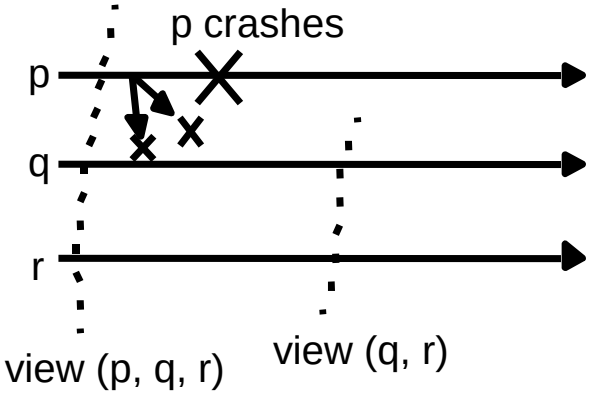


Services provided for process groups

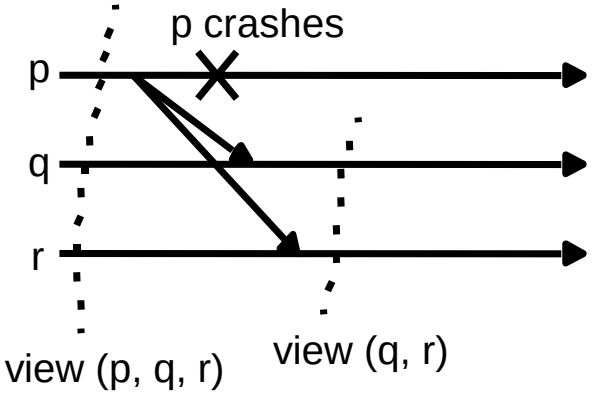


View-synchronous group communication

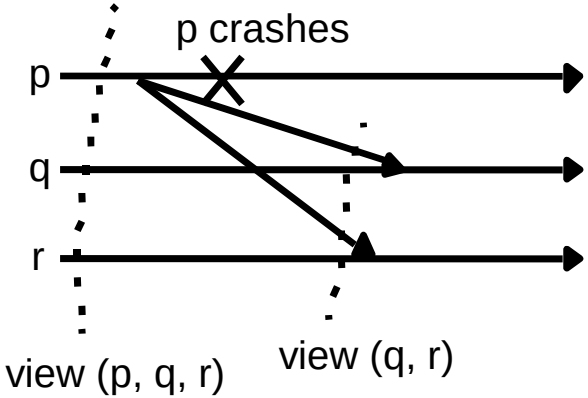
a (allowed).



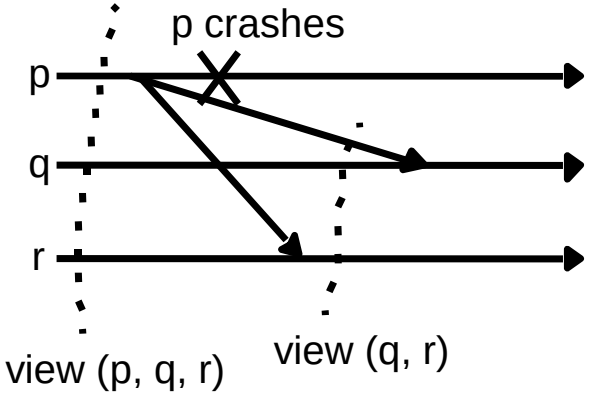
b (allowed).



c (disallowed).



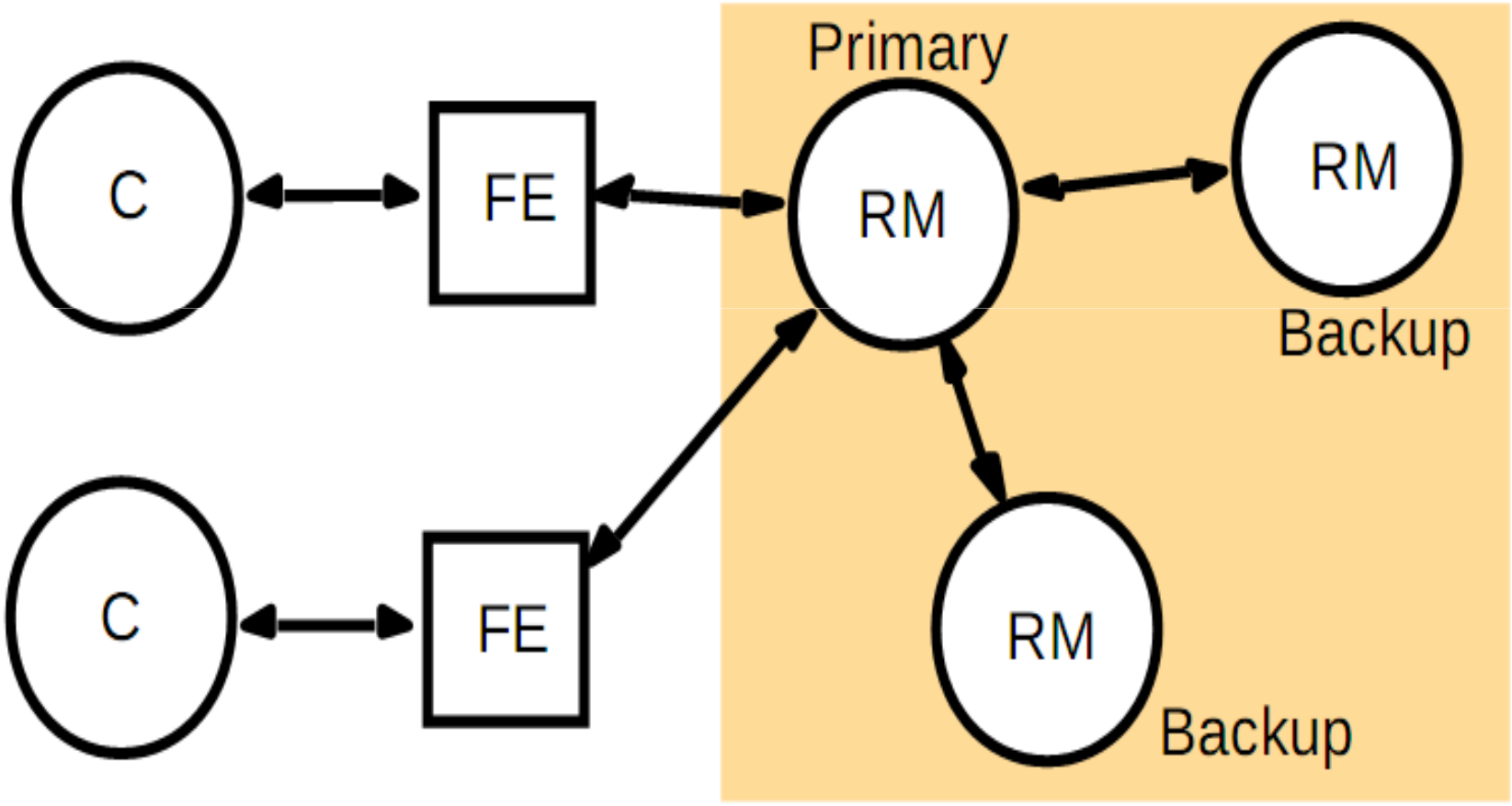
d (disallowed).



Replication Management

- **Front End**: request communication
  - Requests can be made to a single RM or to multiple RMs
- **Coordination**: RMs decide
  - whether the request is to be applied
  - the order of requests
- **Execution**: The RMs execute the request tentatively.
- **Agreement**: The RMs attempt to reach consensus on the effect of the request.
- **Response**
  - One or more RMs responds to the front end.

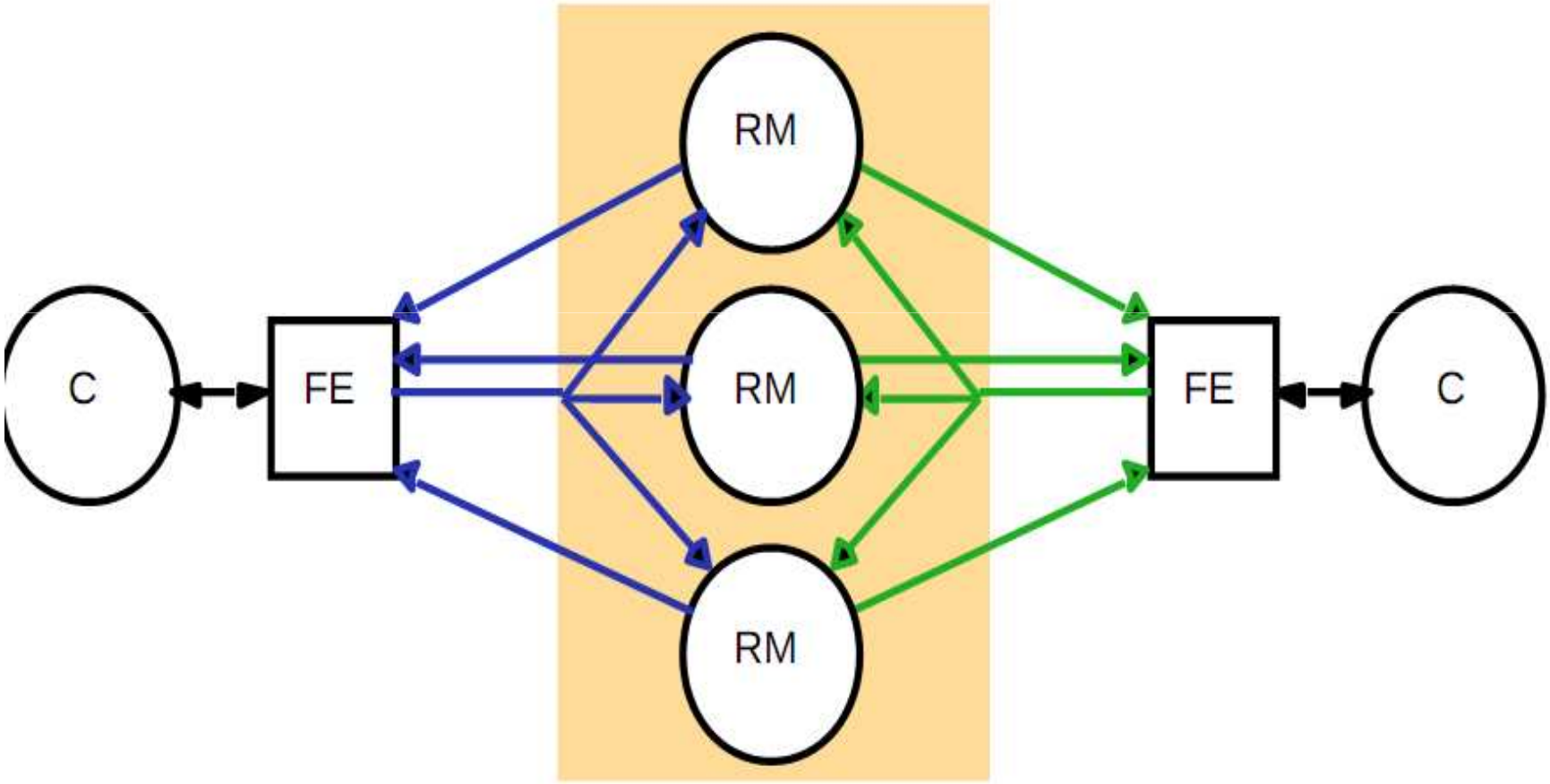
Passive Replication



Passive Replication

- If primary server is **down**, pick *one backup to be primary*
- **Disadvantage**: big overhead (primary must **wait** until all data is propagated to backups)
- **Variant**: FE sent all read request directly to backups
- **Example**: Sun Network Information System (NIS)

Active Replication

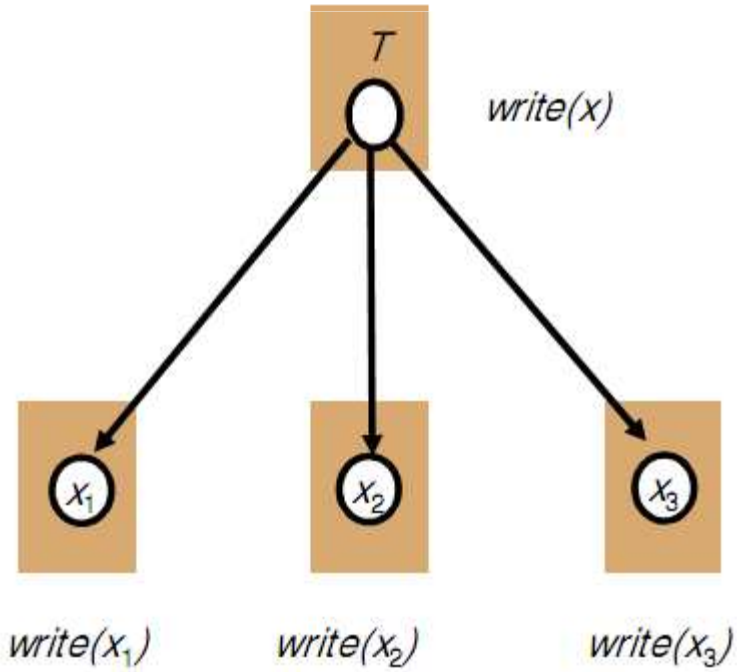


Active Replication

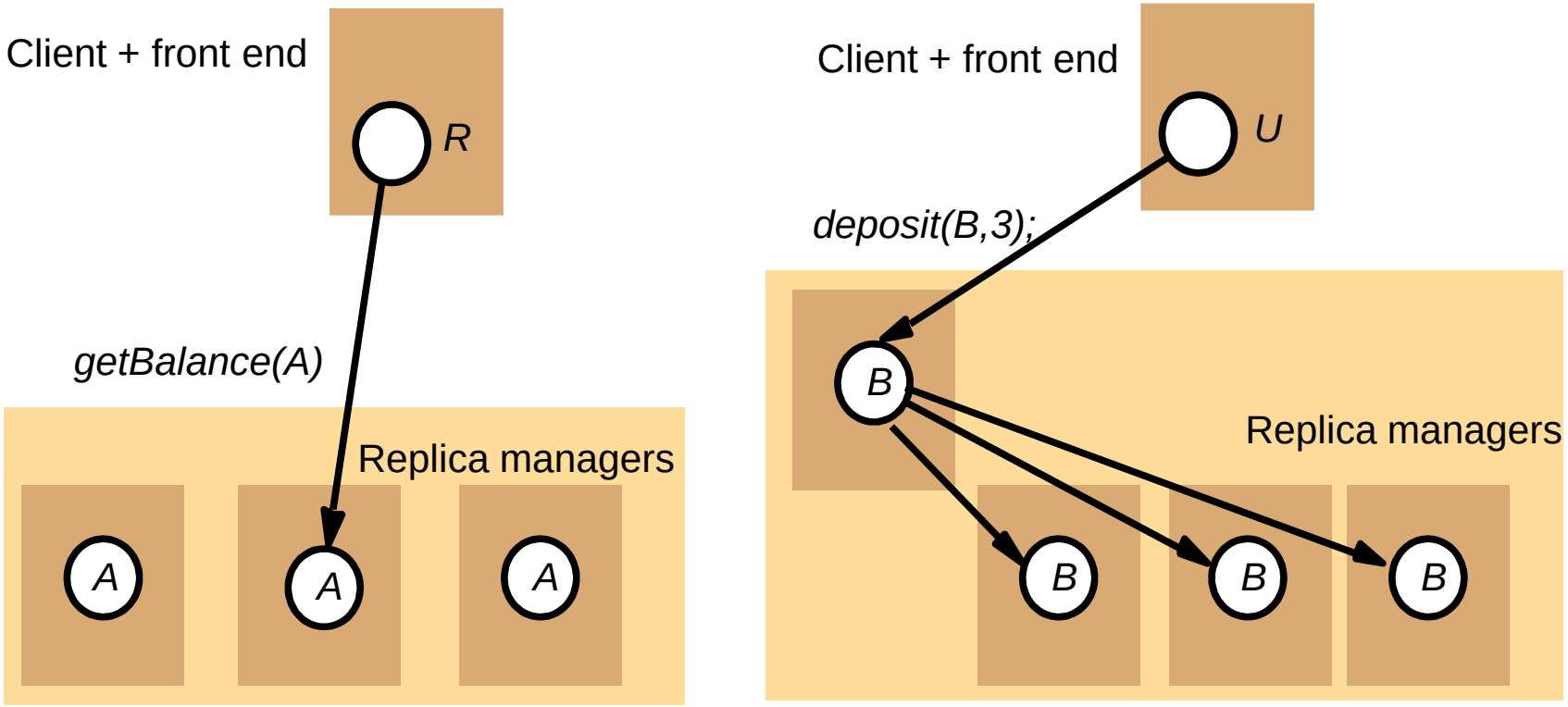
- **Request**
  - FE send **multicast** request to RM
  - **Read** access only to **one** RM
  - **Write** request goes to **all** RM in sequential orders
- **Coordination**
  - Group communication system **send request** to **each** RM

Physical & Logical Object

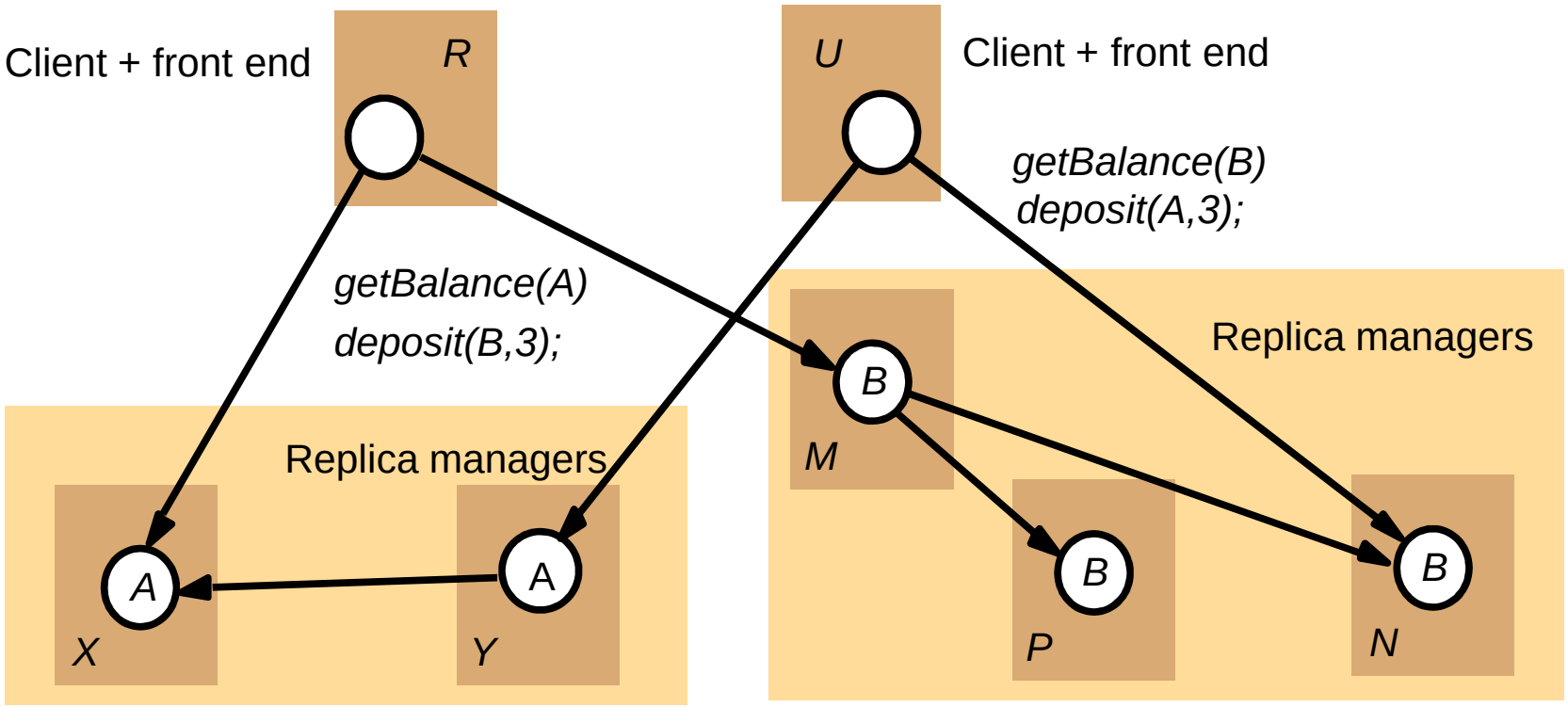
- There are **physical** copies of **logical** objects in the system.
- Operations are specified on **logical** objects, but **translated** to operate on **physical** objects.



Transactions on replicated data



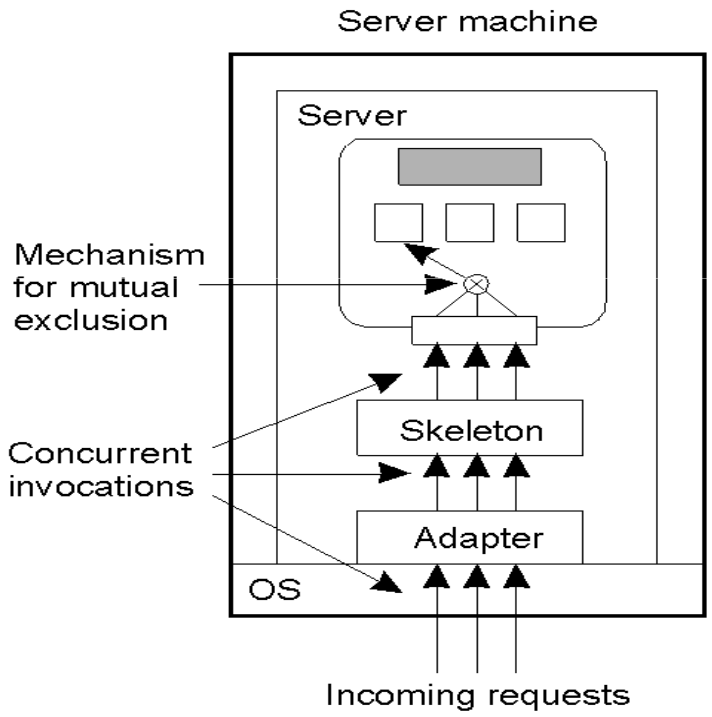
vailable copies

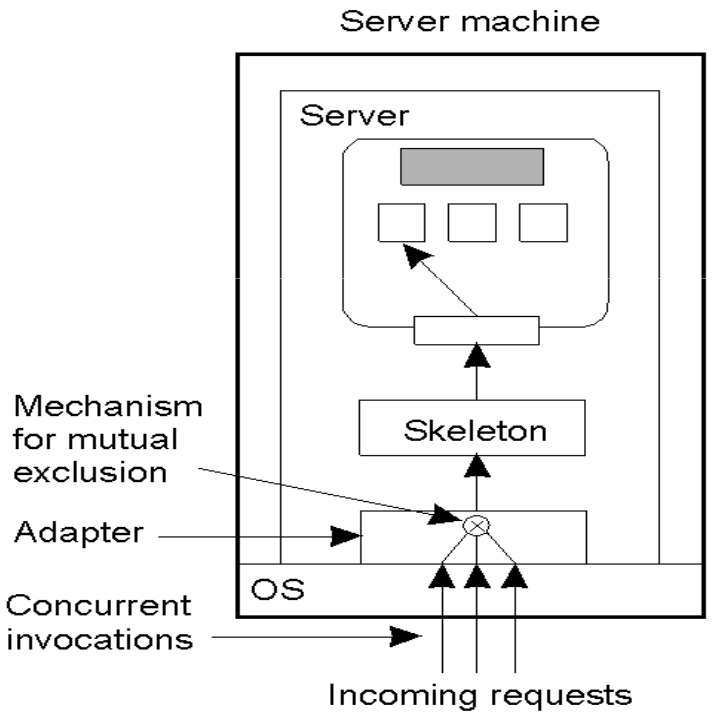


Object Replication

a) A remote object capable of handling concurrent invocations on its own.

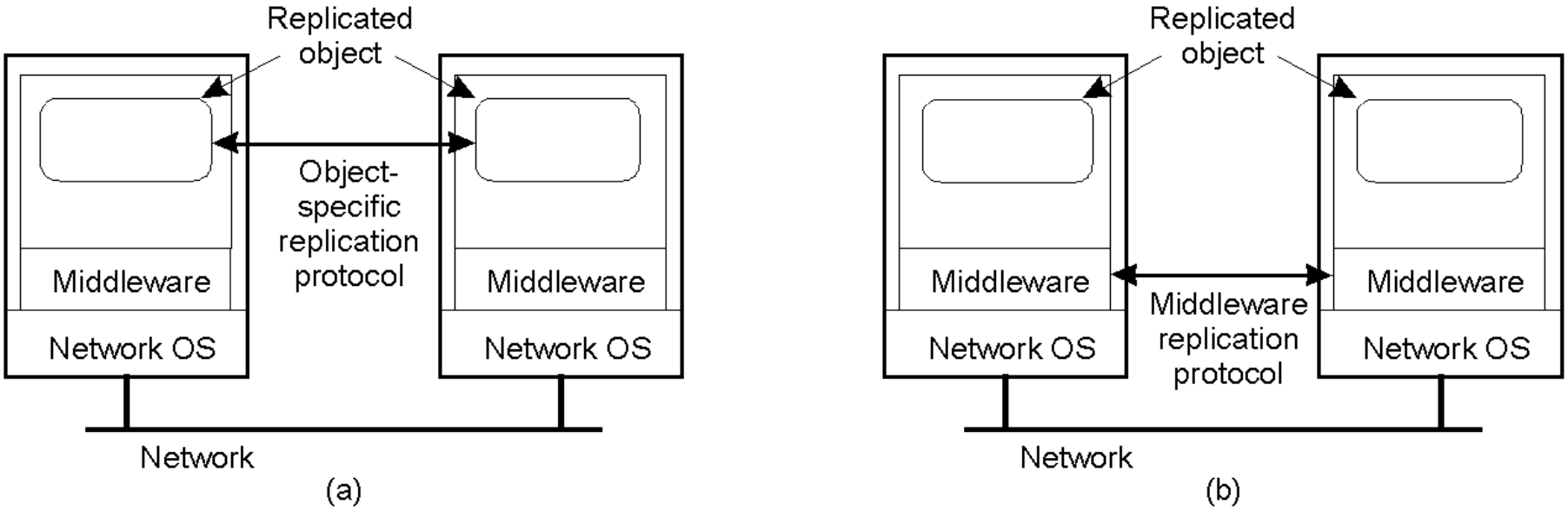
b) A remote object for which an object adapter is required to handle concurrent invocations





Concurrency-aware objects vs system-managed concurrency

Object Replication



Replication-aware objects can adopt object-specific policies

a) Replication-aware distributed objects.

b) A distributed system responsible for replica management

Operation on data store

- **Read**. Ri(x)b -> Client i performs a read for data item x and it returns b
- **Write**. Wi(x)a -> Client i performs write on data item x setting it to a
- Operations depends on:
  - Time of issue (when request is sent by client)
  - Time of execution (when request is executed at a replica)
  - Time of completion (when reply is received by client)

Inconsistency

- Data:
  - How old is the data?
  - How old is the data allowed to be?
  - Time modified
  - Versions
- Operation order:
  - Were operations performed in the right order?
  - What orderings are allowed?

Consistency

- Clients **can modify** resource on any of the replicas.
- What happens if another client requests resource before replica has informed others of modification, as in cache consistency in distributed file systems?
  - Answer depends upon application...

Consistency

- Non-distributed data store:
  - Program order is maintained
- Updates and concurrency may result in **conflicting** operations
- Conflicting Operations:
  - **Read-write** conflict (only 1 write)
  - **Write-write** conflict (multiple concurrent writes)
- Consistency:
  - The **order** in which conflicting operations are performed affects consistency

Contoh

Client A: `x = 1; x = 0;`

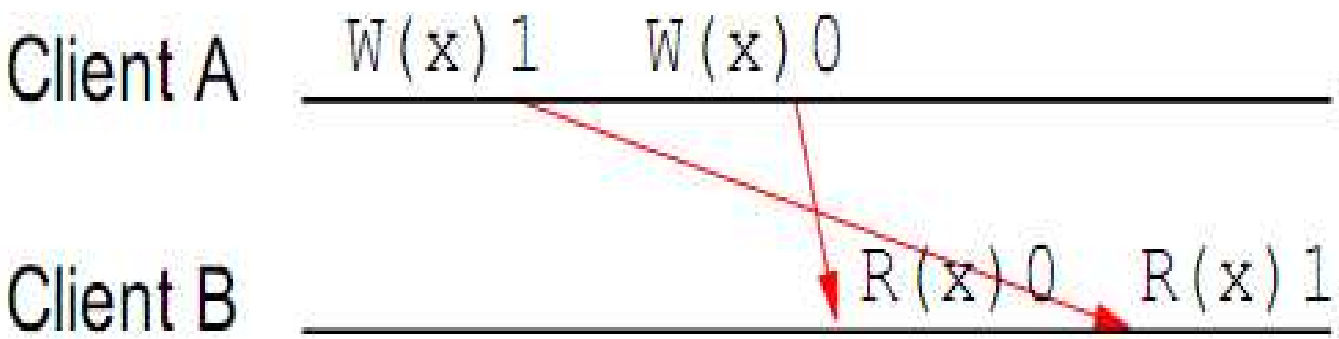
Client B: `print(x);`

`print(x);`

Possible results:

--, 11, 10, 00

How about 01?



Consider three processes

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

- Three shared variables: *x, y, z*
  - Initialized to **zero**
- Each updates one variable, and then reads and prints other two

Four Valid Execution Sequences

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

Prints : 001011

Signature : 001011

(a)

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

Prints: 101011

Signature: 101011

(b)

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

Prints: 010111

Signature: 110101

(c)

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

Prints: 111111

Signature: 111111

(d)

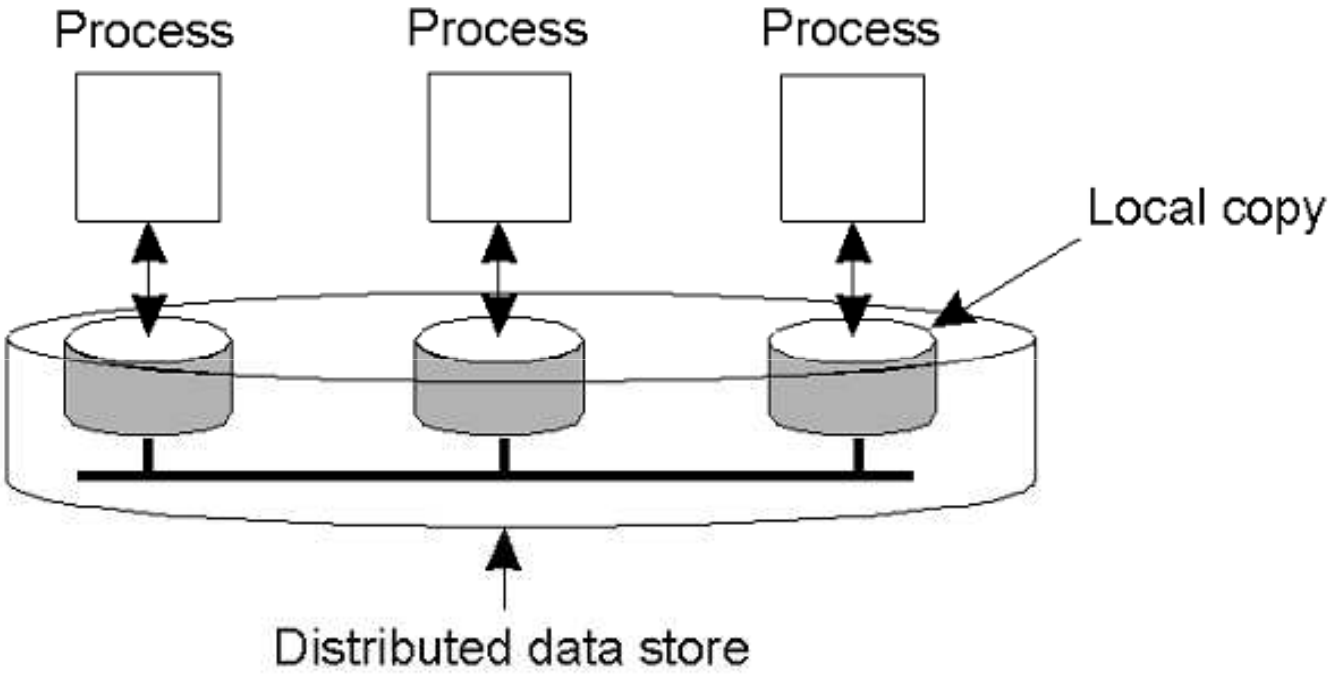
Coherence vs Cosistency

- Data **Coherence**: ordering of operations for **single data item**
  - e.g. a read of x will return the **most recently** written value
- Data **Consistency**: ordering of operations for **whole data store**
  - includes ordering of operations on **other data** items too

Consistency Model

- **Data** centric
  - Strict consistency
  - Sequential consistency
  - Release consistency
  - Lazy release consistency
- **Client** centric
  - Monotonic reads
  - Monotonic writes
  - Read your writes
  - Write follows read

Data centric consistency model



The general organization of a logical data store, physically distributed and replicated across multiple machines.

Data centric

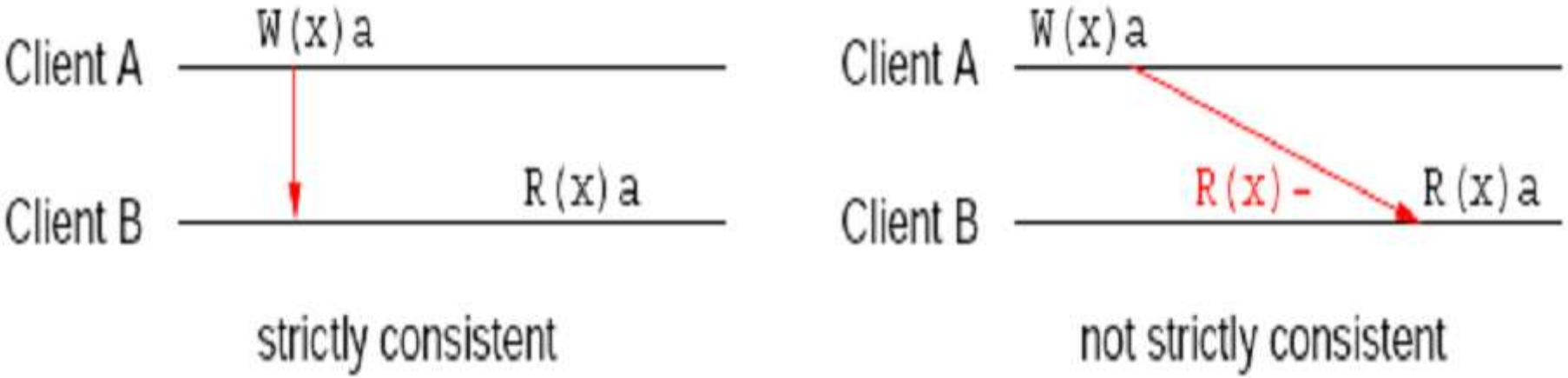
- Contract between processes and the data store. If processes obey certain rules, data store will work correctly
- Normally one would like: "*read returns the result of most recent write*"
- **However**: No global clock! What is most recent (last) write?
- **Conflict**: Two operations in the same interval on the same data item and at least one is a write.

Simbol

- **W(x) a**, berarti client tertentu menulis sesuatu bernilai a
  - Ex: x = a
- Tanda panah merah, berarti client tertentu menulis di tempat client lain, sehingga bisa menyebabkan ketidak konsisten an
- **R(x) a**, berarti client tertentu membaca nilai variabel a
  - Ex: print(x) -> a

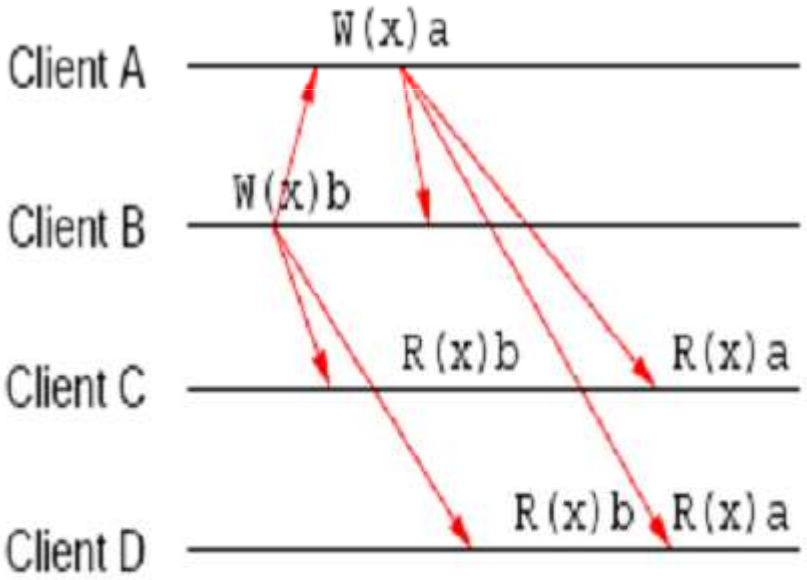
Strict consistency

- Any **read** on data item x returns a value corresponding to the results of the **most recent write** on x
- Implicitly assumes the presence of a global clock
- A write is immediately visible to all processes
- Hard to implement on distributed system because of "**most recent**" due to network delays and no global clock

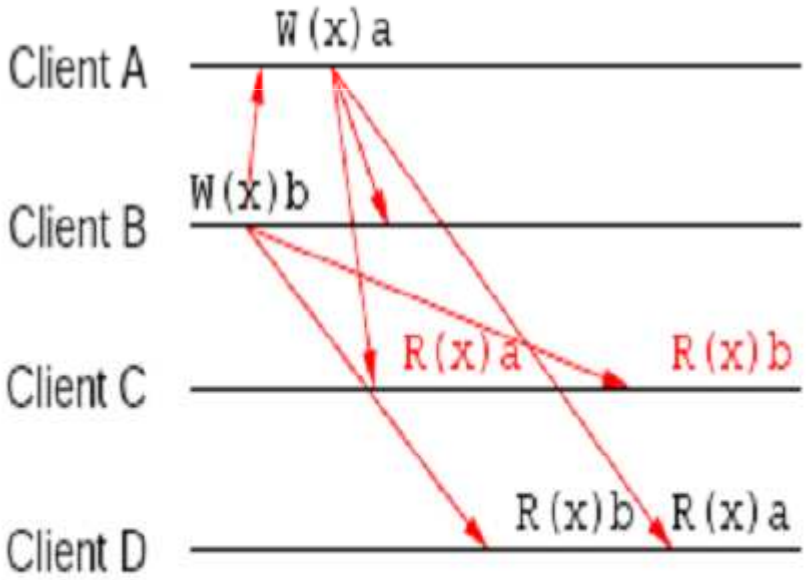


Sequential Consistency

- All **write** operation are done **sequentially**
- Not ordered according to “time”



sequential

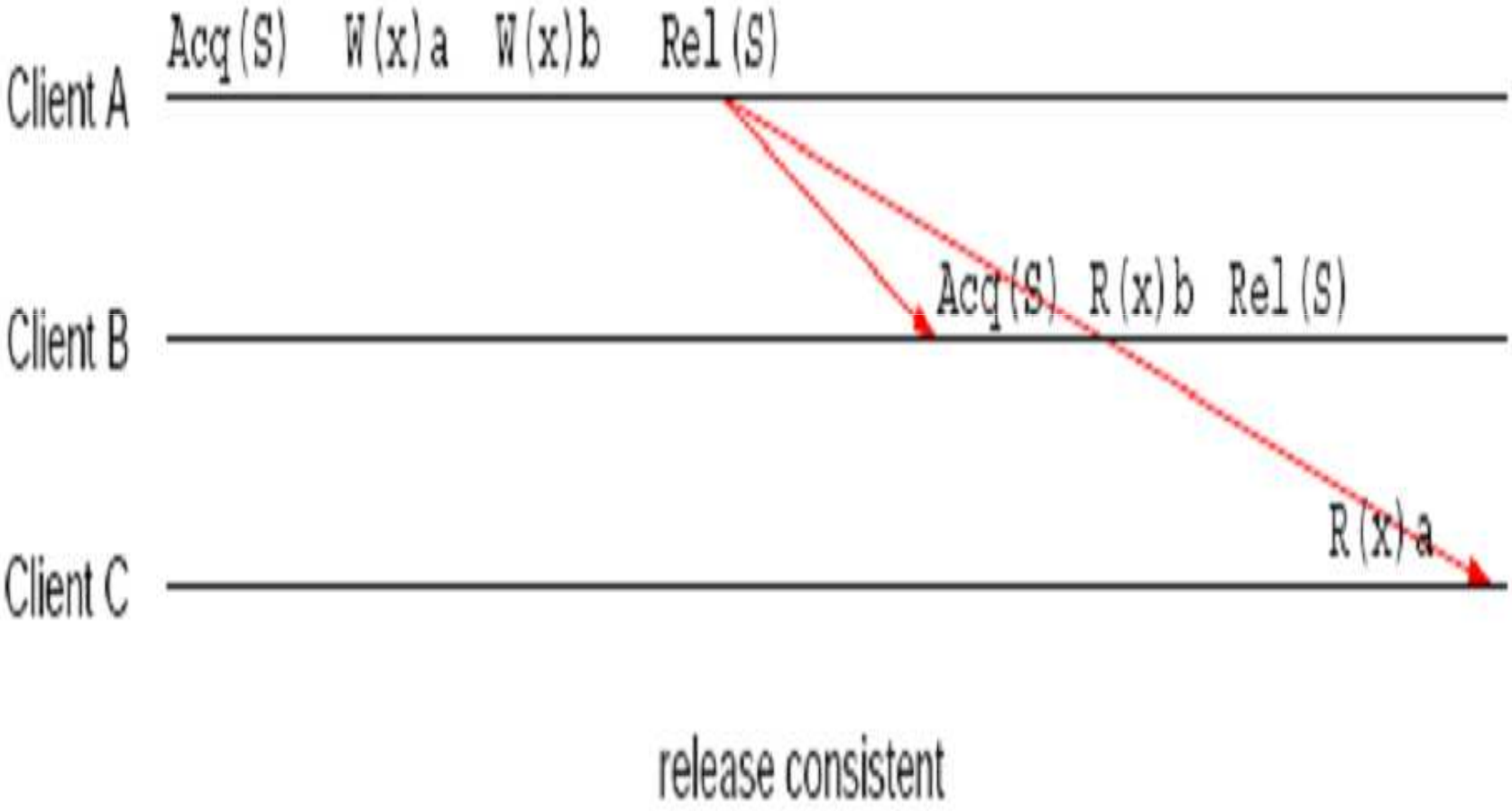


not sequential

Release consistency

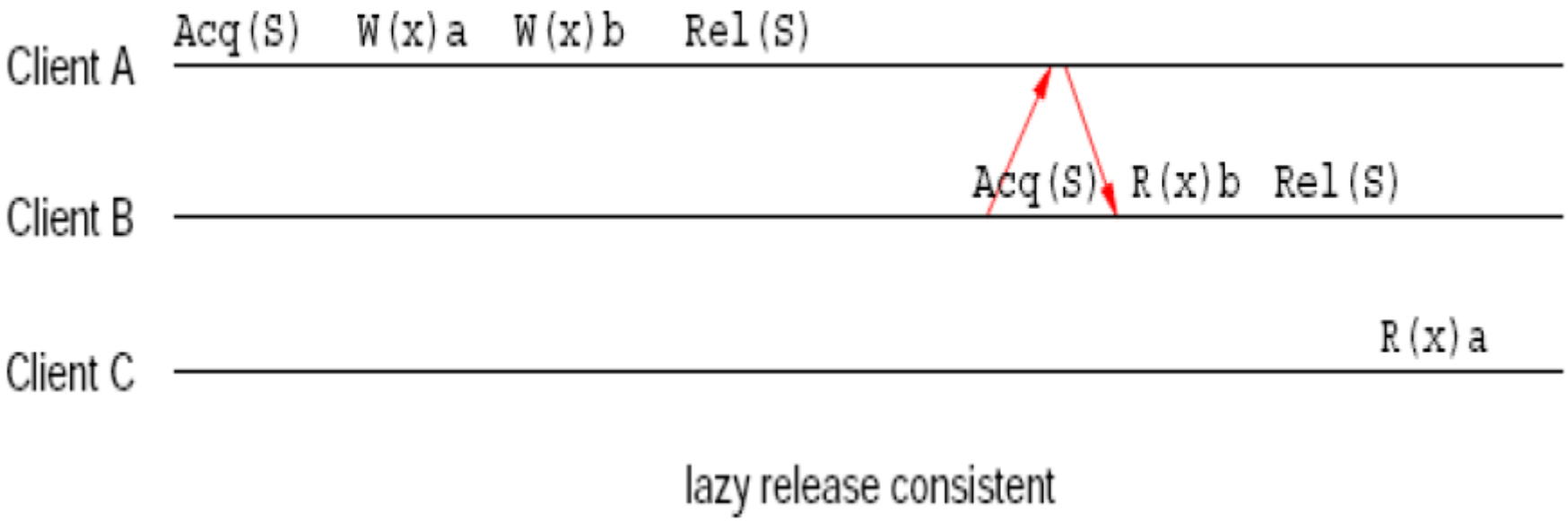
- Explicit separation of synchronization task
  - **Acquire** -> proses pengaksesan data ter-up-to-date
  - **Release** -> proses melepas semua data yg dipegangnya
- Orders are **FIFO**
- **Release** only after all read/write by client is completed
- Read/write only after all **acquire** by client is completed

Eager Release consistency



Lazy Release Consistency

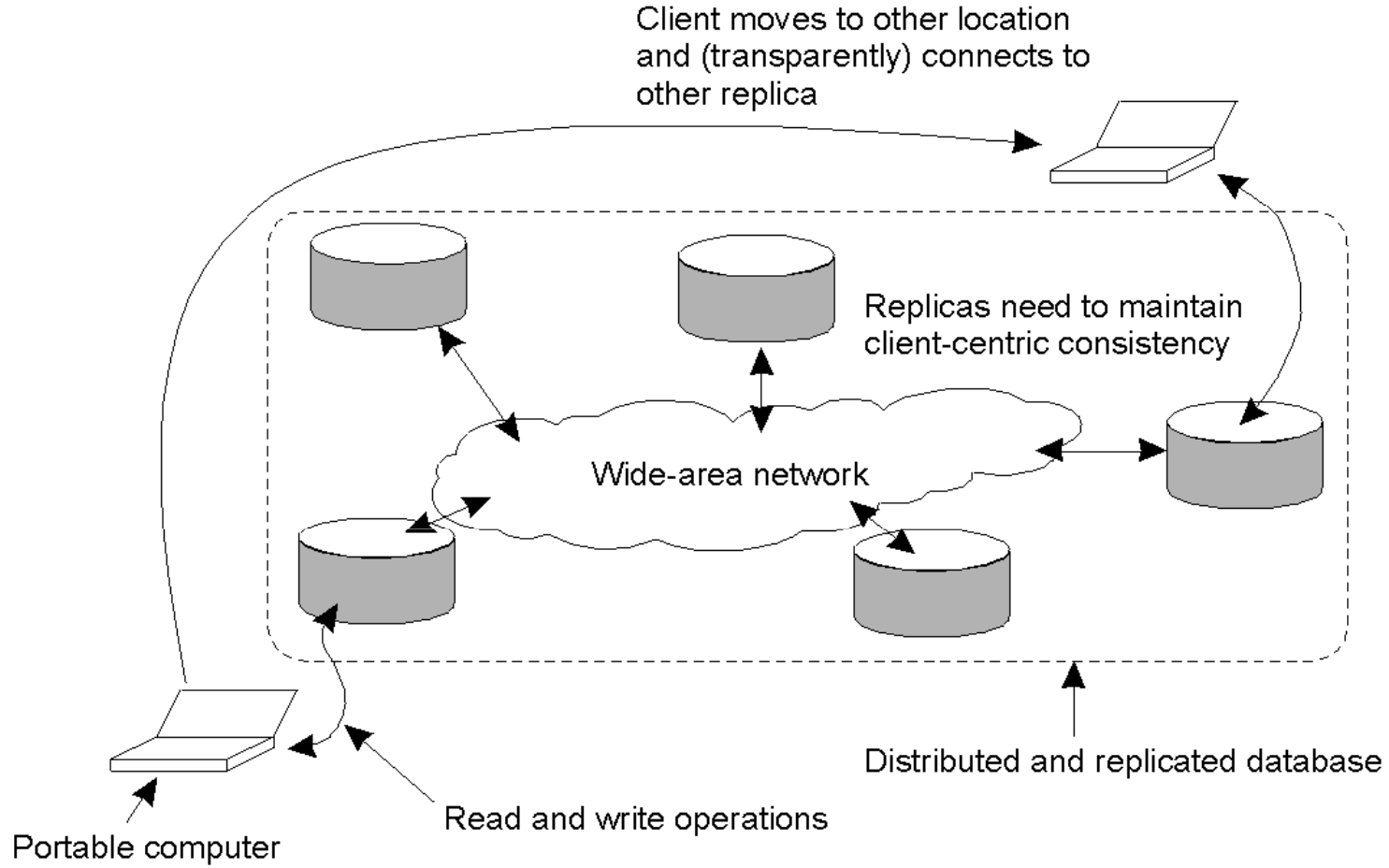
- **Don't** send updates on release
- **Acquire** cause clients get newest state
- Done by client -> **more efficient**



Client Centric

- Provides **guarantees** about ordering of operation for **single** client
- **Single client** access data store
- Client accesses **different** replicas
- Data **isn't shared** by clients
- Each client will see **different** orderings
- Effects on operation **depends on** the **clients** itself and also from historical operations the **client** has performed

Client Centric : Mobile Users

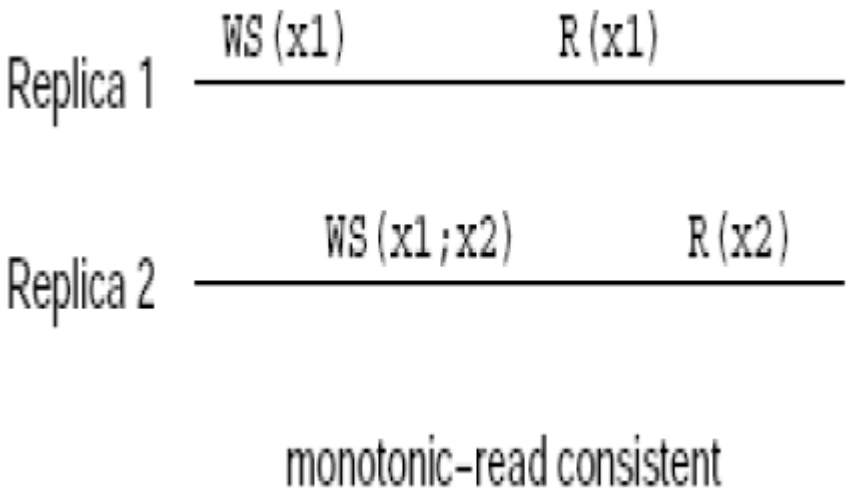


Mobile users present a challenge

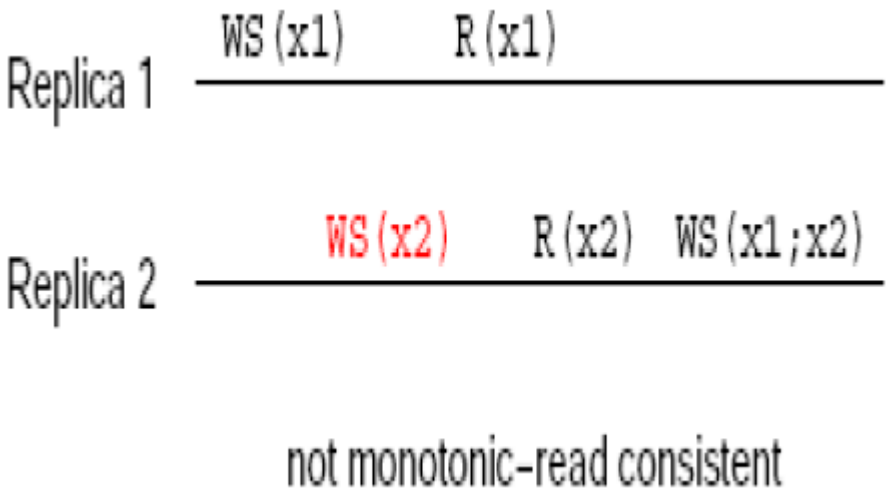
- Client may access replica 1, make some updates
- Client moves, accesses replica 2
- Modifications to replica 1 may not have migrated to replica 2 yet!

Monotonic Reads

- The read operations performed by a single process *P* at two different local copies (R1 and r2) of the same data store.
- If a client **has seen** a value of x at a time t, it will **never** see older version of x in the future
- Reading incoming email messages will fetches the latest updates
- 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.



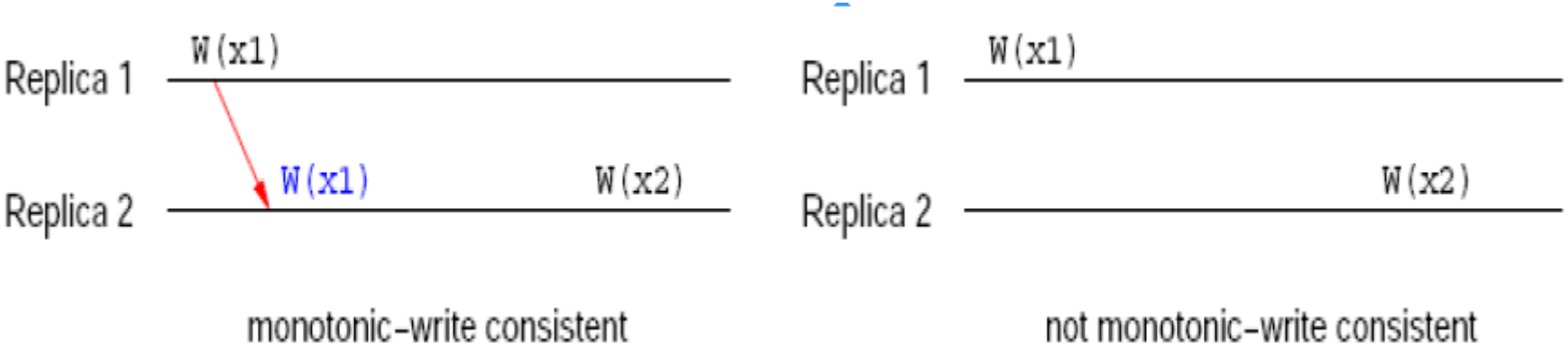
monotonic-read consistent



not monotonic-read consistent

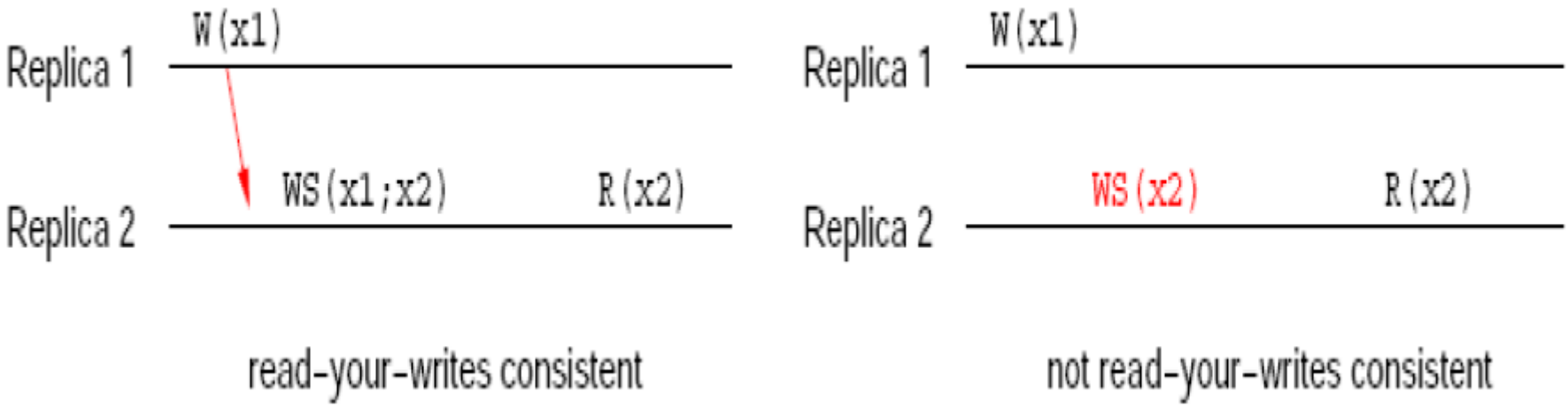
Monotonic Write

- The write operations performed by a single process *P* at two different local copies of the same data store
- A **write** operation on data item x is **completed** before any successive write on x by the same client
- All writes by single client are sequentially ordered
- Eg.: maintaining version of replicated files in correct order everywhere



Read your Write

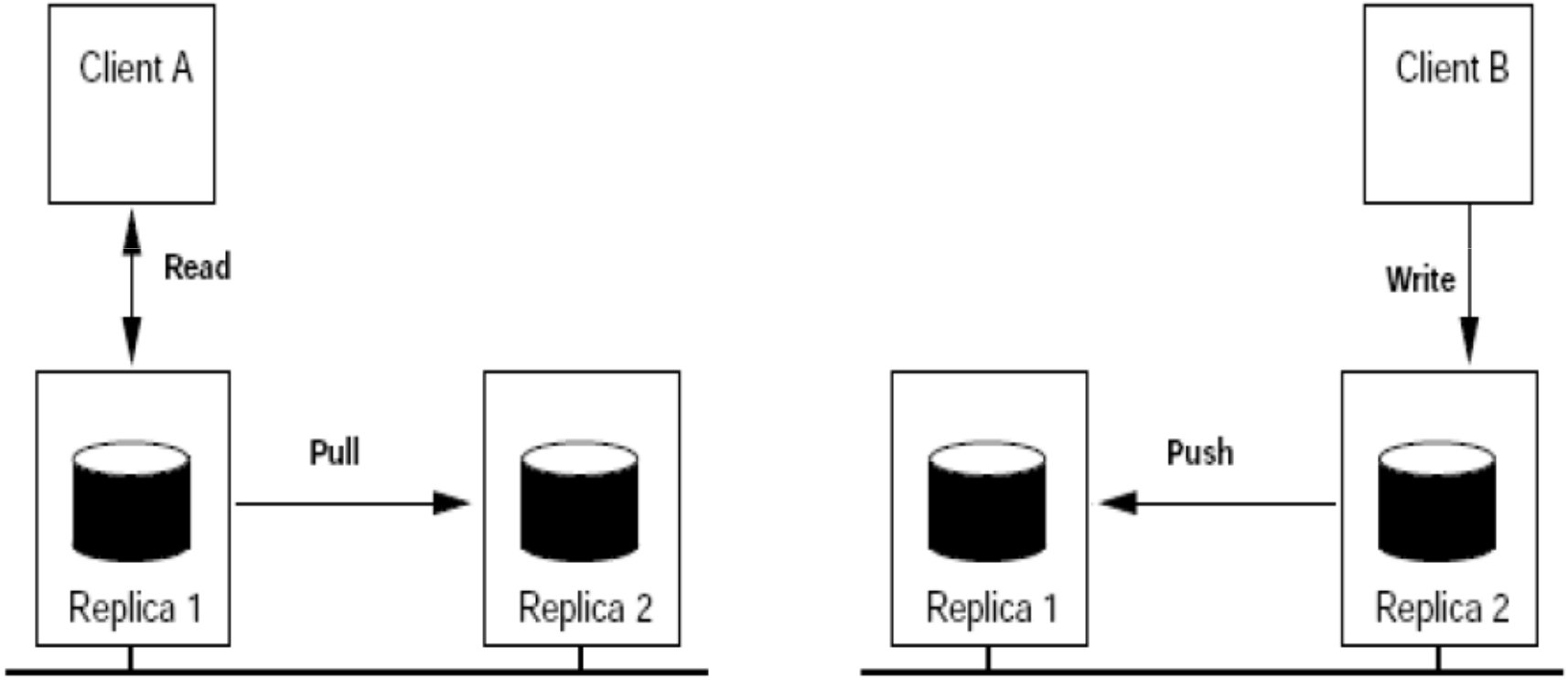
- Effect of a **write** on x will always be seen by a **successive read** of x by same client
- Ex: editor and browser, if not integrated, you may not read-your-writes of an HTML page



Write follow reads

- A write operation on x will be performed on a copy of x that is **up to date** with the values most recently read by the same client
- Ex: comments on news group, let A an article read recently, R the response to that article, then R must follows A.

Update propagation



Pull dan Push

- Pull
  - On request by read
  - Client-based pull
  - client requests another server to send it any updates it has at that moment.
  - R/W low (depends on read frequently)
  - polling delay
- Push
  - When client writes, it pushes to all replicas
  - updates are propagated to other copies actively
  - Server-based push
  - Have to keep track all replicas
  - R < W
  - Use lease time

Perbedaan

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

More...

- Consistency and Redundancy
  - All copies must be strongly consistent
  - All copies must contain full state
  - Reduced consistency -> reduced reliability
- Consistency and Performance
  - Consistency requires extra work + communication
  - Can results in loss of overall performance
- Consistency and Scalability
  - Implementation of consistency must be scalable
  - Avoid centralized approach
  - Avoid too much communication

NEXT

- Peer to Peer Systems
- Ada jadwal ganti untuk tgl 30 Nov (kelas A)
  - Mau tgl 26/27/3/4??
- Ada jadwal ganti untuk presentasi:
  - Tgl 6 Des 2010
  - Jam 8 (A-B33) dan 11 (B-D33)