Algorithms and Distributed Systems 2019/2020 (Lecture Ten – Final Lecture)

MIEI - Integrated Master in Computer Science and Informatics

Specialization block

João Leitão (jc.leitao@fct.unl.pt)

Lecture structure:

- Homework Solution
- Atomic Commit Problem
- Two Phase Commit
- Three Phase Commit

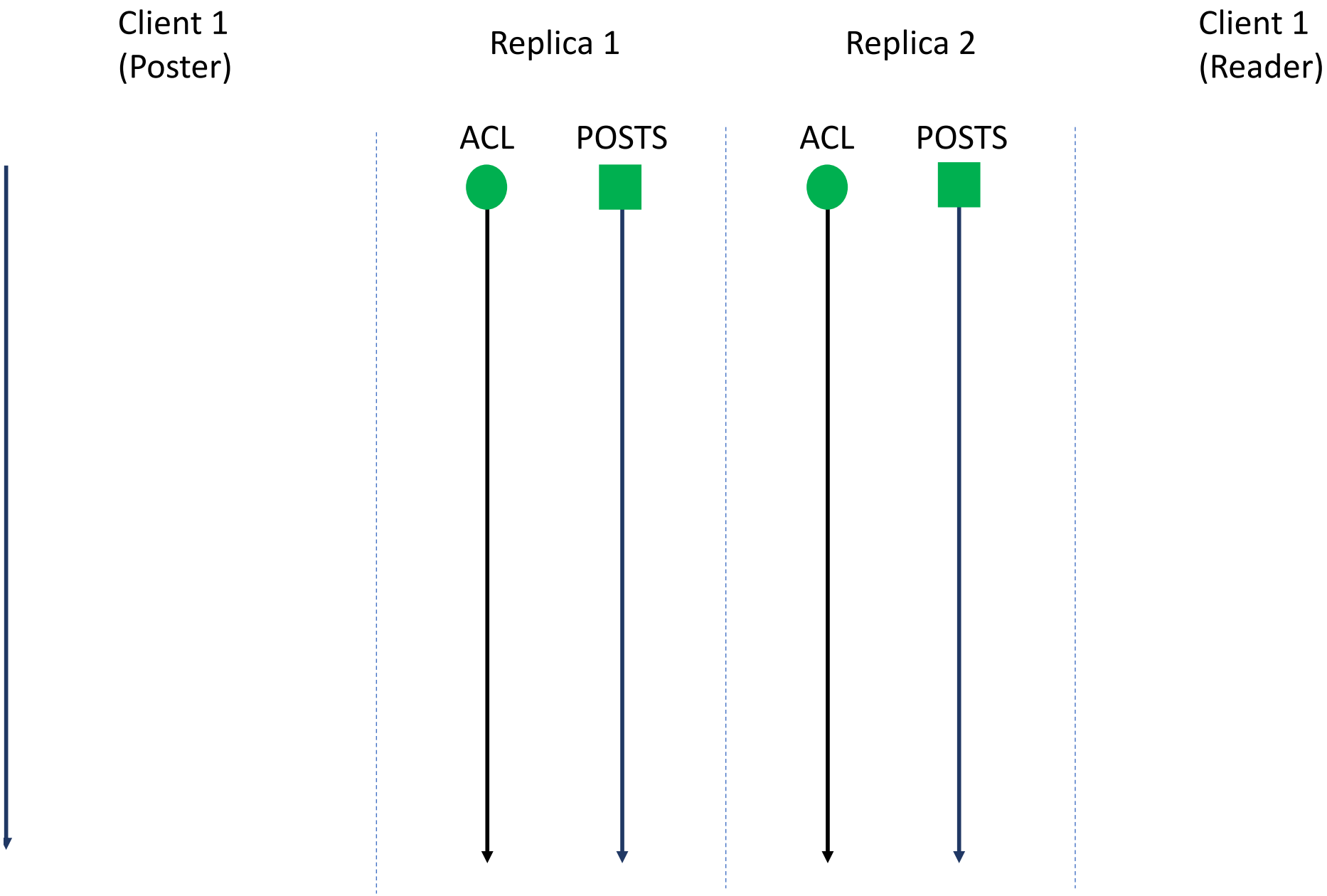
(Final) Homework 4:

- Consider the exemple of the social network with the access control list and the photo.

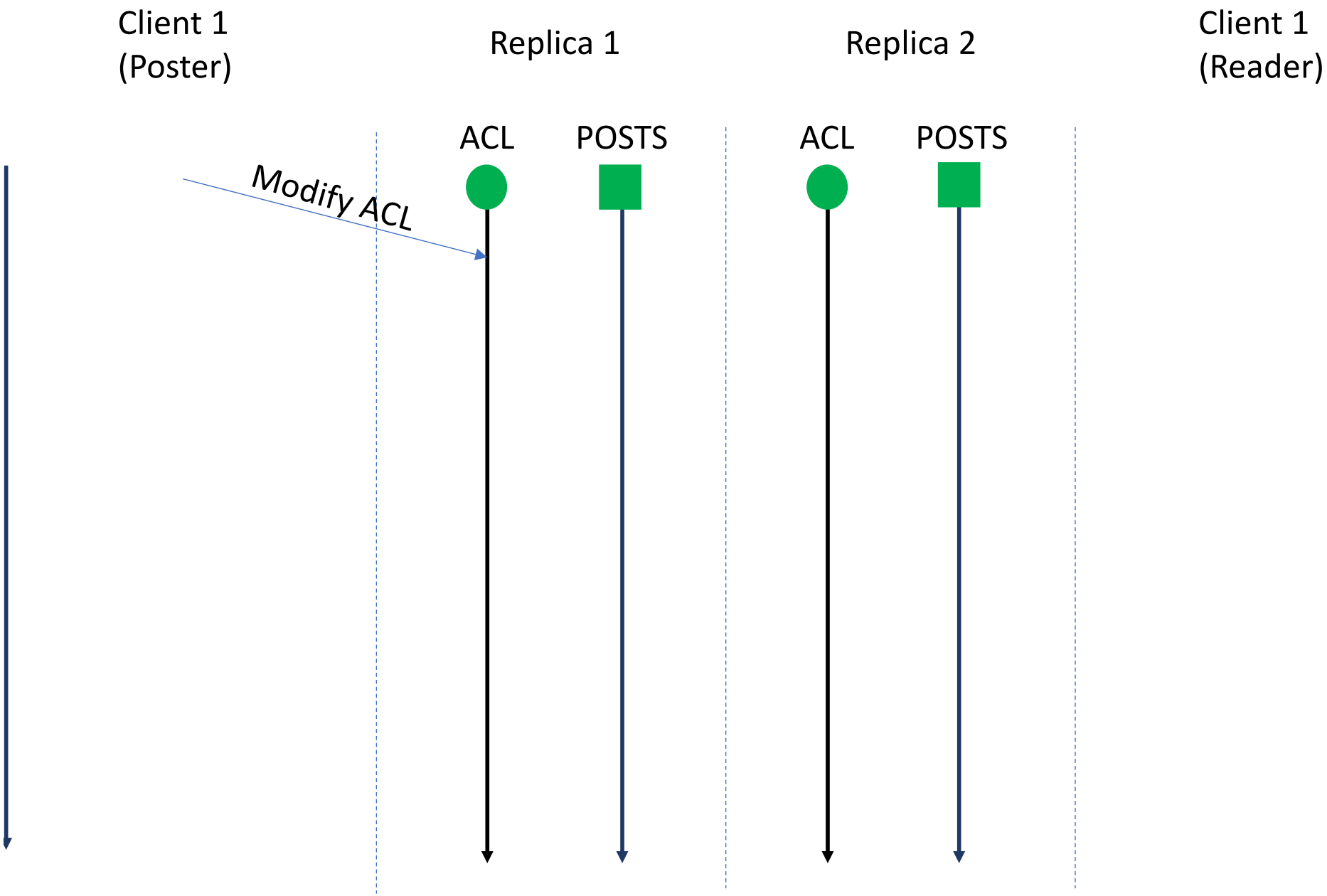
- If operations are isolated, and you start by updating the access control list and then post the photo you do not want to be seen, causality might not be enough to avoid the scenario discussed here.

- Explain why with an exemple and considering the properties of causal consistency
  (no need for pseudo-code).

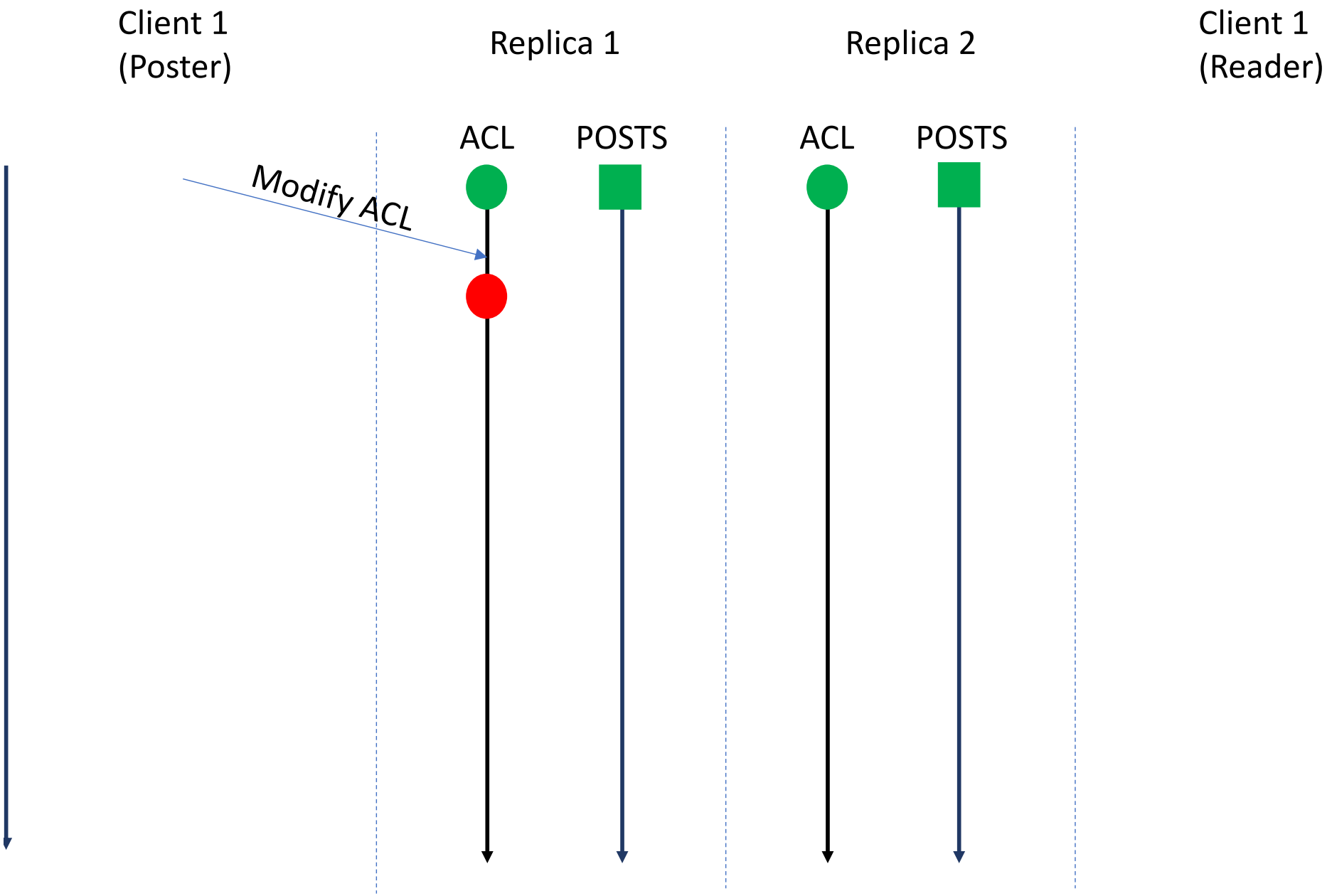
How can this go Wrong:



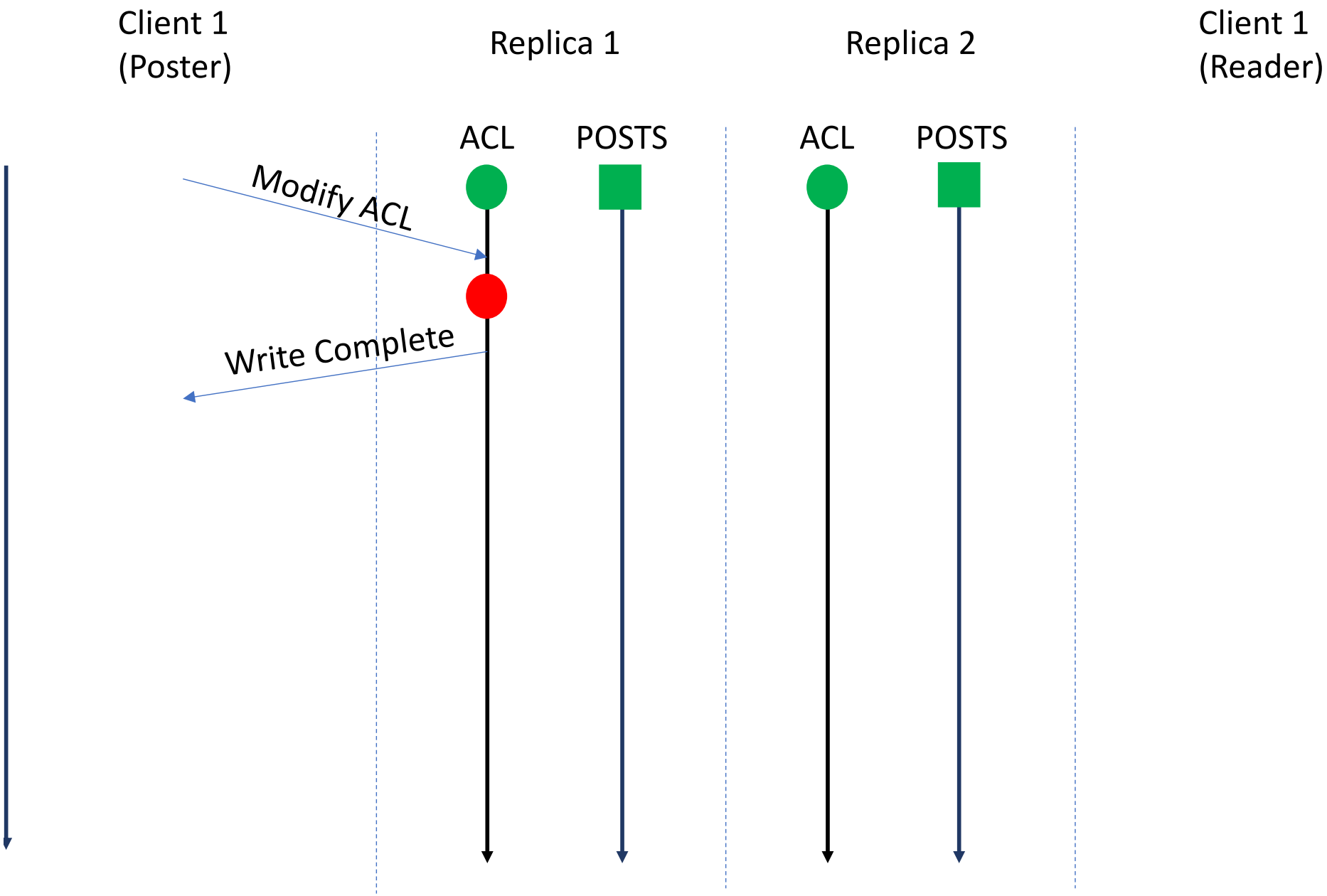
How can this go Wrong:



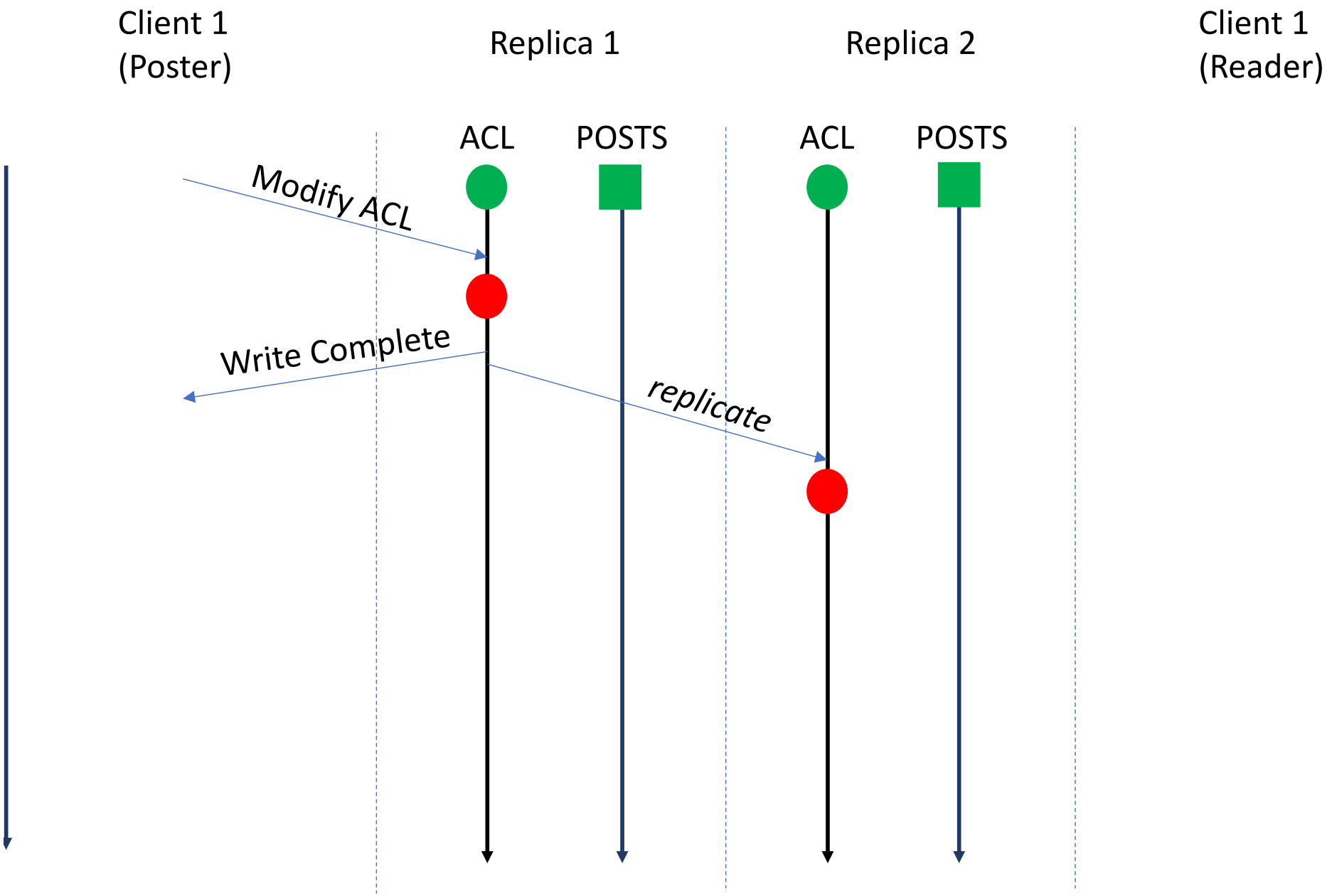
How can this go Wrong:



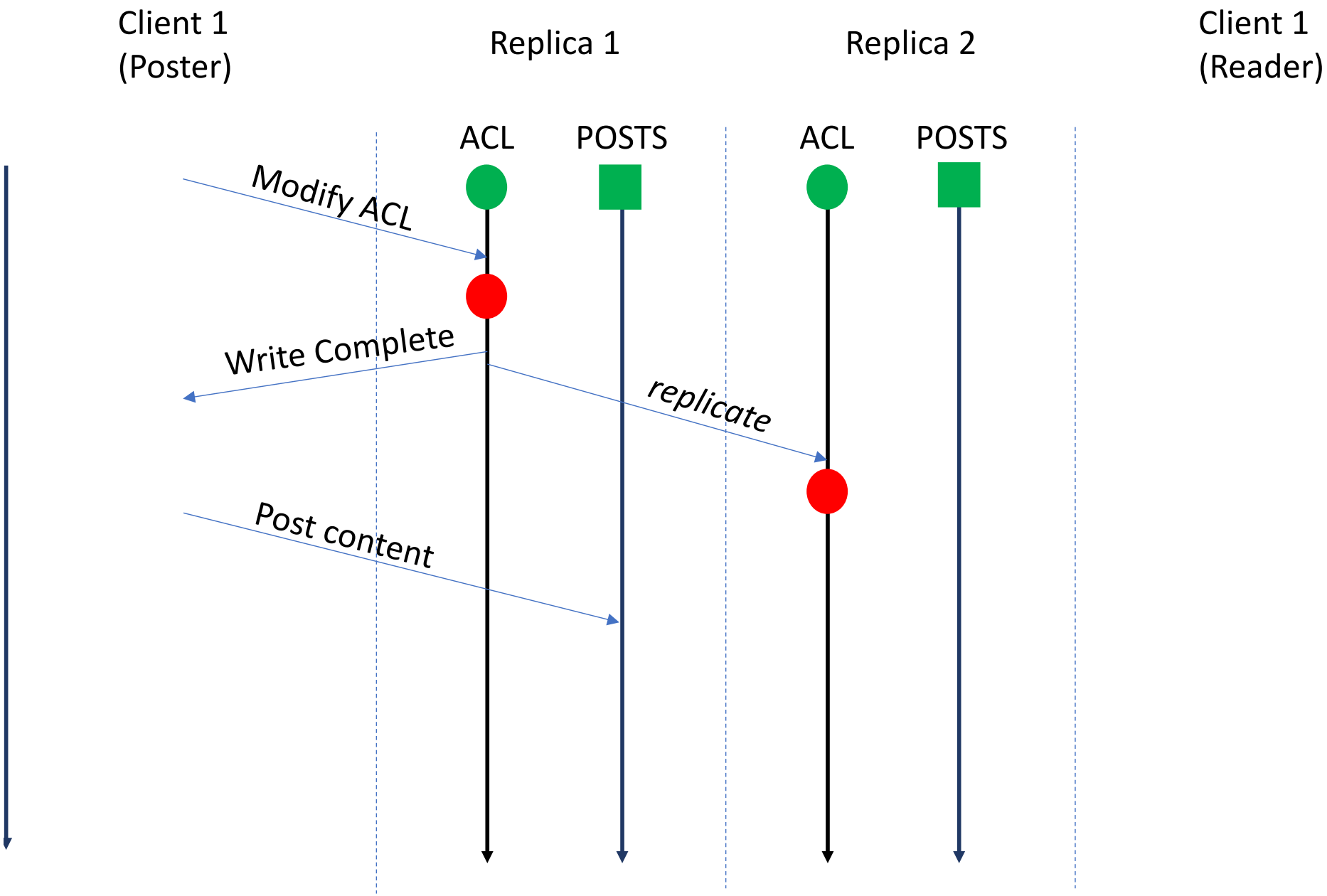
How can this go Wrong:



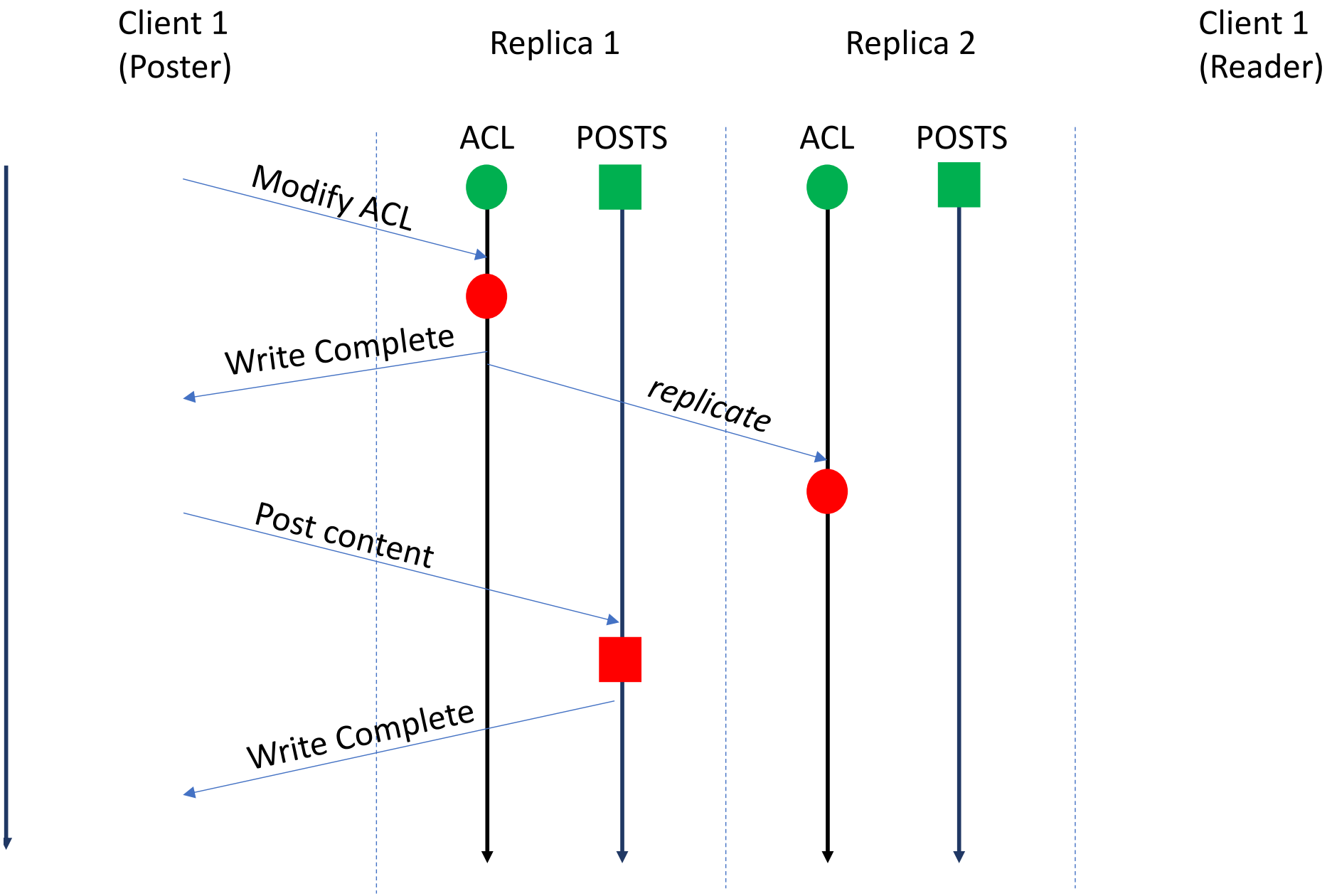
How can this go Wrong:



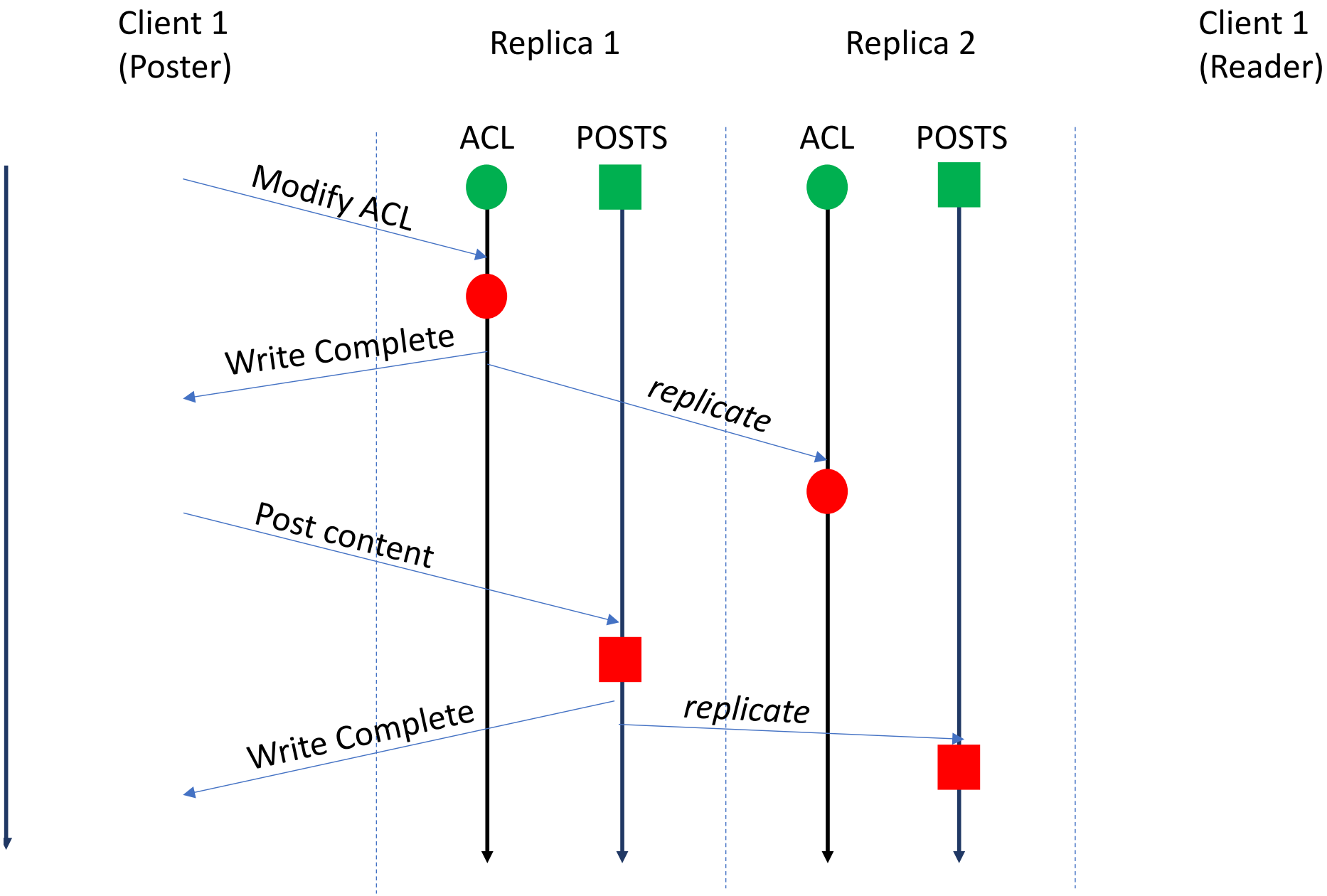
How can this go Wrong:



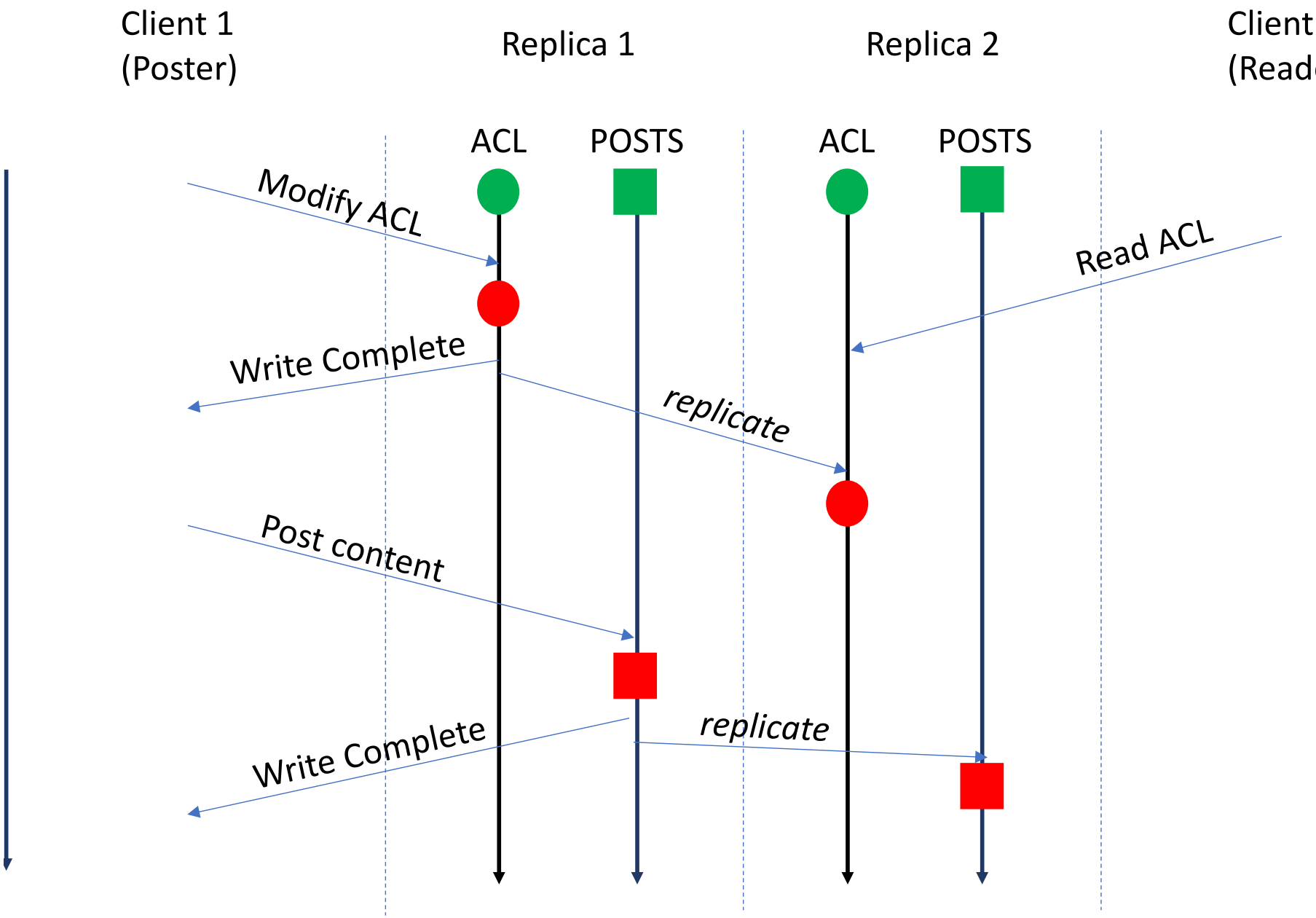
How can this go Wrong:



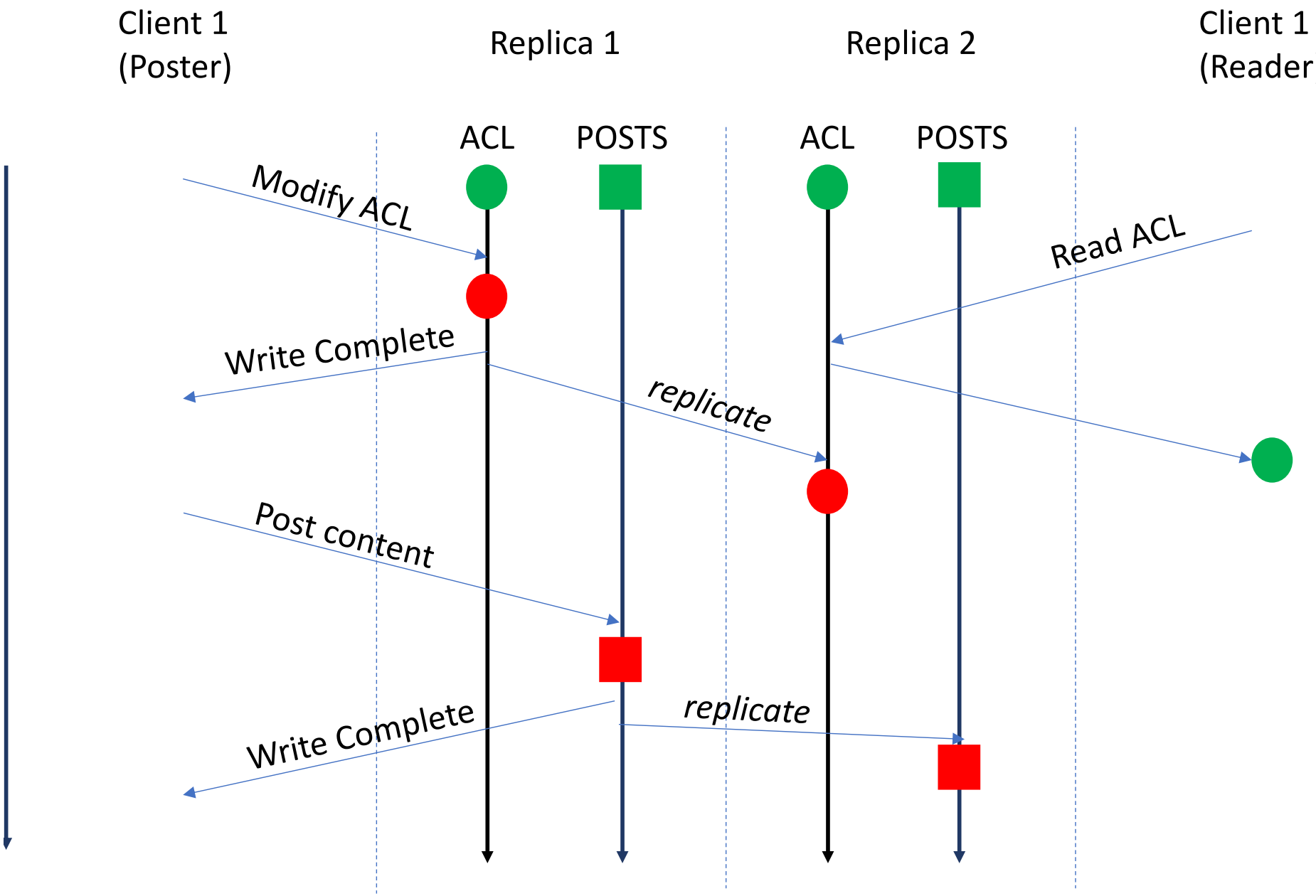
How can this go Wrong:



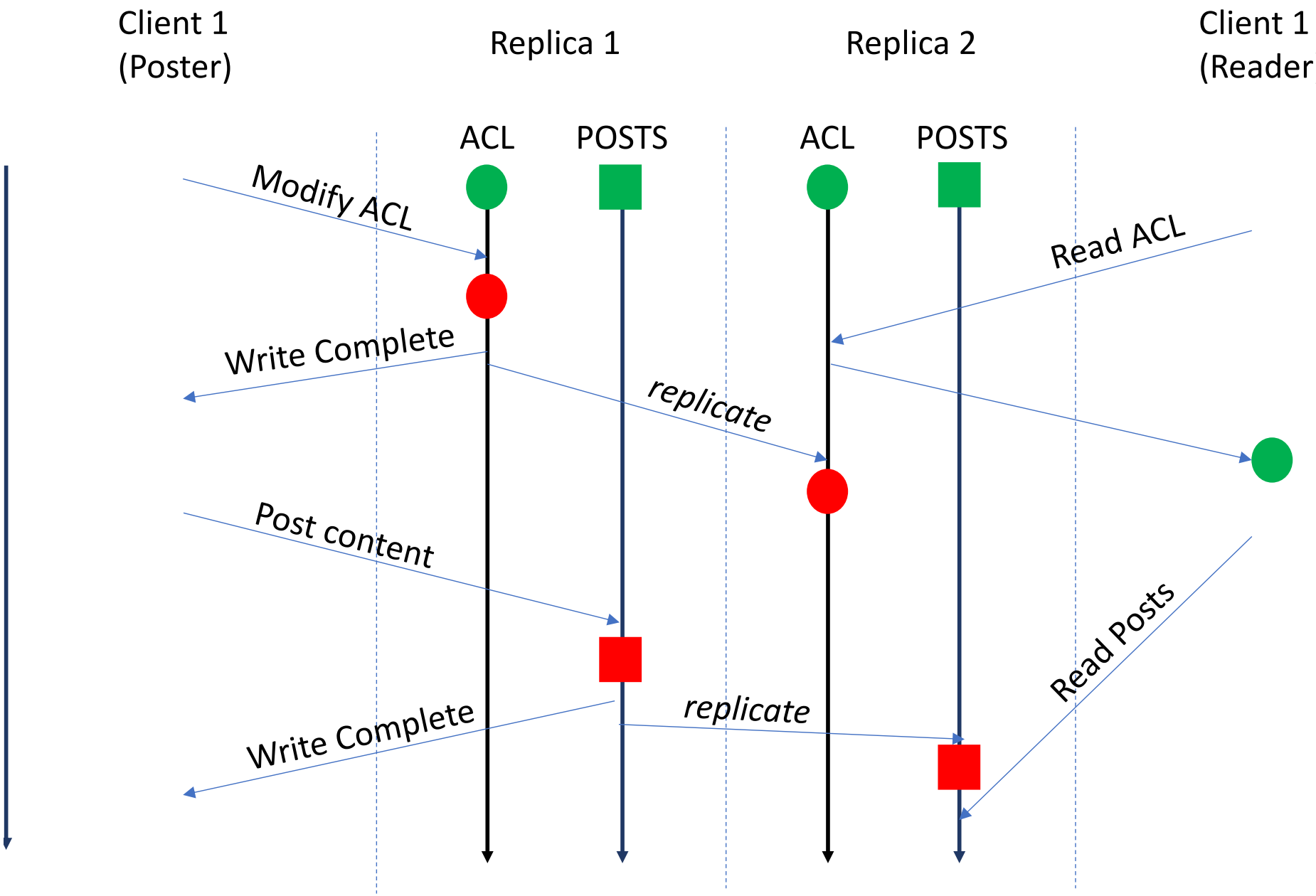
How can this go Wrong:



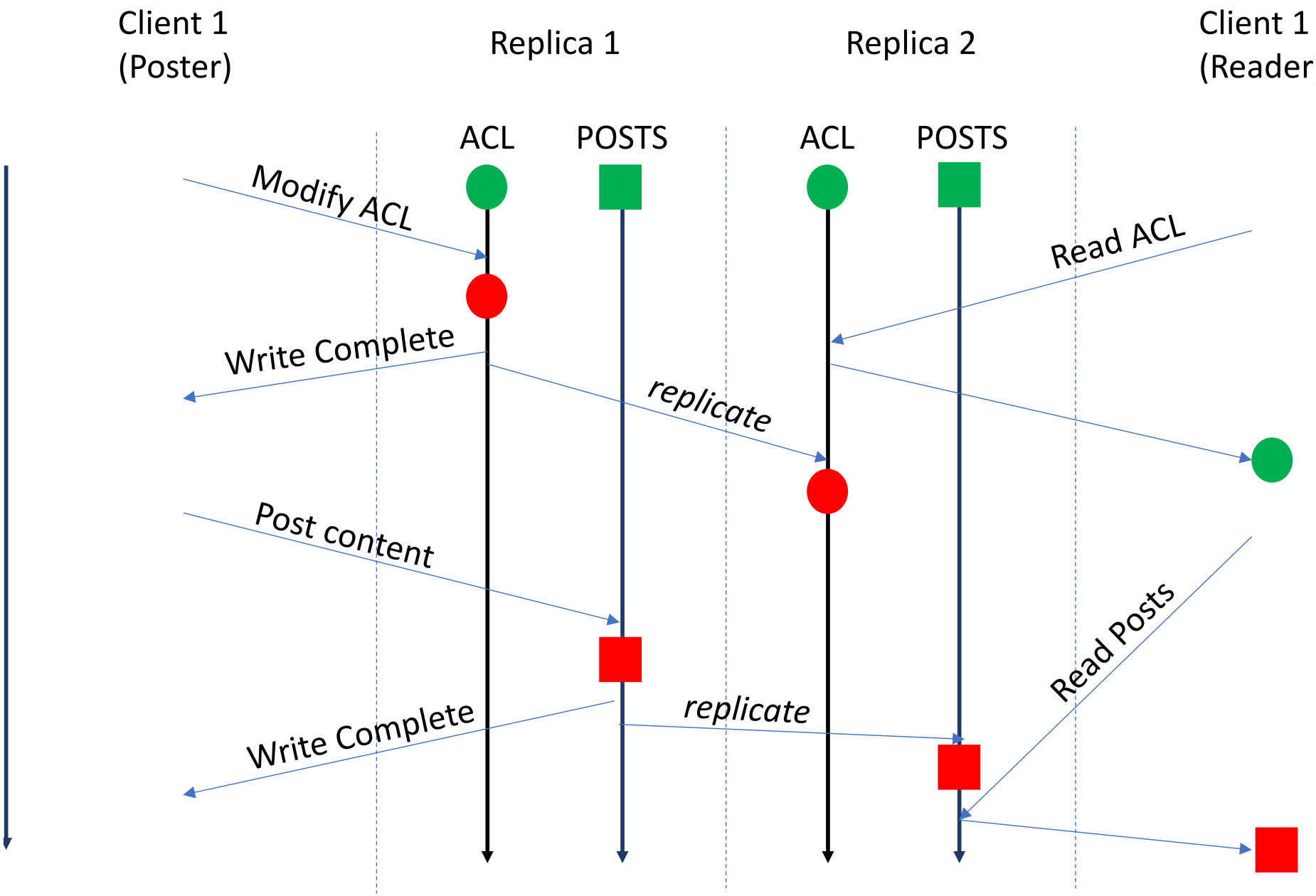
How can this go Wrong:



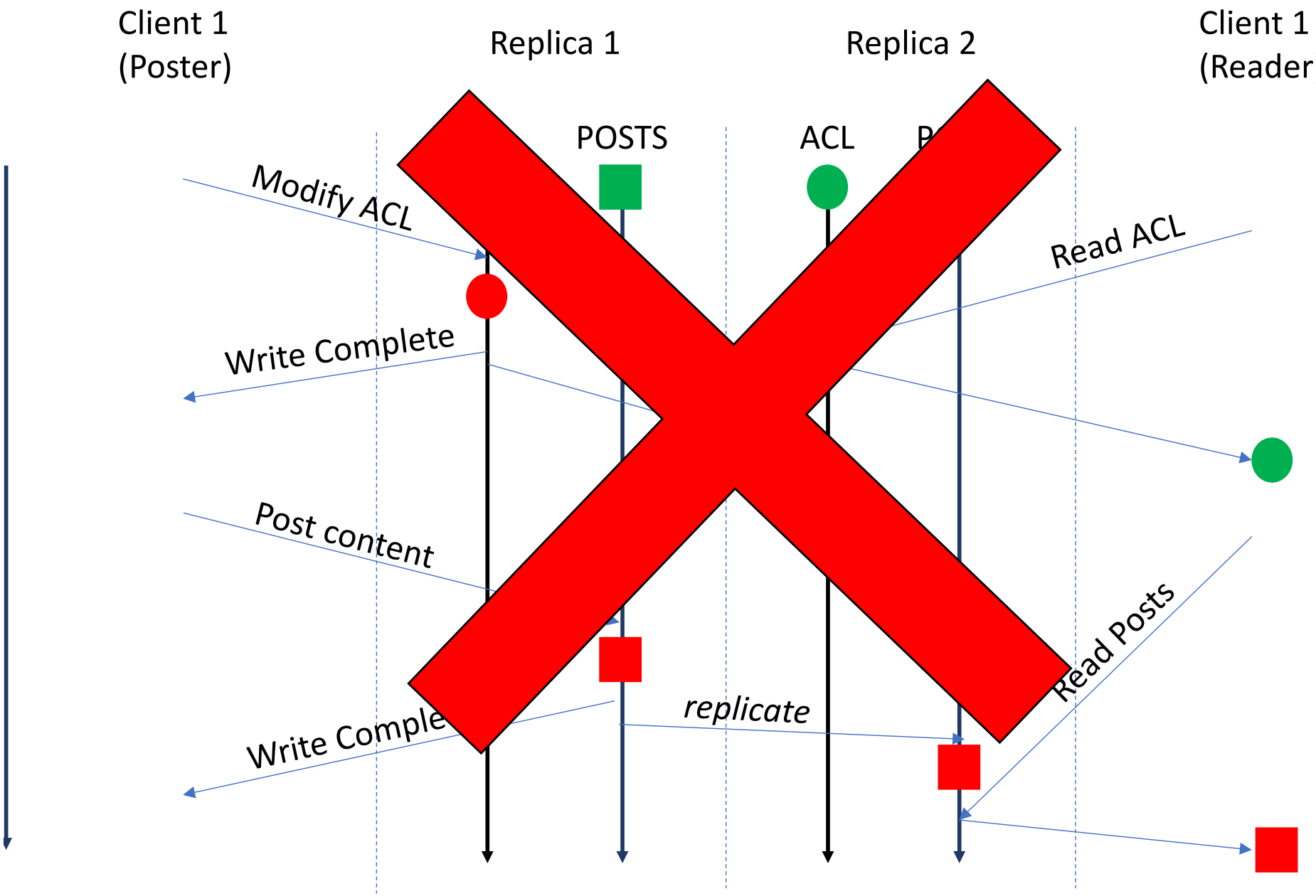
How can this go Wrong:



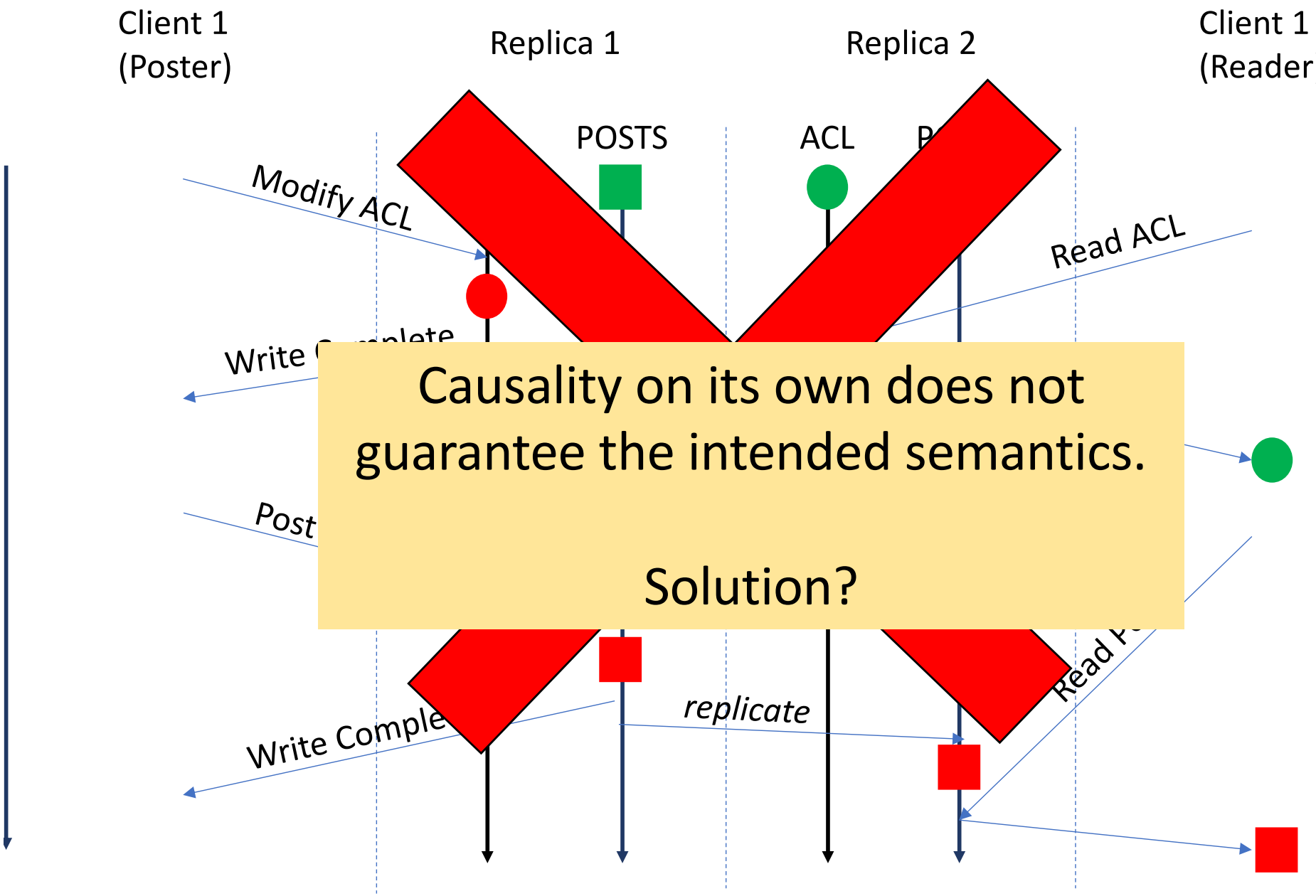
How can this go Wrong:



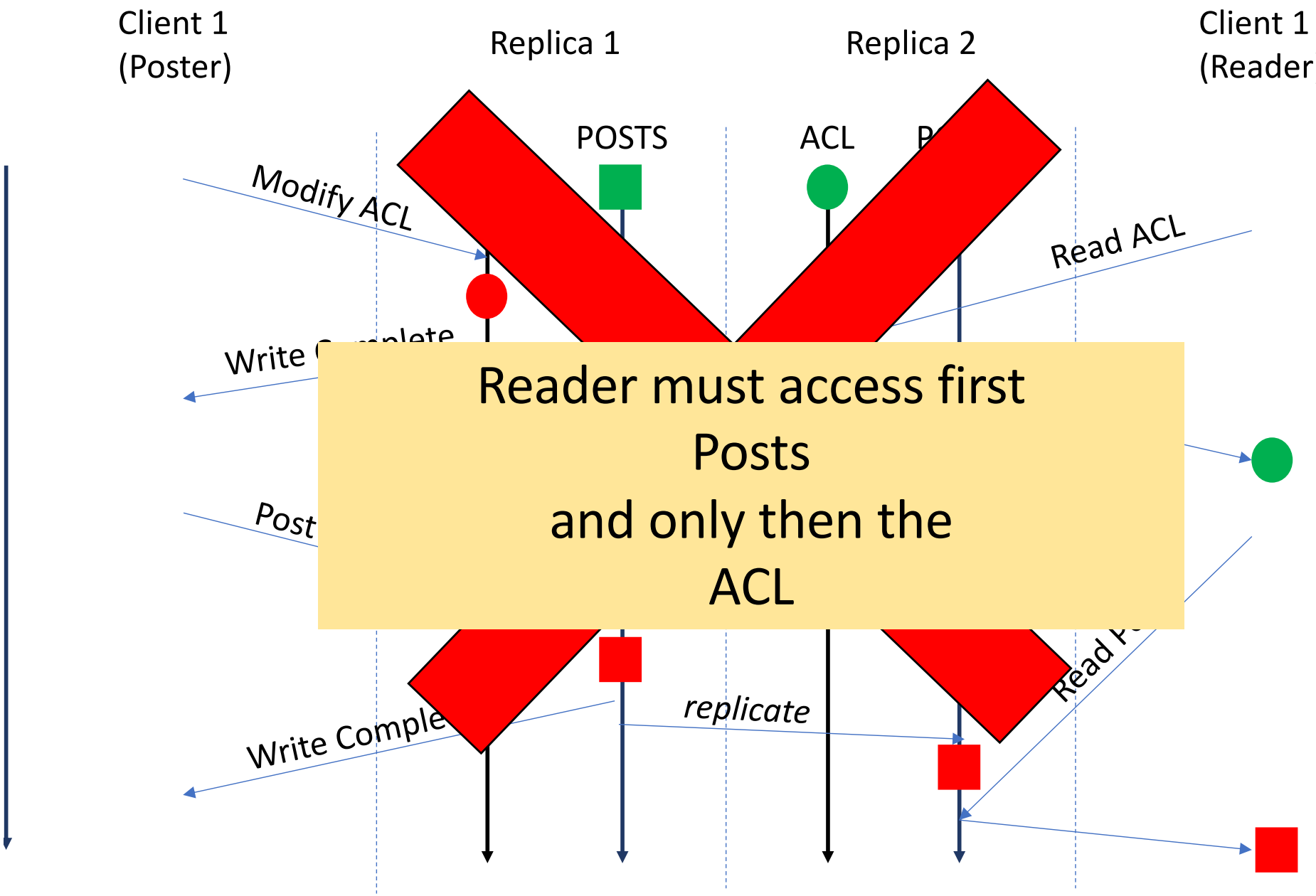
How can this go Wrong:



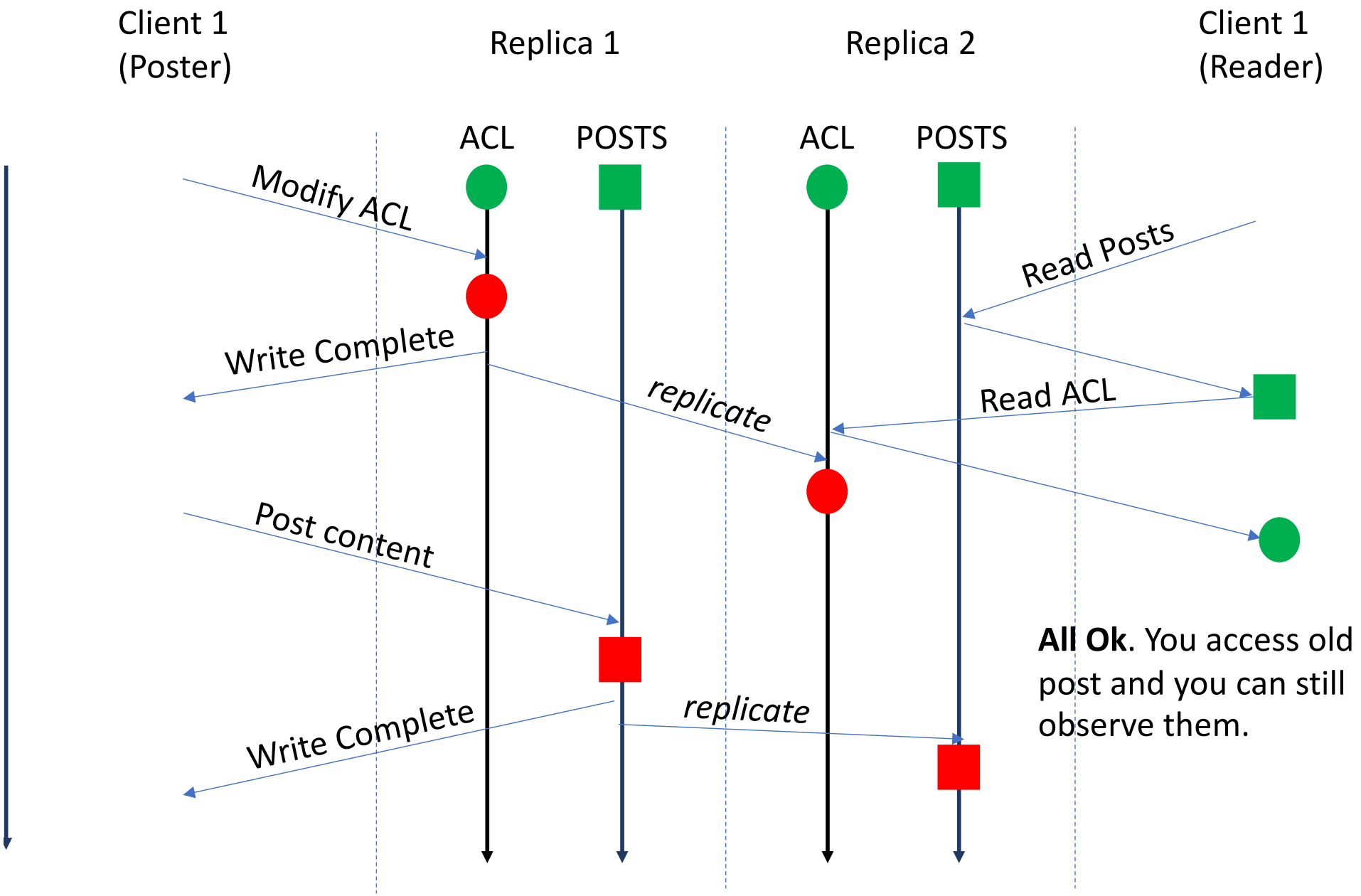
How can this go Wrong:



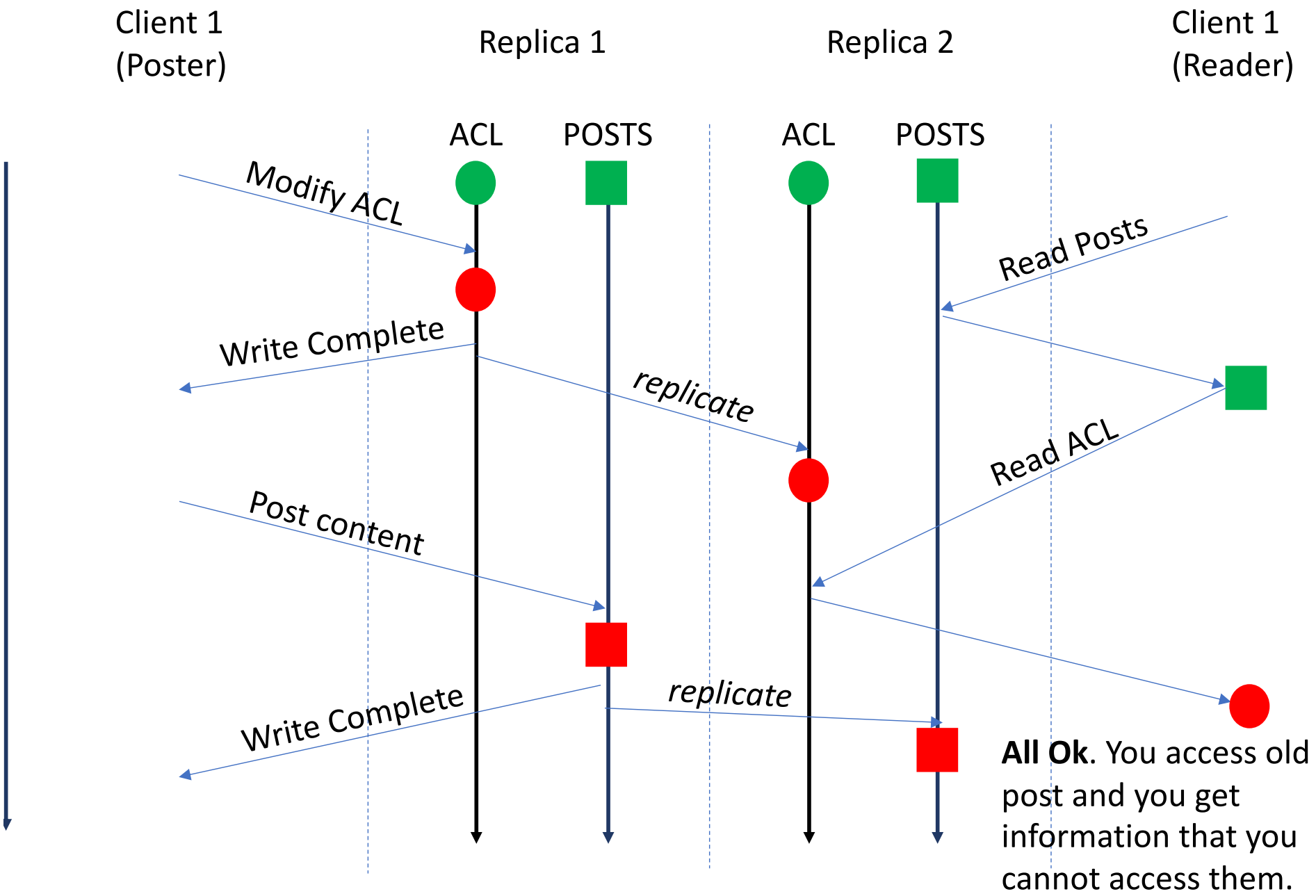
How can this go Wrong:



How can this not go Wrong (Take1):

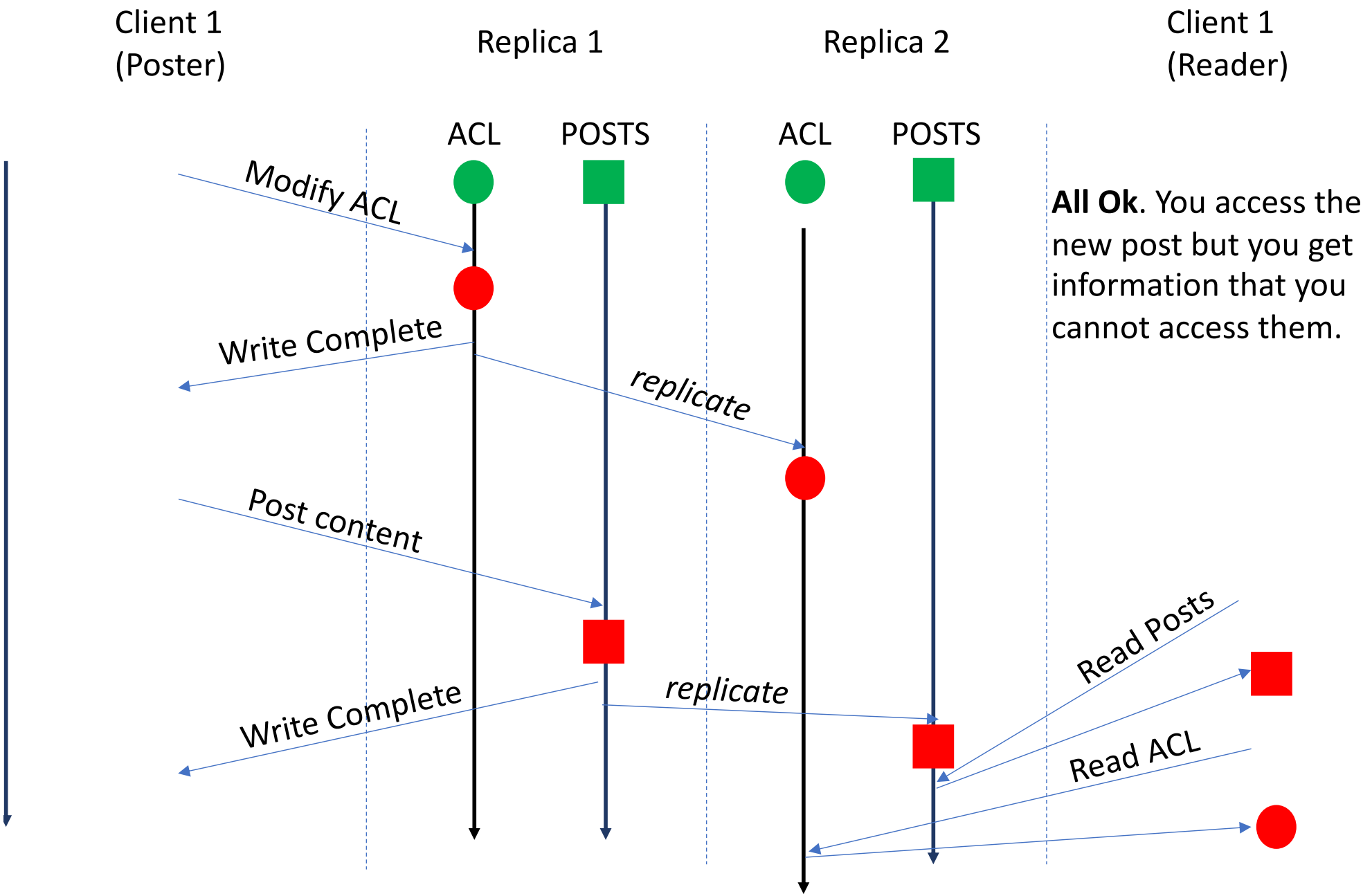


How can this not go Wrong (Take2):



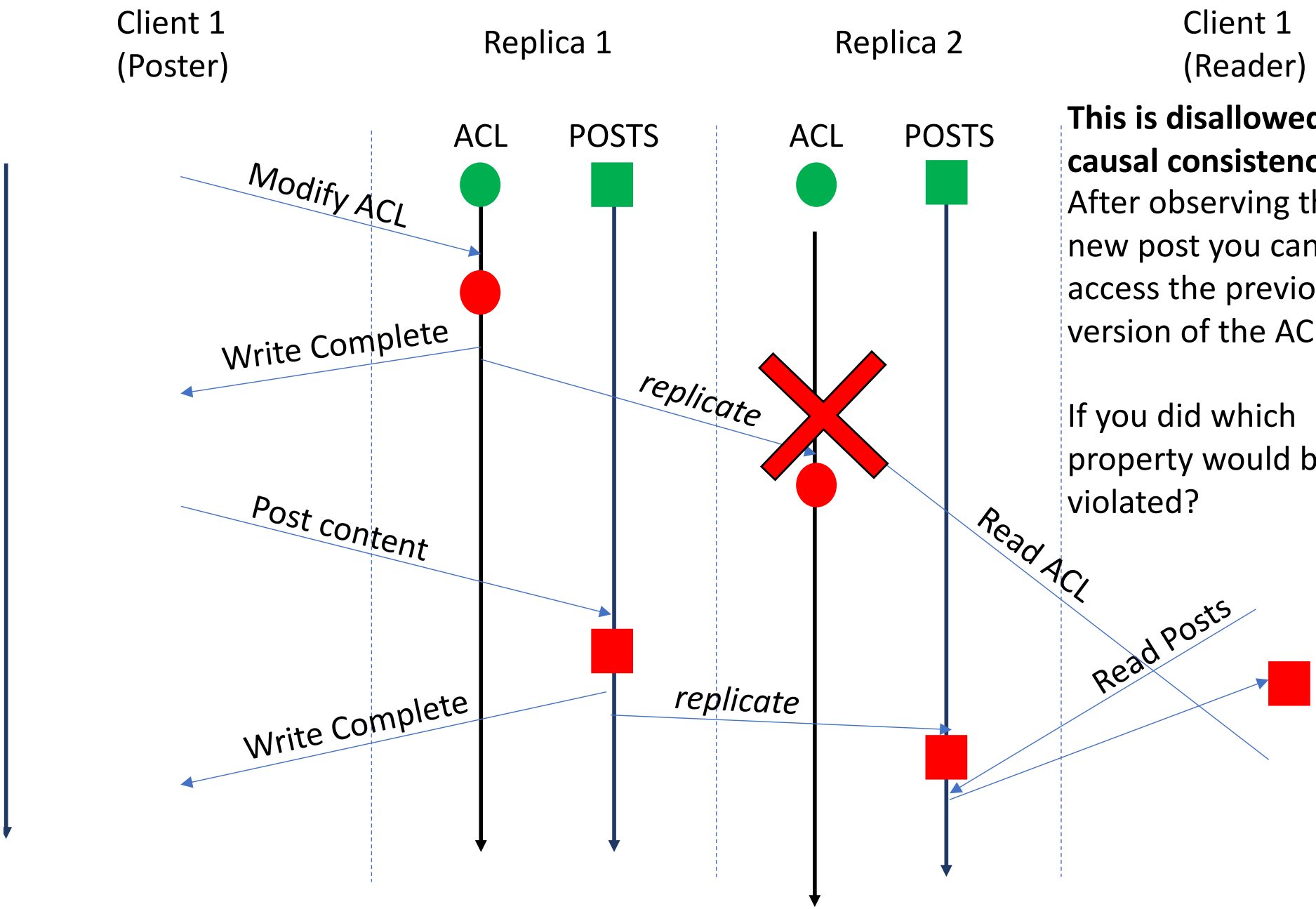
All Ok. You access old post and you get information that you cannot access them.

How can this not go Wrong (Take3):



All Ok. You access the new post but you get information that you cannot access them.

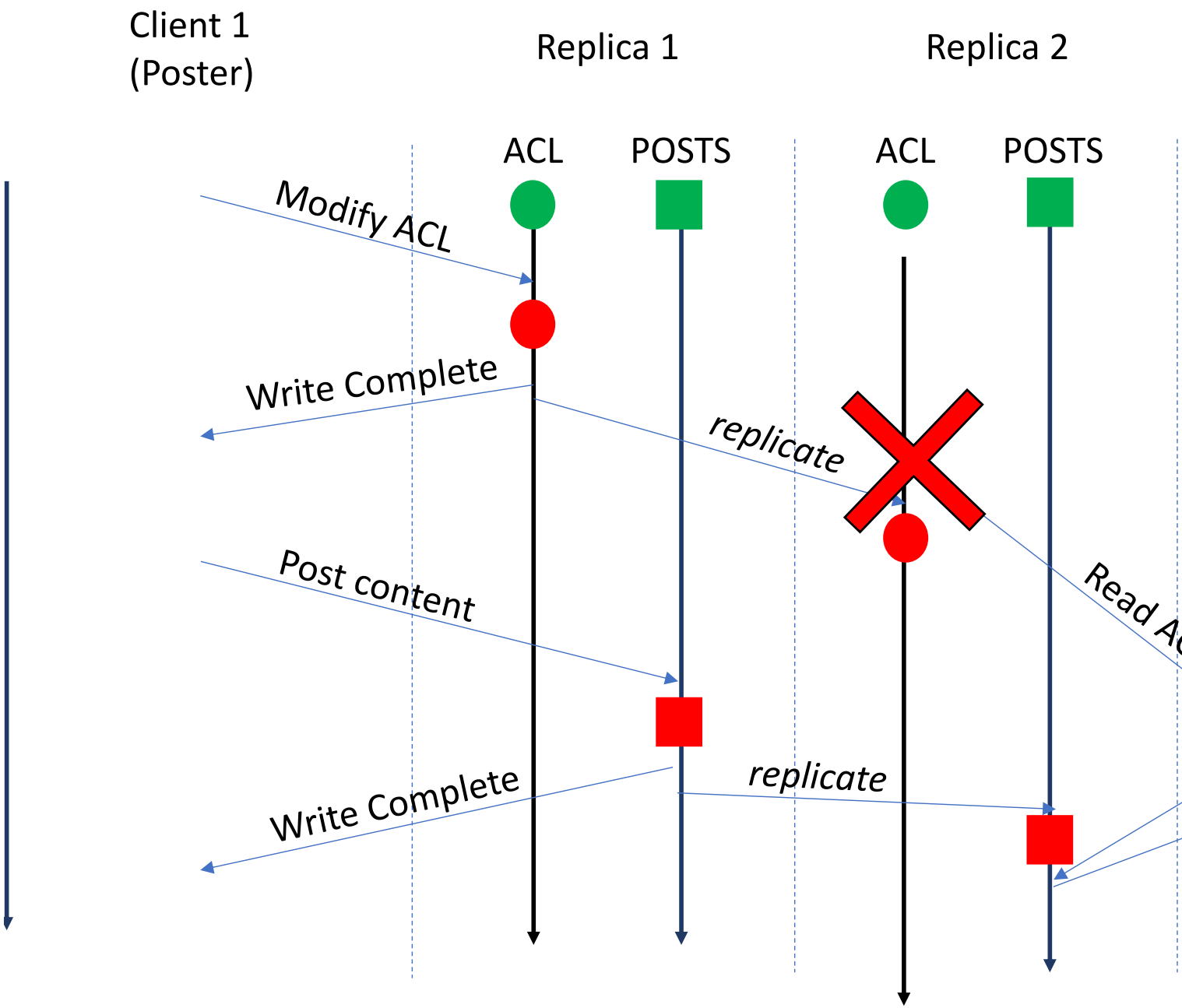
How can this not go Wrong (Take3):



This is disallowed by causal consistency. After observing the new post you cannot access the previous version of the ACL.

If you did which property would be violated?

How can this not go Wrong (Take3):



This is disallowed by causal consistency. After observing the new post you cannot access the previous version of the ACL.

If you did which property would be violated?

Monotonic Writes

Previous Class:

- Cap Theorem
- Causal Consistency
- Storage Solution using Causal Consistency (Labs)
  - Variant of Chain Replication

Let's Think about Concrete Applications for Distributed Storage

- Social Networks:
  - Manipulation of two objects: Access List and Post List
  - Can this be solved with causal consistency?

Let's Think about Concrete Applications for Distributed Storage

- Social Networks:
  - Manipulation of two objects: Access List and Post List
  - Can this be solved with causal consistency?
  - What if one of the operations fails?

Let's Think about Concrete Applications for Distributed Storage

- Social Networks:
  - Manipulation of two objects: Access List and Post List
  - Can this be solved with causal consistency?
  - What if one of the operations fails?
  - There is no problem here, if one is unable to make a new post after modifying the access control list, there is no violation of the semantics of the operation.

Let’s Think about Concrete Applications for Distributed Storage

- Banking:
  - Transfer 500€ from account A to account B.
  - Can this be solved with causal consistency?
  - What if one of the operations fails?

Let's Think about Concrete Applications for Distributed Storage

- Banking:
  - Transfer 500€ from account A to account B.
  - Can this be solved with causal consistency?
  - What if one of the operations fails?

- There is a problem here, If I start to execute the operation that removes the money from account A, but then fail to put the money in account B, money disappears… and this is a big no no.

Let's Think about Concrete Applications for Distributed Storage

- Banking:
  - Transfer 500€ from account A to account B.
  - Can this be solved with causal consistency?
  - What if one of the operations fails?

- There is a problem here, If I start to execute the operation that removes the money from account A, but then fail to put the money in account B, money disappears… and this is a big no no.
- Even if we use Paxos to coordinate each operation this might happen...

We need a different abstraction…

- The abstraction that we are looking for allows to execute two or more operations in a way that:
  - Either all operations terminate with success.
  - Or if one of the operations fails, then none of the operations can have any visible effect.

- Going back to the banking example:
  - If I cannot take the 500€ from account A, then I should also not add the 500€ to account B.
  - If for some reason the operation to add 500€ to account B fails, then the effects of removing 500€ from A should never be visible.

We need a different abstraction…

- The abstraction that we are looking is related with transactions:
  - A transaction is an operation that groups multiple read and write operations into a single (meta) operation.
  - Transactions were derived form the Database community, and they provide the **ACID** properties.
  - Transactions have a special operation to start (Begin Transaction) and a special operation to finish (Commit).
  - The final operation return value indicates if the operation succeeded of failed.

ACID Properties

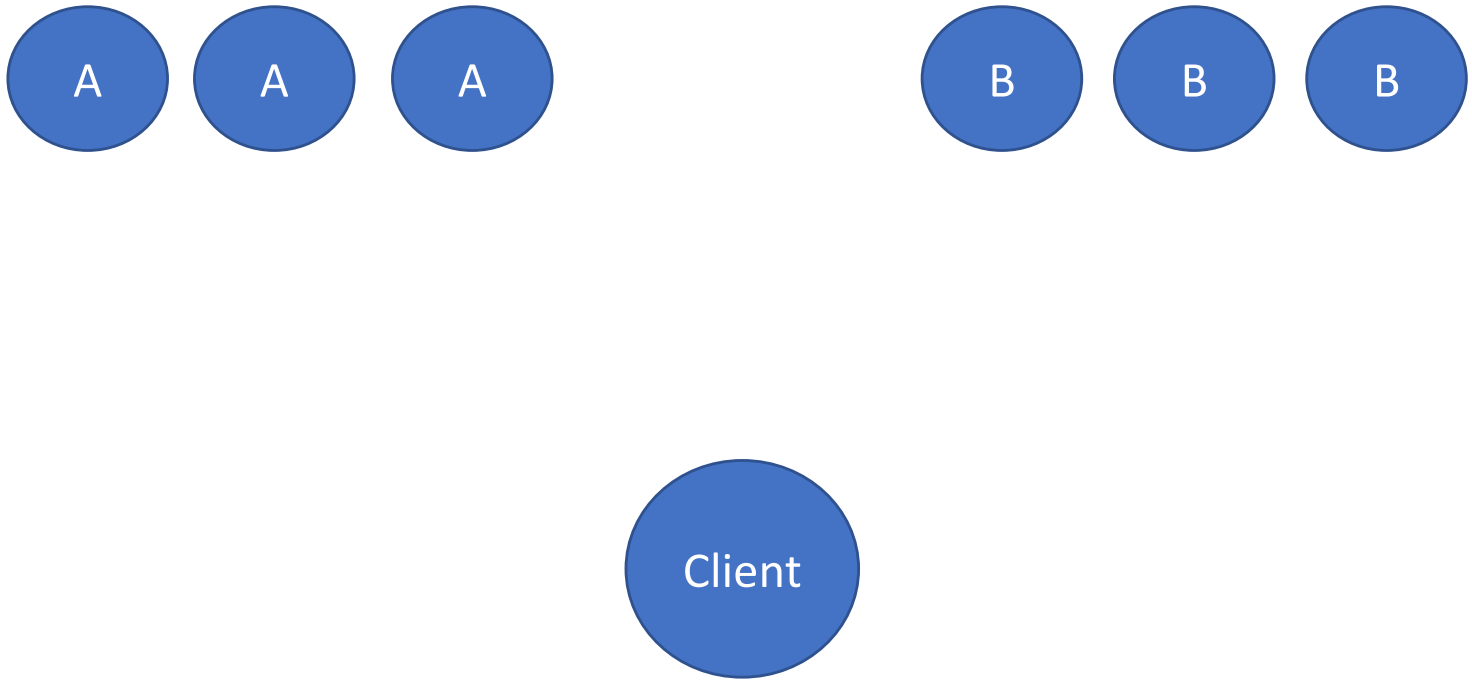
- **A**tomicity: All operations have their intended effects or none have.
- **C**onsistency: The state of the database respects all invariants of the data model before the execution of a transaction and after the execution of a transaction (*this is different from the consistency of replication protocols*).
- **I**solation: Each transaction executes with no interference from other (concurrent) transactions, either all effects of a transaction are observed by a running transaction or none are.
- **D**urability: The effects of a transaction that terminates successfully are not lost (independently of failures that might happen)

Distributed Transactions

- While Databases that provide ACID properties have multiple solutions to locally enforce these properties (e.g., MySQL, OracleDB, etc…) when generalizing the execution of transactions across multiple processes (not necessarily replicas) we have another challenge.

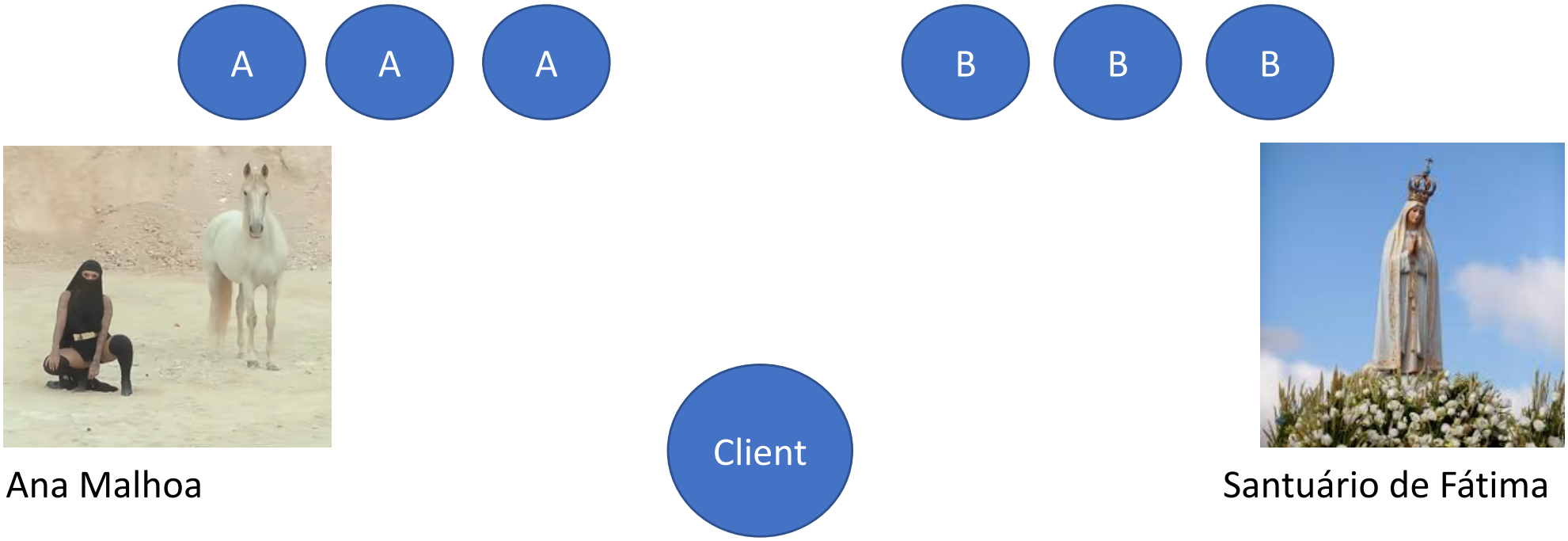
Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g., different Banks):



Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g., different Banks):

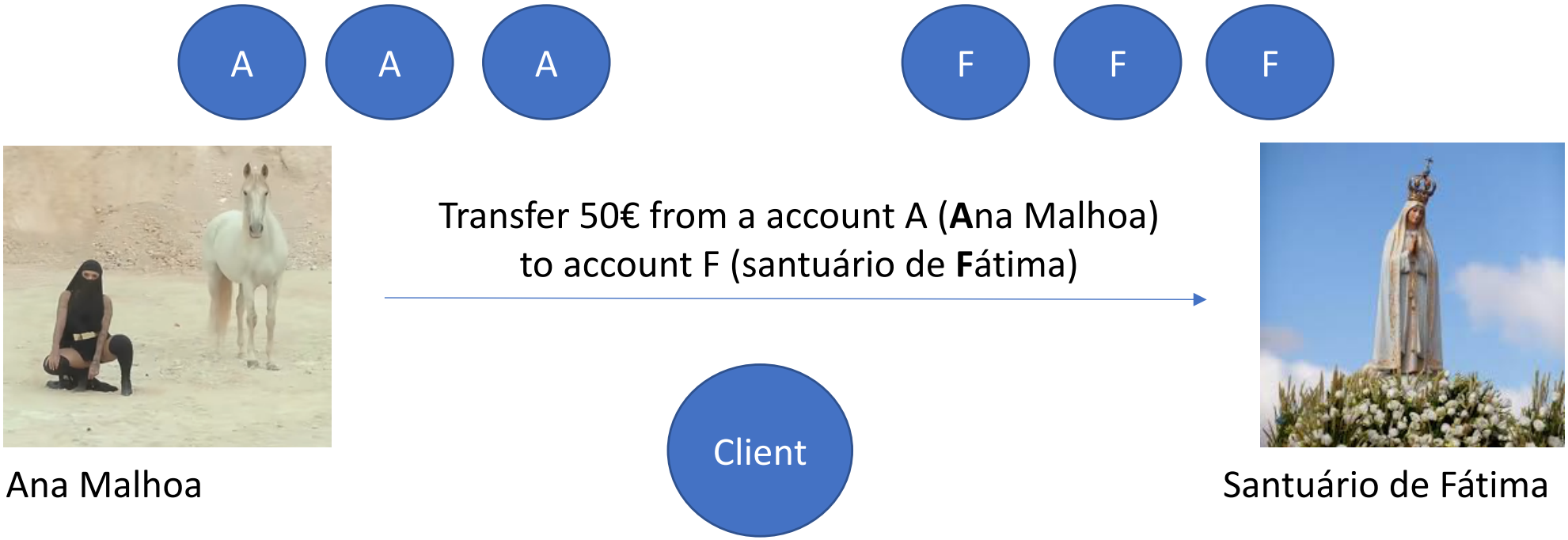


Ana Malhoa

Santuário de Fátima

Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g., different Banks):



Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g., different Banks):

Ana Malhoa

How can we do this?

Santuário de Fátima

Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and

Using State Machine Replication and Paxos would require:

- Having an operation Transfer(Acc1, Acc2, Amount) on the state machine that would only success if there is enough money in Acc1 and it is possible to modify the value of Acc2 accordingly).

Ana Malhoa

Client

Santuário de Fátima

Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and

Using State Machine Replication and Paxos would require:

- Having an operation Transfer(Acc1, Acc2, Amount) on the state machine that would only success if there is enough money in Acc1 and it is possible to modify the value of Acc2 accordingly).
- And then this operation would be ordered among all operations that can modify the values in any bank account…

Ana Malhoa

Client

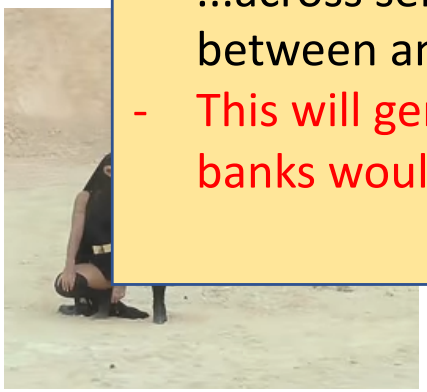
Santuário de Fátima

Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and

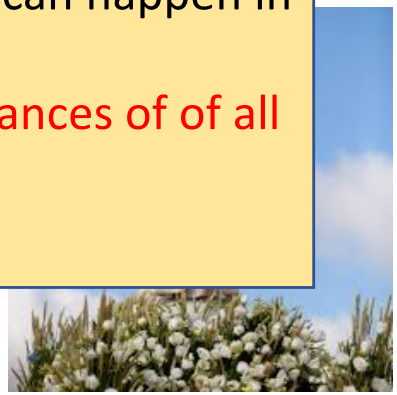
Using State Machine Replication and Paxos would require:

- Having an operation Transfer(Acc1, Acc2, Amount) on the state machine that would only success if there is enough money in Acc1 and it is possible to modify the value of Acc2 accordingly).
- And then this operation would be ordered among all operations that can modify the values in any bank account…
- ...across servers of all Banks in the World since a transfer operation can happen in between any account in any bank.
- This will generate a major contention, since a majority of paxos instances of of all banks would have to participate in each operation.



Ana Malhoa

Client



Santuário de Fátima

Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and

Using Paxos would require:

- Each set of replicas for each account to propose if they want to accept the transaction (i.e., if locally the invariants of the database are respected by the execution of the database or not).

Ana Malhoa

Client

Santuário de Fátima

Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and

Using Paxos would require:

- Each set of replicas for each account to propose if they want to accept the transaction (i.e, if locally the invariants of the database are respected by the execution of the database or not).
- However, the decisions that Paxos support are independent of all the proposed values. Therefore is a single set of replicas propose to accept the transaction, the decision could be accept.
- This would destroy the invariants of the application, since in our case if one of the replicas is unable to execute the operation then the whole transaction must abort.

Ana Malhoa

Client

Santuário de Fátima

Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g., different Banks):

Ana Malhoa

We need something else…

Santuário de Fátima

Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g., different Banks):

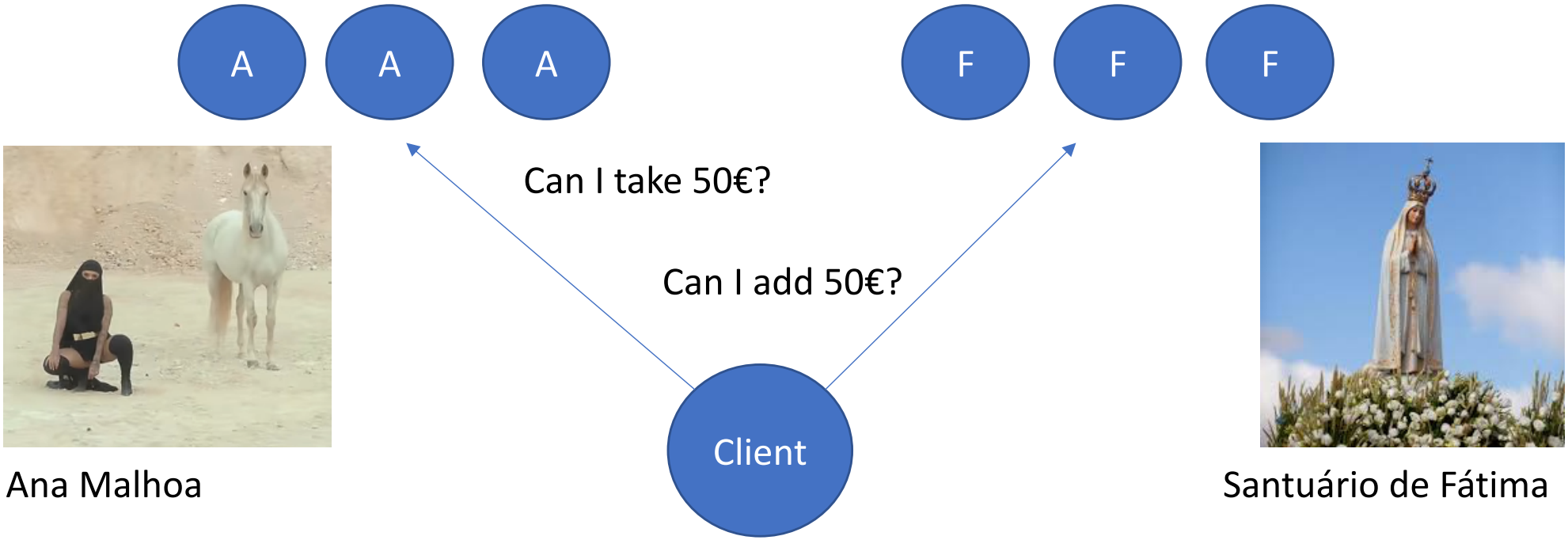
Ana Malhoa

What we are looking is for the
Atomic Commit Problem

Santuário de Fátima

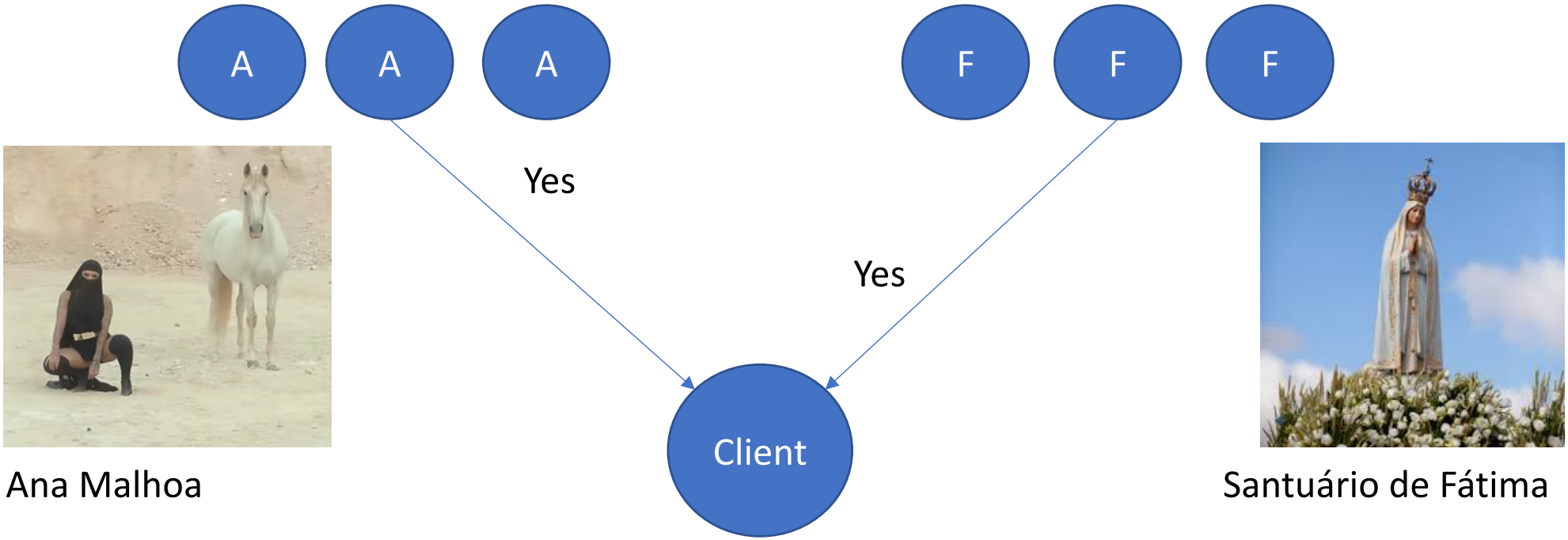
Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g, different Banks):



Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g, different Banks):

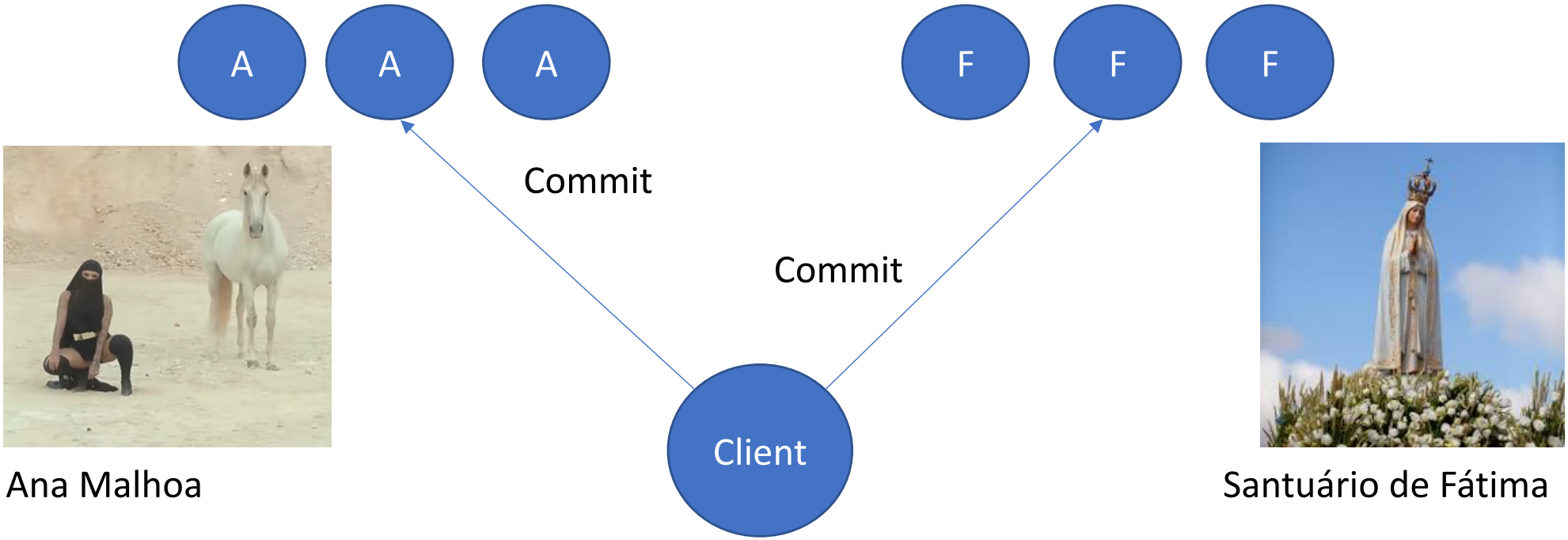


Ana Malhoa

Santuário de Fátima

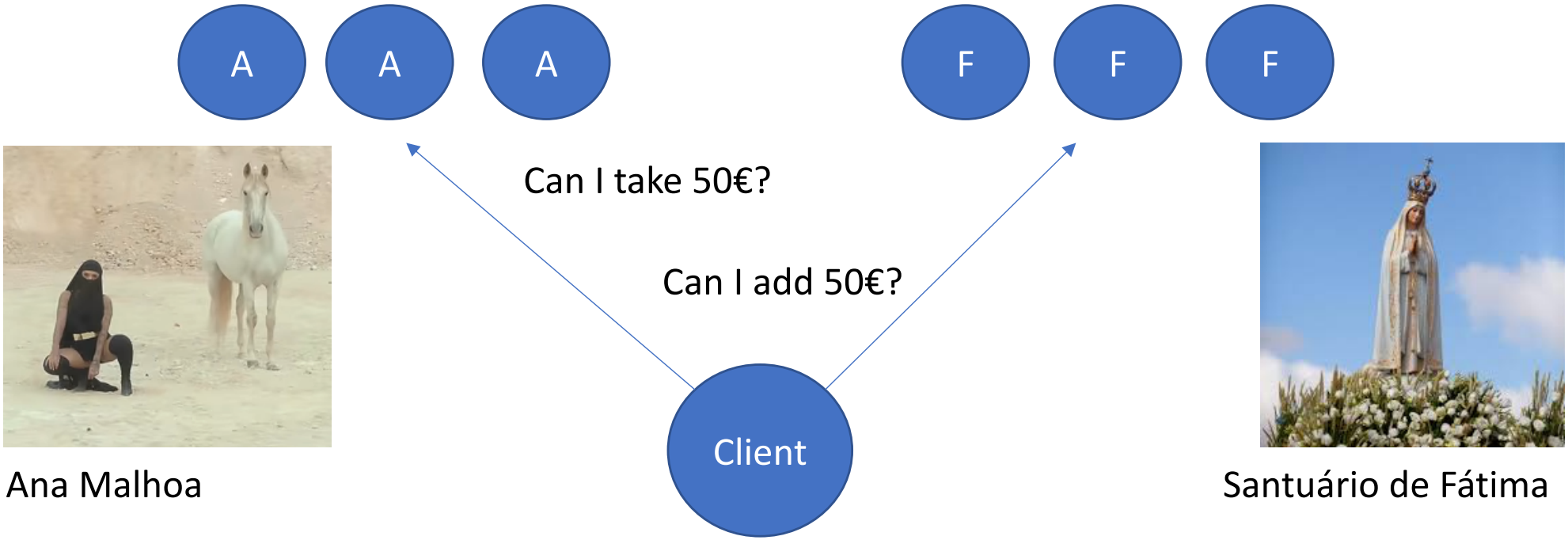
Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g, different Banks):



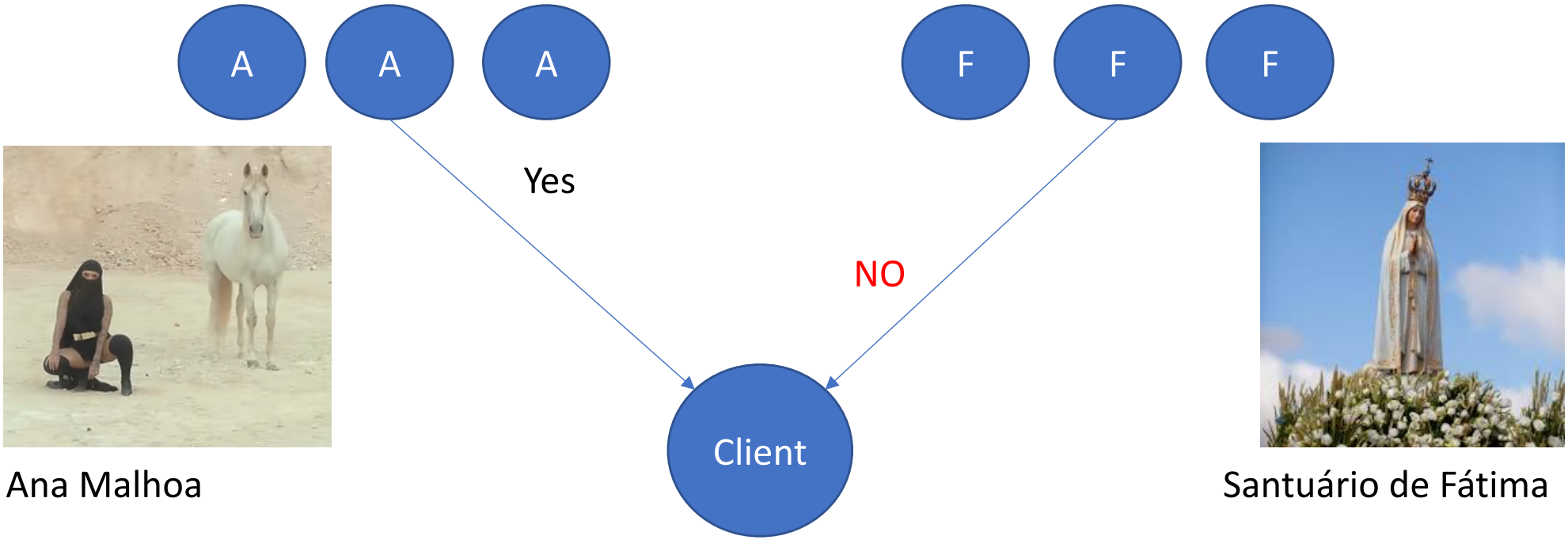
Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g, different Banks):



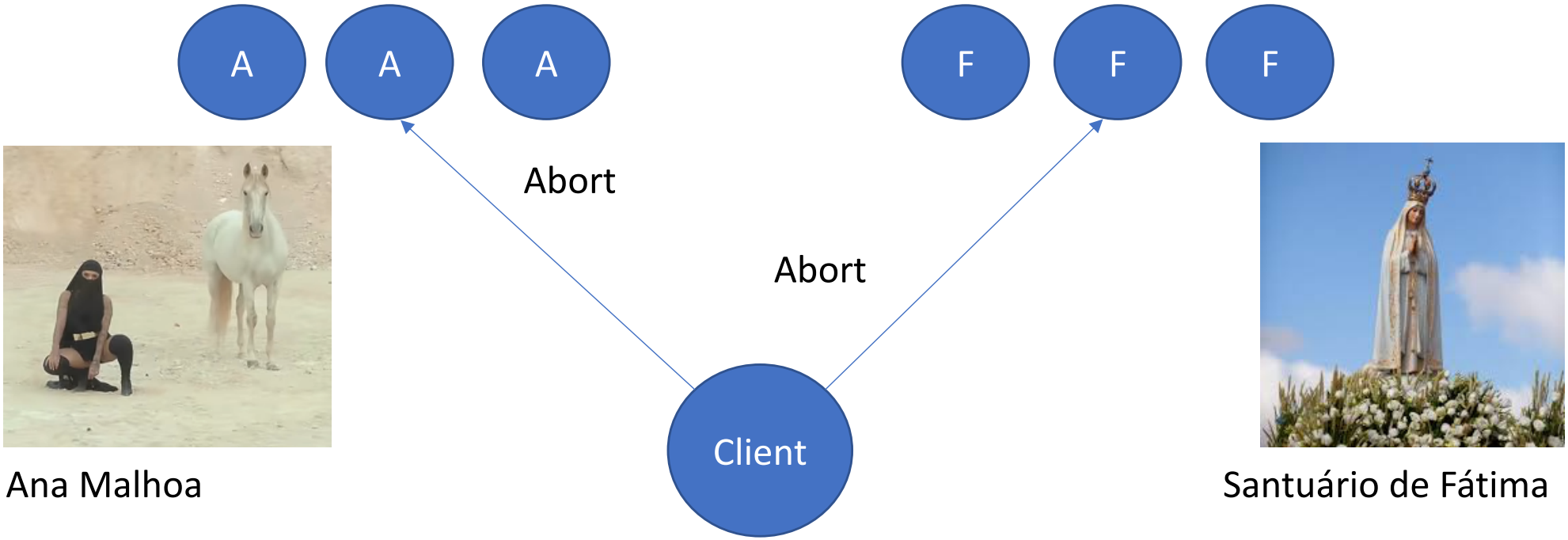
Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g, different Banks):



Distributed Transactions

- Going back to the Banking problem, imagine that the replicas hosting information for account A and account B are different (e.g, different Banks):



Distributed transactions

- Goal: preserve the properties of transactions (ACID) in a distributed scenario with failures
- Model: Distributed database
  - Transaction with several processes
  - One coordinator, other are participants
- Fault model: crash-recovery
- Processes have access to persistent storage (to write to a “log”)

Problem of “atomic commit” (AC): inputs and outputs

- Input of p_i: $vote_i$ in {yes,no}
- Output of p_i: $decision_i$ in {commit, abort}

- $vote_i$ is the vision of each process of whether the transaction respects ACID properties
  - With optimistic concurrency control, identifies if the transaction can be serialized in the order proposed by the coordinator → has read the correct version of data
  - With pessimistic concurrency control / locks, identifies whether all locks have been acquired

Specification of Atomic Commit

- **AC1 (agreement)**: Any two processes that decide, decide the same value
- **AC2 (validity, part 1)**: If some process starts with the value “no” then “abort” is the only possible decision
- **AC3 (validity, part 2)**: If all processes start with value “yes” and none fails, then “commit” is the only possible decision
- **AC4 (termination)**: If eventually all processes recover from all faults, then eventually all processes decide

Two-phase commit (2PC)

- Solves the problem of AC
- Assumes a synchronous system
  - In particular, timeout == crash (although a node can recover in the future)
  - In practice 2PC can be used in an asynchronous system but does not guarantee property AC3

- Note that AC is weaker than consensus because it can default to ABORT when suspects a failure + less strict termination property

One-phase commit

- Coordinator
  - Sends decide_i (commit or abort) to every participant
- Participant
  - If decide_i = commit, store result in stable memory
  - If decide_i = abort, discard intermediate results

- Which properties hold / do not hold?

One-phase commit

- Coordinator
  - Sends $decide_i$ (commit or abort) to every participant
- Participant
  - If $decide_i$ = commit, store result in stable memory
  - If $decide_i$ = abort, discard intermediate results

- Which properties hold / do not hold?
  - AC2 does not hold
  - Participants cannot influence the final result

2PC

Coordinator C

1. Sends vote-req to all participants

3. If all voted yes then
   decide$_c$ = COMMIT
   send COMMIT to all
   Else
   decide$_c$ = ABORT
   send ABORT to “yes” voters

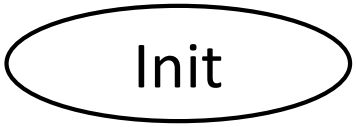
Participant pi

2. Send vote$_i$ to coordinator
   If (vote$_i$ == no) then
   decide$_i$ = abort;

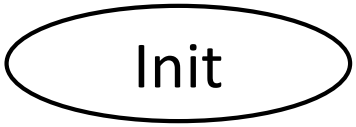
4. If receive COMMIT then
   decide$_i$ = COMMIT
   else
   decide$_i$ = ABORT

2PC – diagrama de estados

- Coordinator

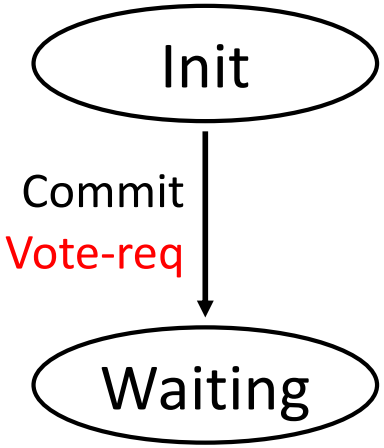


- Participant

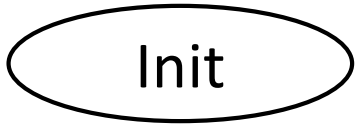


2PC – diagrama de estados

- Coordinator

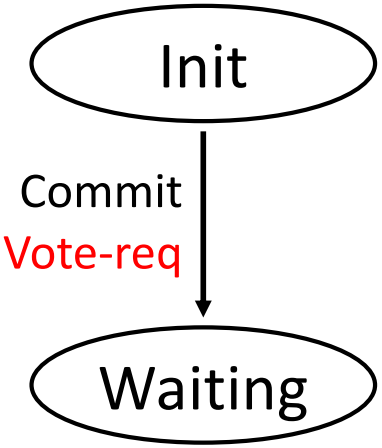


- Participant

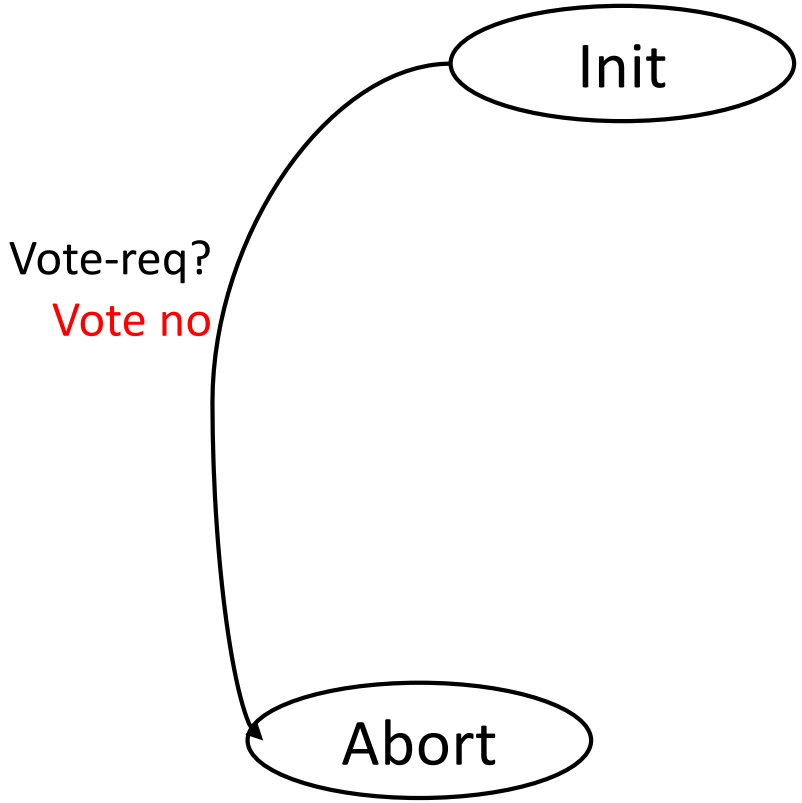


2PC – diagrama de estados

- Coordinator

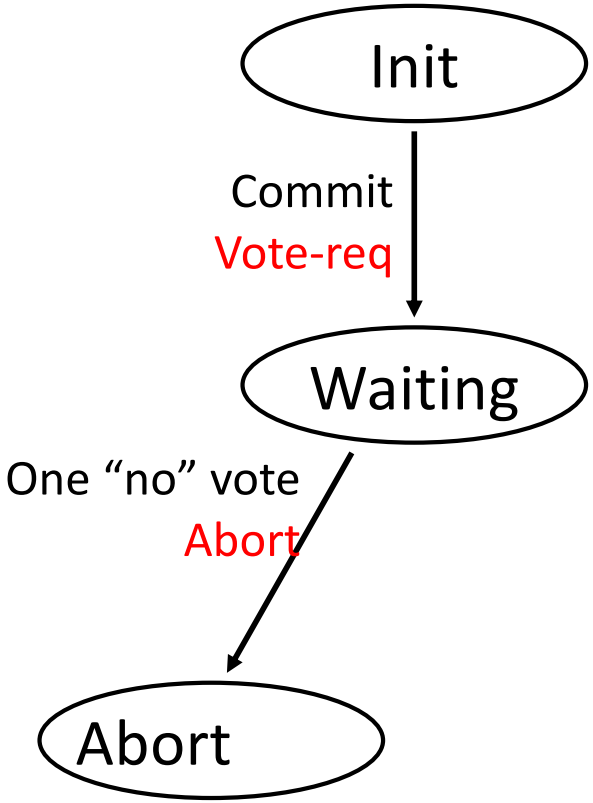


- Participant

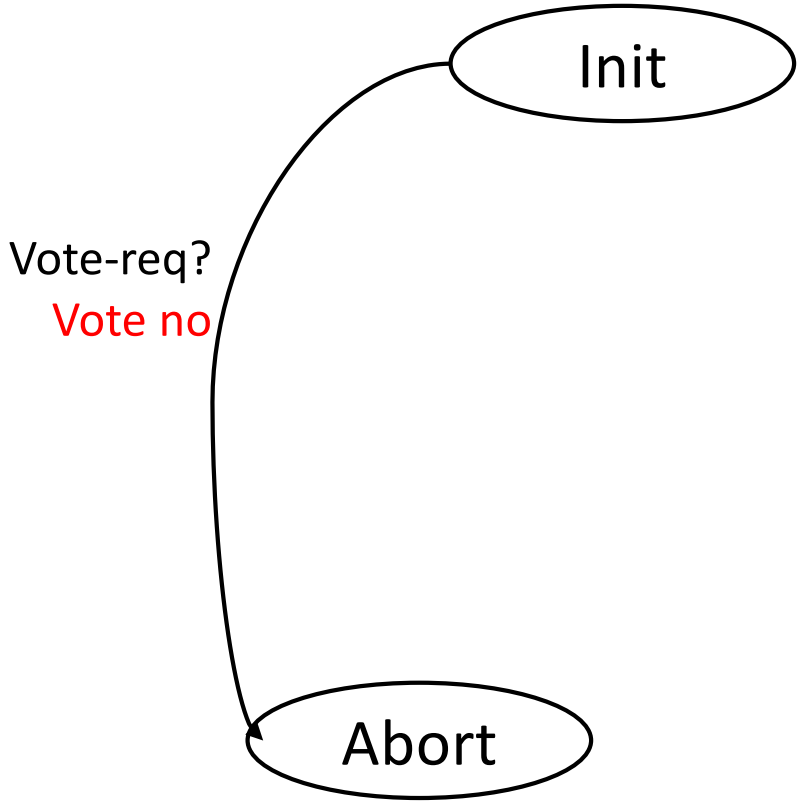


2PC – diagrama de estados

- Coordinator

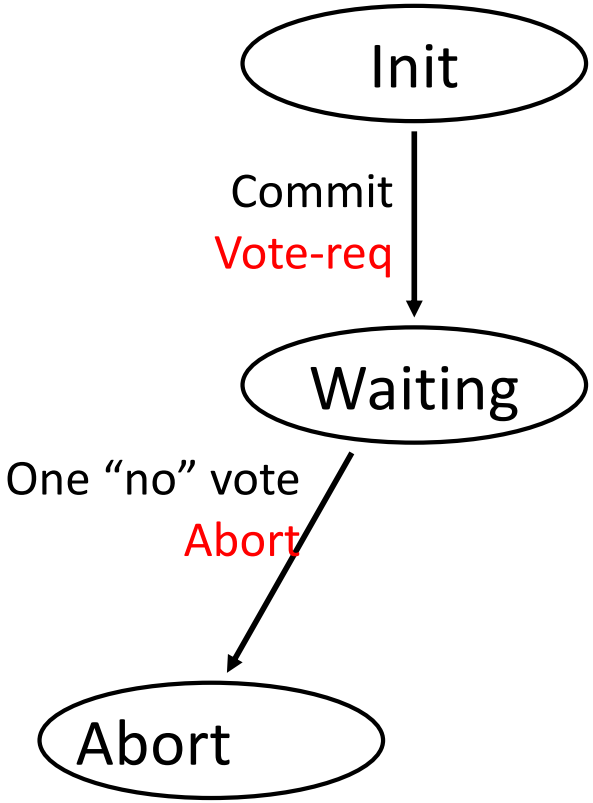


- Participant

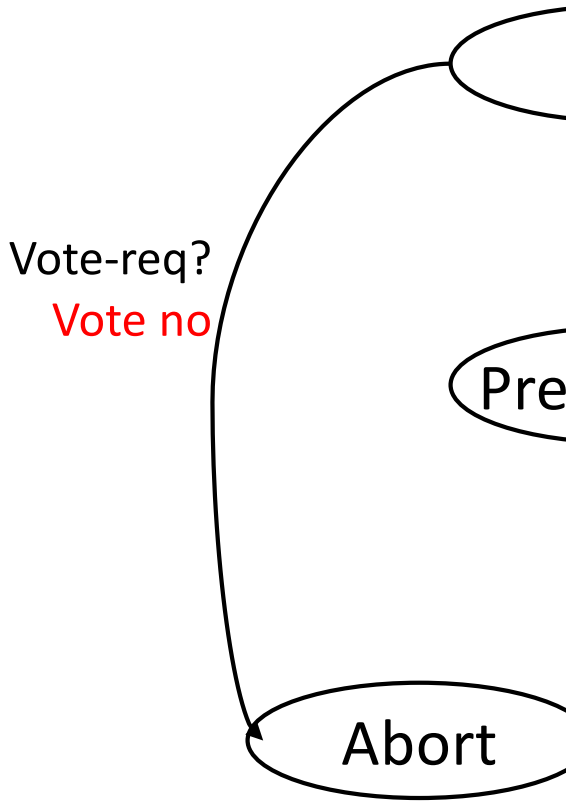


2PC – diagrama de estados

- Coordinator

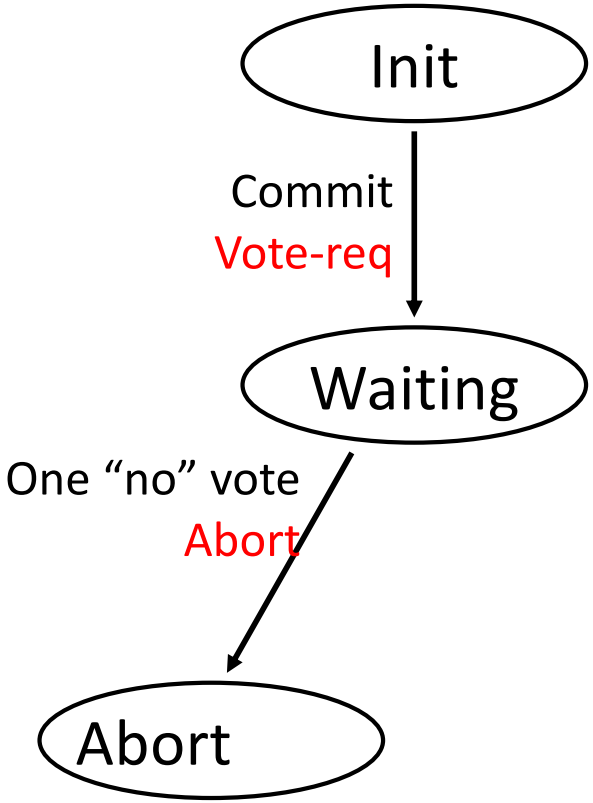


- Participant

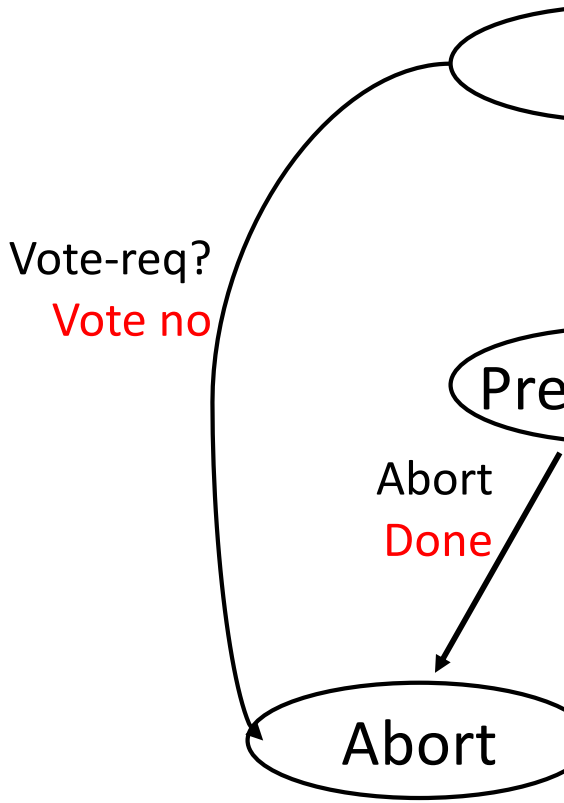


2PC – diagrama de estados

- Coordinator

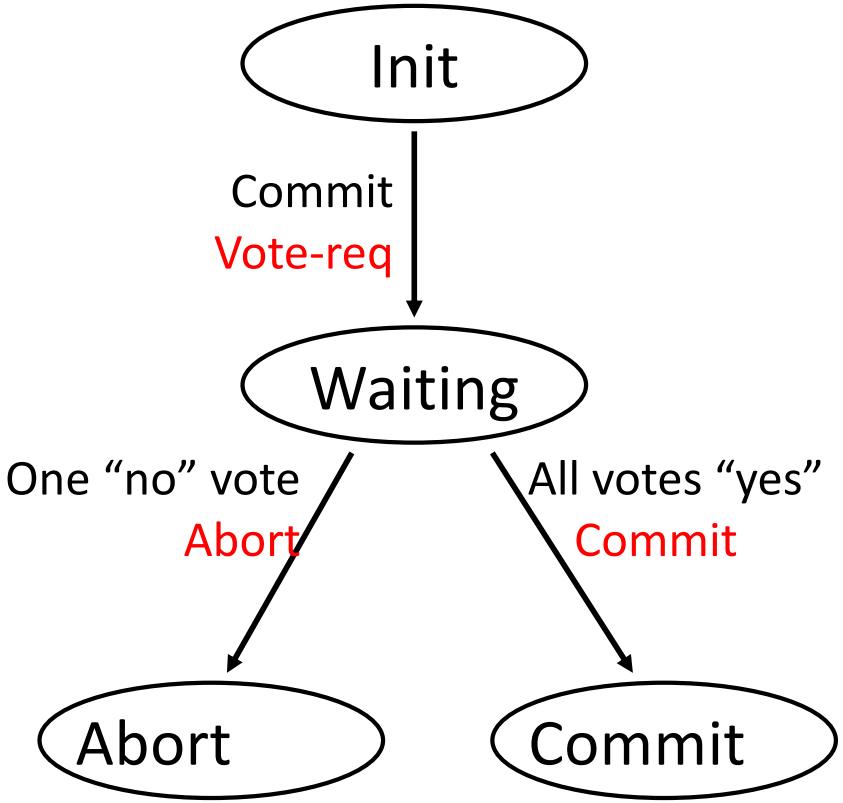


- Participant

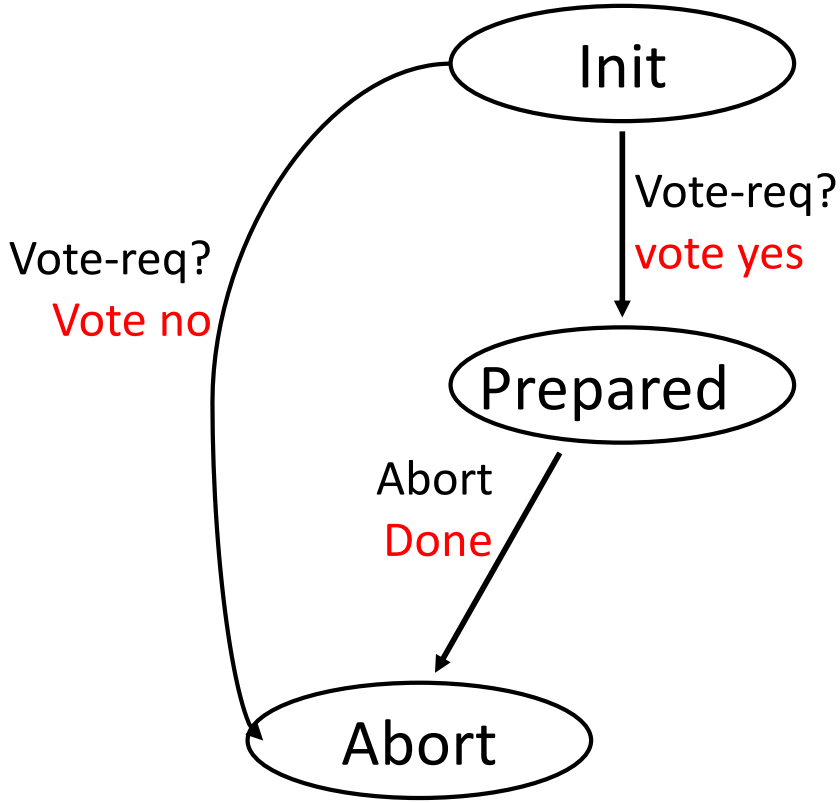


2PC – diagrama de estados

- Coordinator

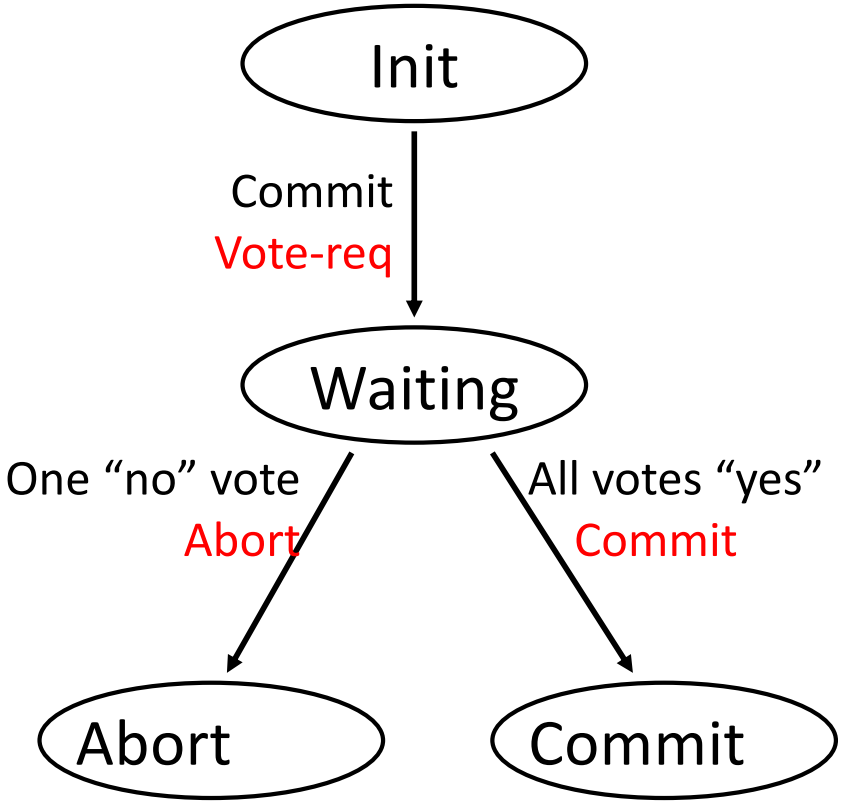


- Participant

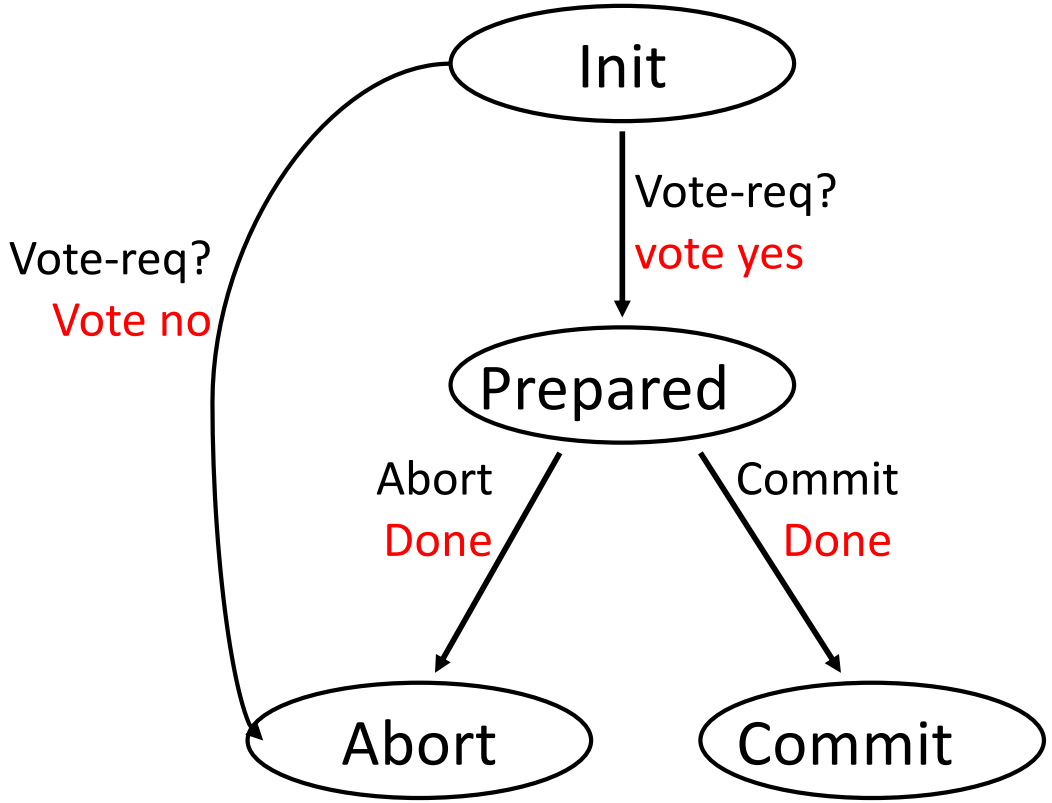


2PC – diagrama de estados

- Coordinator



- Participant



Properties that hold?

AC1 (agreement): Any two processes that decide, decide the same value

AC2 (validity, part 1): If some process starts with the value “no” then “abort” is the only possible decision

AC3 (validity, part 2): If all processes start with value “yes” and none fails, then “commit” is the only possible decision

AC4 (termination): If eventually all processes recover from all faults, then eventually all processes decide

- All properties from AC1-3
- Why is it that there is a problem with AC4?

Properties that hold?

AC1 (agreement): Any two processes that decide, decide the same value

AC2 (validity, part 1): If some process starts with the value “no” then “abort” is the only possible decision

AC3 (validity, part 2): If all processes start with value “yes” and none fails, then “commit” is the only possible decision

AC4 (termination): If eventually all processes recover from all faults, then eventually all processes decide

- All properties from AC1-3
- Why is it that there is a problem with AC4?
- If a process crashes, when it recovers it forgets if it had sent a vote before… consequently, all processes might be waiting for ever and never terminate.

Properties that hold?

AC1 (agreement): Any two processes that decide, decide the same value

AC2 (validity, part 1): If some process starts with the value “no” then “abort” is the only possible decision

AC3 (validity, part 2): If all processes start with value “yes” and none fails, then “commit” is the only possible decision

AC4 (termination): If eventually all processes recover from all faults, then eventually all processes decide

- All properties from AC1-3
- Why is it that there is a problem with AC4?
- If a process crashes, when it recovers it forgets if it had sent a vote before… consequently, all processes might be waiting for ever and never terminate.
  - One needs to have a log of all previous actions in stable storage.
  - One can use a timeout to avoid waiting for messages that will never arrive.

What to do in the event of a timeout:

- Processes have to wait in steps: 2,3,4
- Step 2: pi waits for vote-req from the coordinator
- Step 3: coordinator waits from votes from all participants
- Step 4: pi (that voted yes) waits for final decision

What to do in the event of a timeout:

- Processes have to wait in steps: 2,3,4
- Step 2: pi waits for vote-req from the coordinator
  - Pi can unilaterally decide abort.
- Step 3: coordinator waits from votes from all participants
- Step 4: pi (that voted yes) waits for final decision

What to do in the event of a timeout:

- Processes have to wait in steps: 2,3,4
- Step 2: pi waits for vote-req from the coordinator
  - Pi can unilaterally decide abort.
- Step 3: coordinator waits from votes from all participants
  - Coordinator can decide ABORT and send it to all processes.
- Step 4: pi (that voted yes) waits for final decision
  - Cannot make a decision and must execute a special protocol to be able to terminate.

Termination Protocol

- A possibility is to wait for the coordinator to recover:
  - This works since AC4 only requires termination if all processes that fail eventually recover.
  - But this might block the recovery mechanism for a long time in an unnecessary fashion.

Termination Protocol

- A possibility is to wait for the coordinator to recover:
  - This works since AC4 only requires termination if all processes that fail eventually recover.
  - But this might block the recovery mechanism for a long time in an unnecessary fashion.
- Is there another way?

Termination Protocol

- A possibility is to wait for the coordinator to recover:
  - This works since AC4 only requires termination if all processes that fail eventually recover.
  - But this might block the recovery mechanism for a long time in an unnecessary fashion.
- Ask the other participants what happened…
  - Processes that have already decided send the decision, processes that never voted can decide ABORT and sent it to all other processes.
  - What if all processes are waiting for a decision of the coordinator?

Why stable logging is required:

- In the worst case, processes might be required to wait for the coordinator to recover and provide the decision that he had taken before.

- This requires the coordinator of the transaction to maintain information in log about any decision that it takes.

Logging

- When sending VOTE-REQ, coord. logs START-2PC.
- Before sending YES, p_i writes YES to the log.
- When sending NO, p_i writes ABORT to the log (can be done after sending the message)
- Upon deciding COMMIT e sending the decision to all participants, the coordinator writes COMMIT in the log
- Upon deciding ABORT, coordinator write ABORT to the log (can be done after sending the message)
- Upon receiving a decision, p_i writes it to the log

Recovery Protocol

- Upon recovery, if the log contains START-2PC:
  - And if the log contains COMMIT decide COMMIT
  - Otherwise decide ABORT
- If the START-2PC is not in the log, the process is only a participant:
  - If log contains decision, decide that value.
  - Otherwise, if log does not contain YES vote, decide ABORT.
  - Otherwise, execute the termination protocol (ask all processes for a decision, and wait either from an ABORT from a participant, or a definite answer from the coordinator)

Problems with *two-phase commit*

- Protocol is blocking
  - Participants need to wait for the recovery of the coordinator (in prepared state)
  - Requires coordinator to wait for the recovery of participants (in commit state)
- Recovery is not independent...

Three-phase commit

- Goal: Turn 2PC into a live (non-blocking) protocol
  - 3PC should never block on node failures as 2PC did

Three-phase commit

- Goal: Turn 2PC into a live (non-blocking) protocol
  - 3PC should never block on node failures as 2PC did

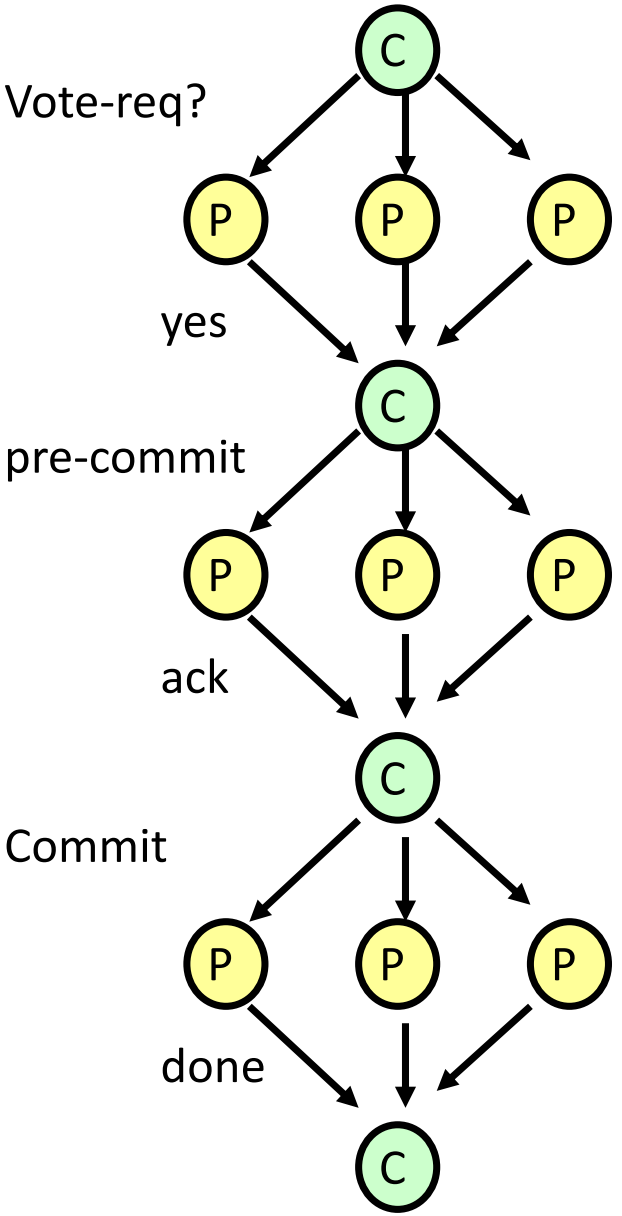
- Insight: 2PC allows nodes to irreversibly commit an outcome before ensuring that the others know the outcome.

Three-phase commit

- Goal: Turn 2PC into a live (non-blocking) protocol
  - 3PC should never block on node failures as 2PC did
- Insight: 2PC allows nodes to irreversibly commit an outcome before ensuring that the others know the outcome.
- Idea in 3PC: split “commit/abort” phase into two phases
  - First communicate the outcome to everyone
  - Let them commit only after everyone knows the outcome

Three-phase commit Protocol

- Phase 1 (similar to *two-phase commit*)
  - Coordinator: sends *vote-req*
  - Participants: vote YES/NO
- Phase 2
  - Coordinator: gathers votes and:
    - Sends *pre-commit* to all participants, if decision is commit.
    - Sends *Abort* to all participants that voted yes if decision is ABORT.
  - Participants: execute Abort or send *acknowledge* of *pre-commit*
- Phase 3
  - Coordinator: gathers *acks* and sends *Commit*
  - Participants: execute *Commit*



Can 3PC solve blocking 2PC examples?

- 2PC: Participants need to wait for the recovery of the coordinator (in prepared state, i.e., after voting)
- If one participant received pre-commit, ...
- If no participant received pre-commit, ...

Can 3PC solve blocking 2PC examples?

- 2PC: Participants need to wait for the recovery of the coordinator (in prepared state, i.e., after voting)

- If one participant received pre-commit, all can commit
  - Safe if participants run recovery protocol

- If no participant received pre-commit, all can abort
  - Safe because it is known that no participant has received Commit

Three-phase commit

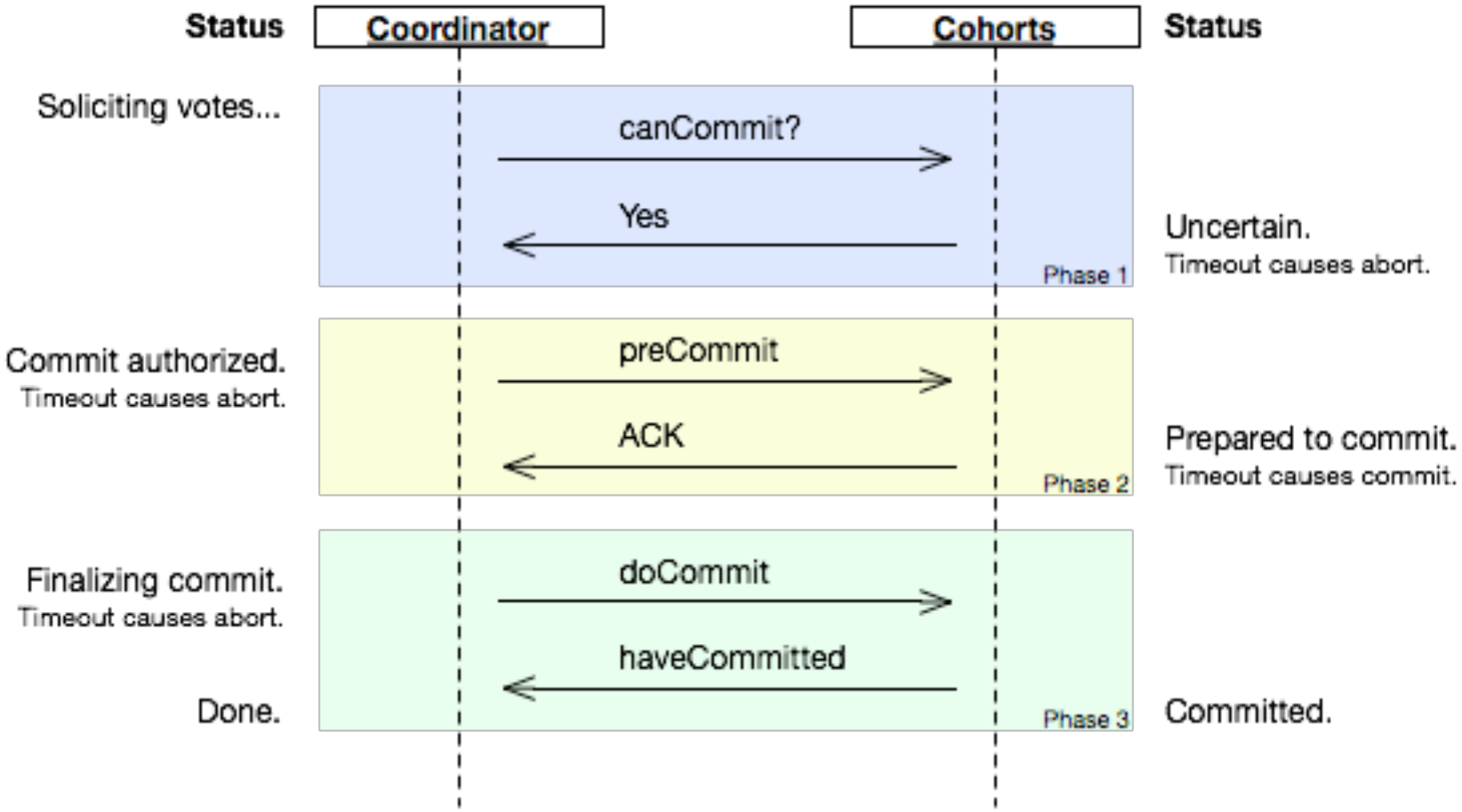


Image source: wikipedia

Does 3PC achieve:

- Liveness (availability):
- Safety (correctness):

Does 3PC achieve:

- Liveness (availability): Yes
  - Doesn’t block, it always makes progress by timing out
- Safety (correctness):

Does 3PC achieve:

- Liveness (availability): Yes
  - Doesn’t block, it always makes progress by timing out
- Safety (correctness): No
  - Can you think of scenarios in which original 3PC would result in inconsistent states between the replicas?

Does 3PC achieve

- Liveness (availability): Yes
  - Doesn’t block, it always makes progress by timing out
- Safety (correctness): No
  - Can you think of scenarios in which original 3PC would result in inconsistent states between the replicas?
  - What happens if Coordinator fails and one participant gets partitioned from other after being the only one to receive pre-commit?

Safety vs. Liveness

- 3PC is not safe under network partitions
  - Or it needs to block

- The way to think about it is that this protocol's design trades safety for liveness
  - 2PC trades liveness for safety

ASD: Finished

- Discussions: You will receive an e-mail with the instructions to register your group for discussions (Discussion are mandatory). **Week of 15 December.**

ASD: Finished

- Discussions: You will receive an e-mail with the instructions to register your group for discussions (Discussion are mandatory). **Week of 15 December.**
- Test: 10th (Tuesday) at 20:30h in room 128 (Ed II).

ASD: Finished

- Discussions: You will receive an e-mail with the instructions to register your group for discussions (Discussion are mandatory). **Week of 15 December.**
- Test: 10th (Tuesday) at 20:30h in room 128 (Ed II).
- Format of the Test: Similar to the 1st Test, materials Covered are: State Machine Replication, Paxos/Multi-Paxos (and hence Consensus), Replication, Strong and Weak Consistency, Causal Consistency, ABD Quorum System, Chain Replication, Two Phase Commit and Three Phase Commit (there is 1 option groups)