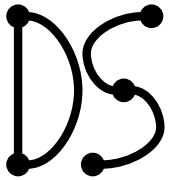


Research Group
Distributed Systems

Distributed Systems

Consistency & Replication III

Olaf Landsiedel

Last Time

- Client Centric Consistency
  - Eventual Consistency
  - Client Consistency Guarantees
- Replica Management
- Content Delivery Networks

Today…

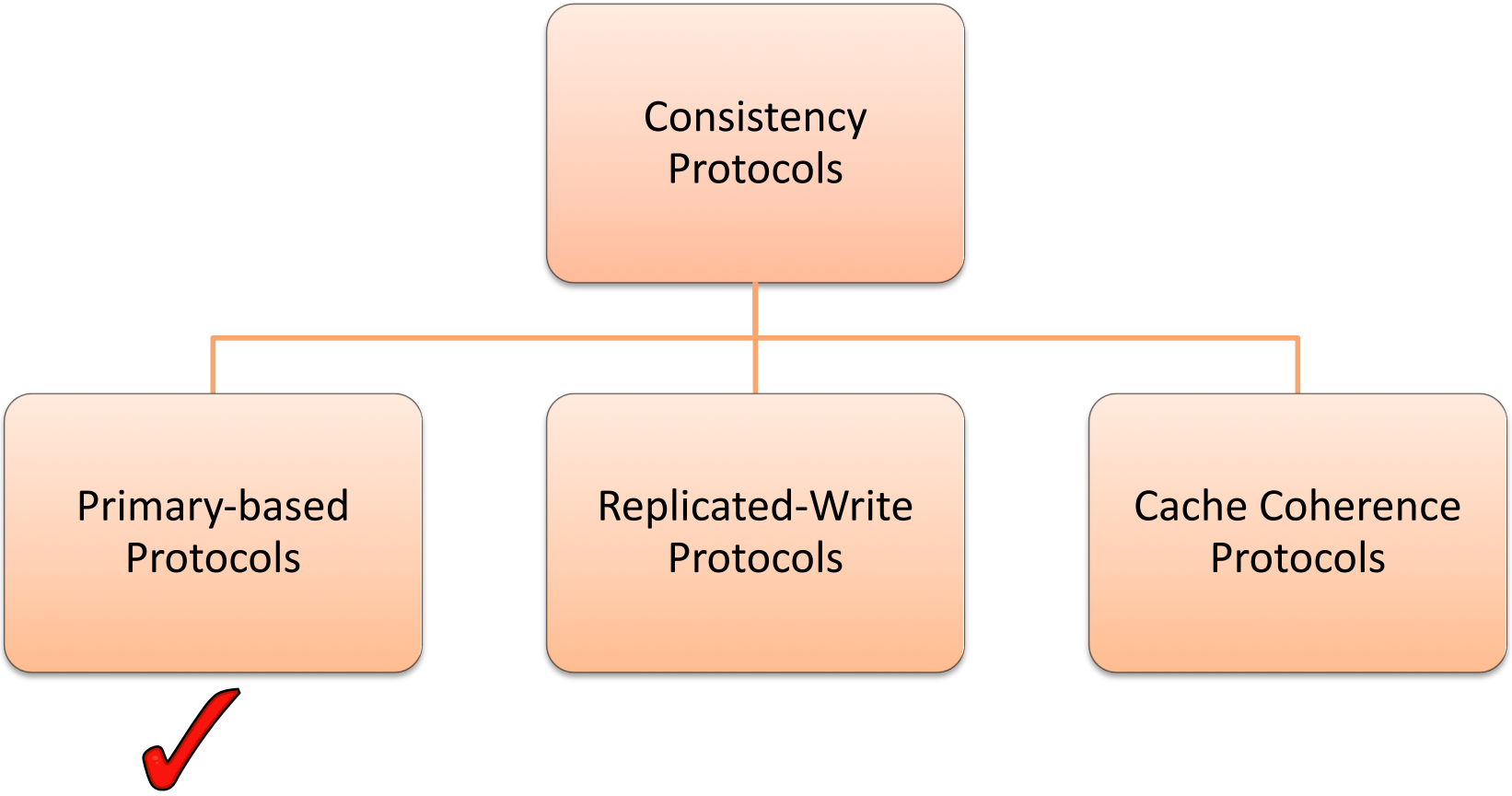
- Consistency and Replication III
  - Consistency protocols
  - Case Study: Protocol for eventual consistency

CONSISTENCY PROTOCOLS

Consistency Protocols

- A consistency protocol describes the implementation of a specific consistency model
- We are going to study three types of consistency protocols:
  - Primary-based protocols
    - One primary coordinator is elected to control replication across multiple replicas
  - Replicated-write protocols
    - Multiple replicas coordinate to provide consistency guarantees
  - Cache-coherence protocols
    - A special case of client-controlled replication

Overview of Consistency Protocols

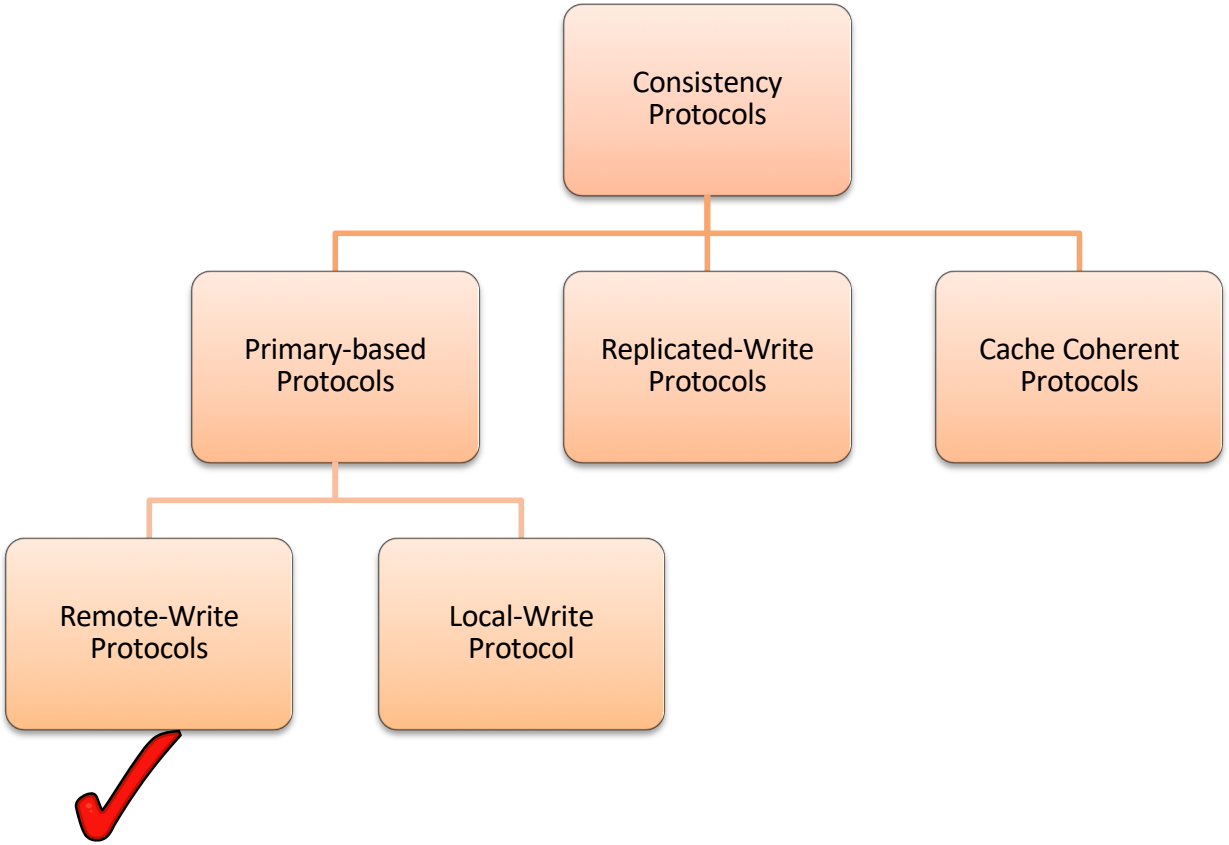


Primary-based protocols

- In Primary-based protocols, a simple centralized design is used to implement consistency models
  - Each data-item **x** has an associated "*Primary Replica*"
  - Primary replica is responsible for coordinating write operations

- We will study two examples of Primary-based protocols that implement the Sequential Consistency Model
  - Remote-Write Protocol
  - Local-Write Protocol

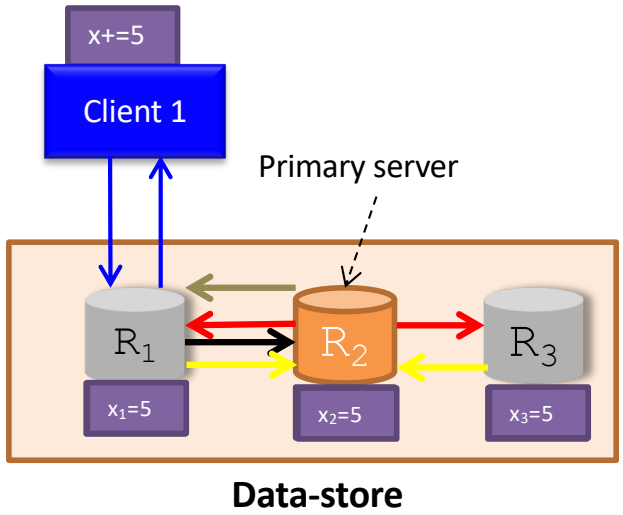
Overview of Consistency Protocols



Remote-Write Protocol

- Rules:
  - All write operations are forwarded to the primary replica
  - Read operations are carried out locally at each replica

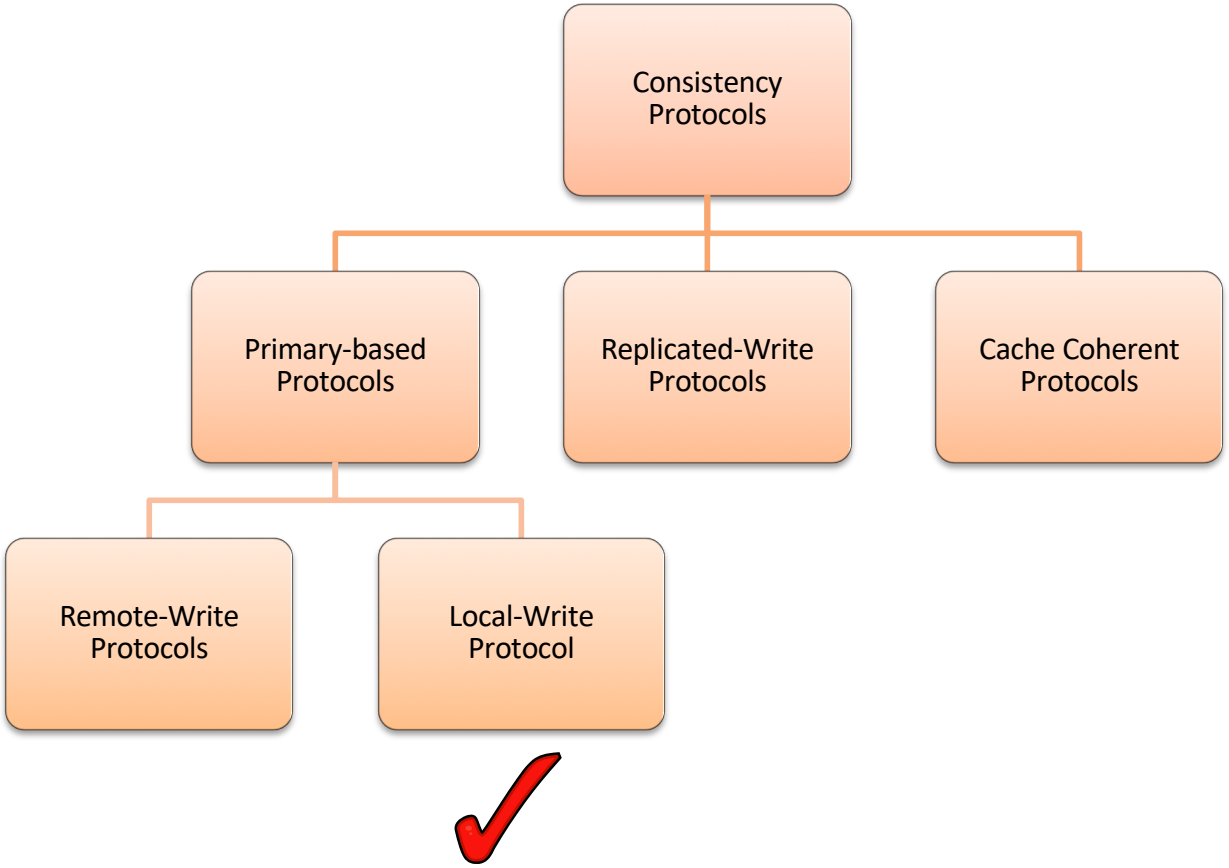
- Approach for write ops
  - Client connects to some replica R_C
  - If the client issues write operation to R_C:
    - R_C forwards the request to the primary replica R_P
    - R_P updates its local value
    - R_P forwards the update to other replicas R_i
    - Other replicas R_i update, and send an ACK back to R_P
  - After R_P receives all ACKs, it informs the R_C that write operation is successful
  - R_C acknowledges to the client that write operation was successful



Remote-Write Protocol – Discussion

- Remote-Write provides
  - A simple way to implement sequential consistency
  - Guarantees that client see the most recent write operations
- However,
  - Latency is high in Remote-Write Protocols
    - Client blocks until all the replicas are updated
- Remote-Write Protocols are applied to distributed databases and file systems that require fault-tolerance
  - Replicas are placed on the same LAN to reduce latency

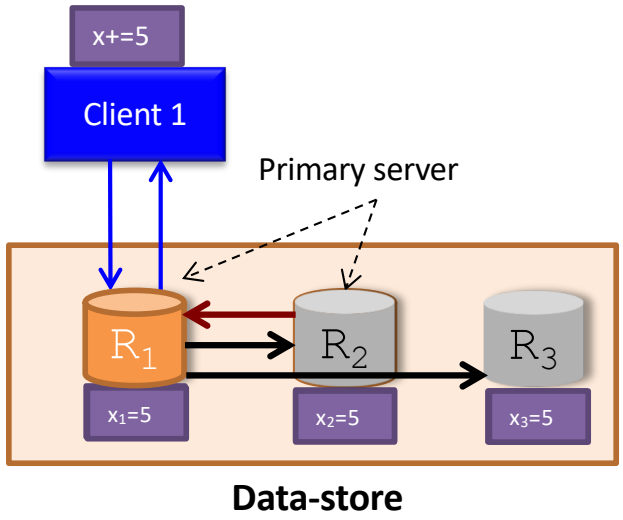
Overview of Consistency Protocols



Local-Write Protocols

- Can we make Remote-Write better for applications that require client-centric consistency model?
  - Single client process issues write operations
  - Reasonably stale value of data is OK when other processes read

- Approach:
  - + Client connects to some replica $\mathbf{R_C}$
  - + If the client issues write op to $\mathbf{R_C}$:
    - + $\mathbf{R_C}$ becomes the primary replica $\mathbf{R_P}$
    - + Rest of the protocol is similar to Remote-Write



Local-Write Protocol

- Scenarios where Local-Write is applicable:
  - When doing multiple consecutive writes
    - Reduces overhead compared to remote write protocols
    - The primary replica is cached at the client
- Scenarios where Local-Write is inappropriate:
  - When (multiple) clients are writing at multiple replicas
    - Overhead of reassigning primary replica is high

Questions

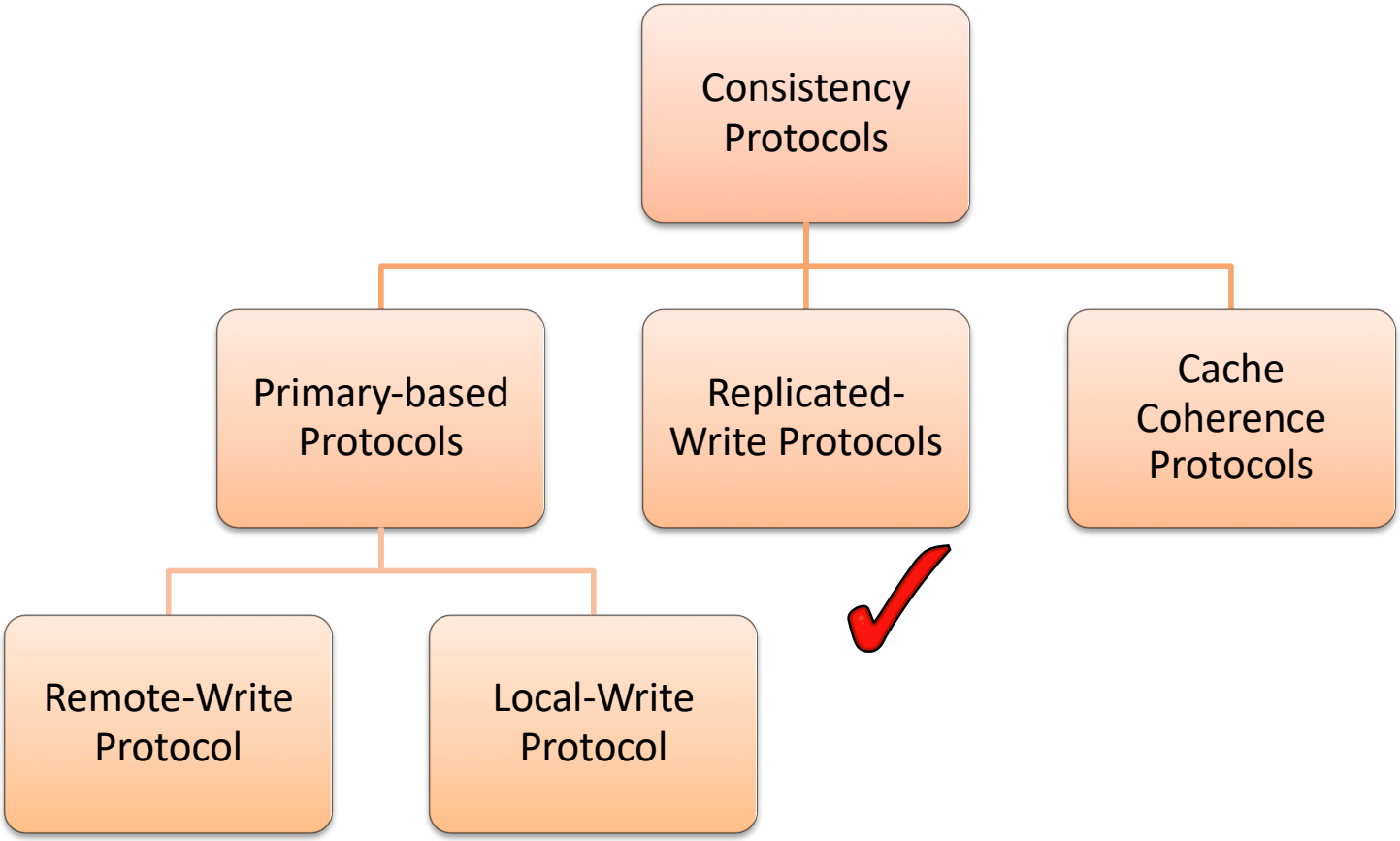
- Remote-Write Protocol?
- Local-Write Protocol?

- Questions
  - Which one is good for a single writing client?
  - Which one is good for multiple writing clients?
  - Which one to use for a flight booking service?
  - Which one to use for a Dropbox-like service?
- olafland.polldaddy.com/s/remote-write-vs-local-write-protocol

Answers

- Which one is good for a single writing client:
  - local write protocol (see prev. slides)
- Which one is good for multiple writing clients
  - remote write protocol (see prev. slides)
- Which one to use for a flight booking service: expect different clients to book flights (and not one client many many flights)
  - remote write protocol
- Which one to use for a Dropbox-like service:
  - expect most writes to from me to my files
  - -> one client, many writes
  - -> local write protocol
  - Exception: shared files:
    - many writing clients -> local write not good here

Overview of Consistency Protocols



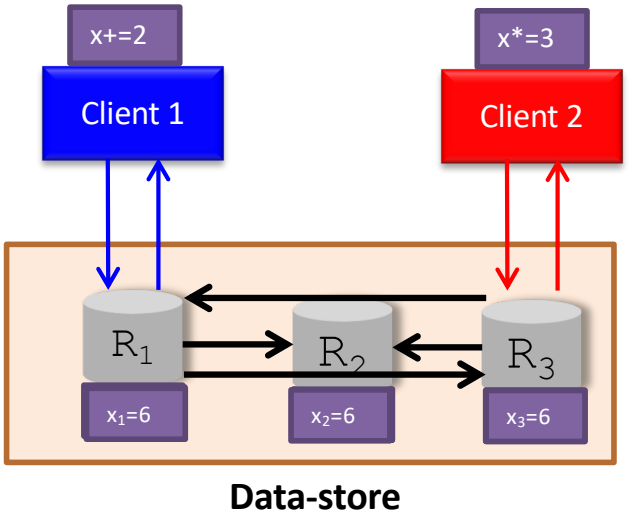
Replicated-Write Protocol

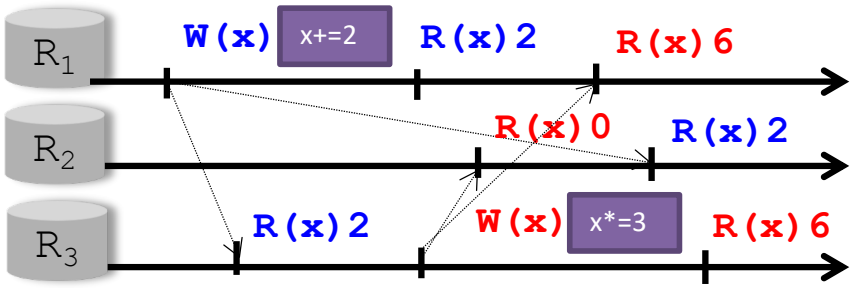
- In a replicated-write protocol, updates can be carried out at multiple replicas

- We study two replicated-write protocols
  - **Active Replication Protocol**
    - Here, clients write at any replica
    - The replica will propagate updates to other replicas
  - **Quorum-Based Protocols**
    - Here, clients read/write at multiple replicas
    - Majority vote

Active Replication Protocol

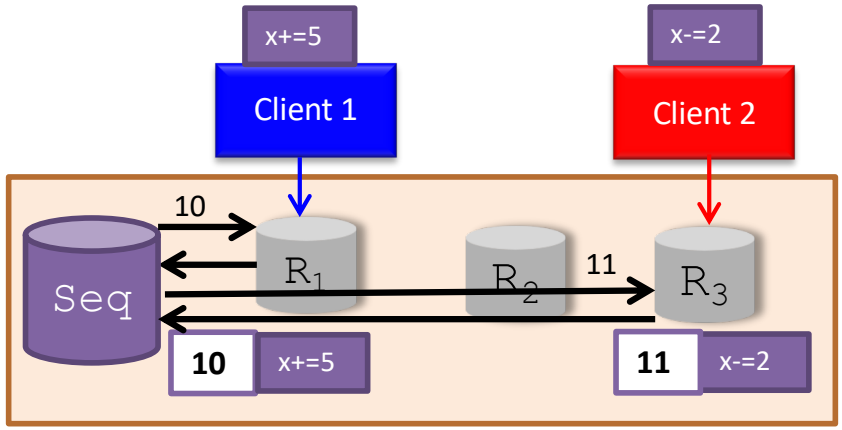
- When a client writes at a replica, the replica will send the write operation updates to all other replicas
- Challenges with Active Replication
  - Problems?
  - Ordering of operations cannot be guaranteed across the replicas
  - How to fix?



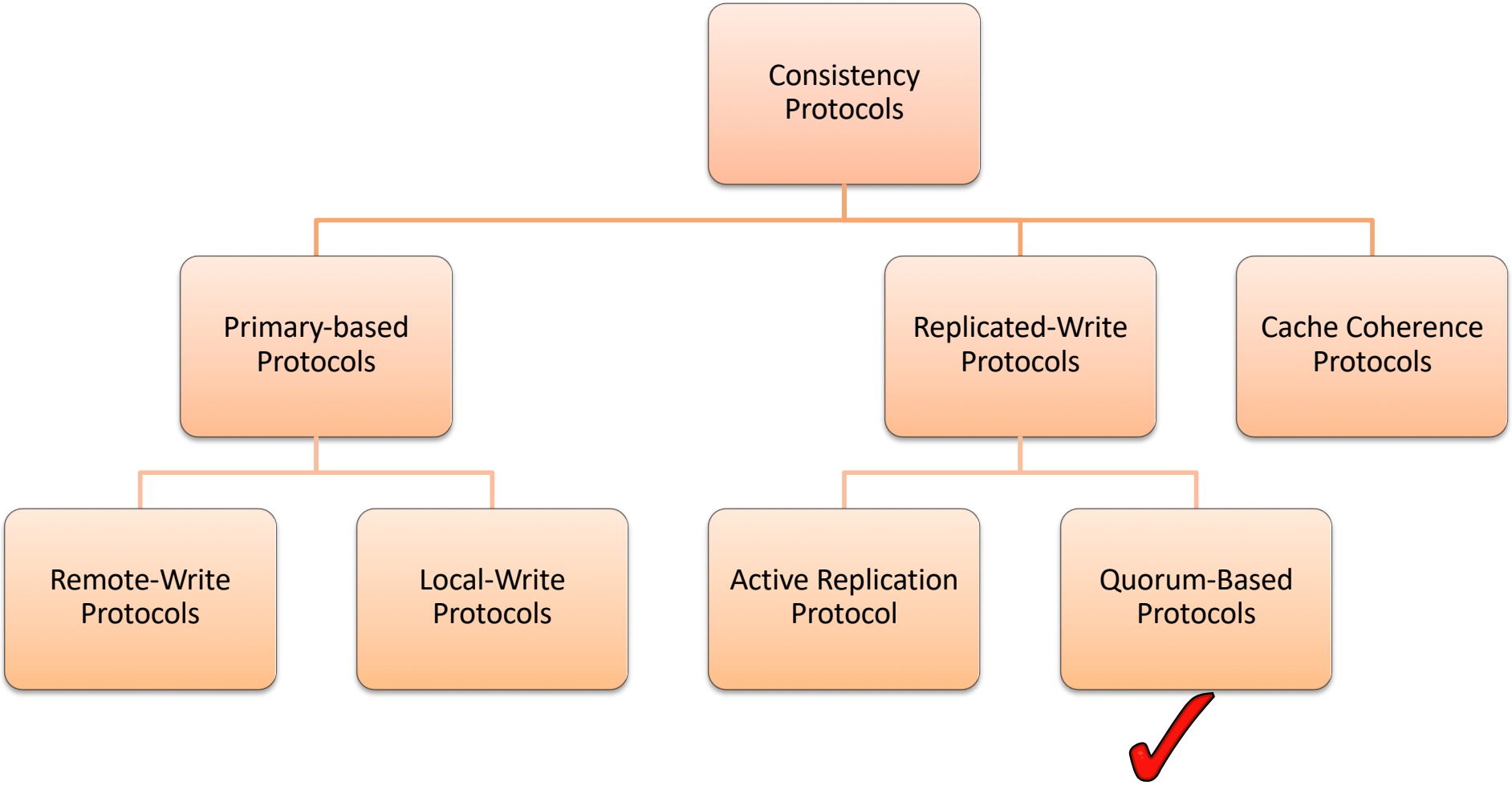


Centralized Active Replication Protocol

- Approach
  - There is a centralized coordinator called *sequencer* (**Seq**)
  - When a client connects to a replica $\mathbf{R_C}$ and issues a write operation
    - $\mathbf{R_C}$ forwards the update to the **Seq**
    - **Seq** assigns a sequence number to the update operation
    - $\mathbf{R_C}$ propagates the sequence number and the operation to other replicas
  - Operations are carried out at all the replicas in the order of the sequence number

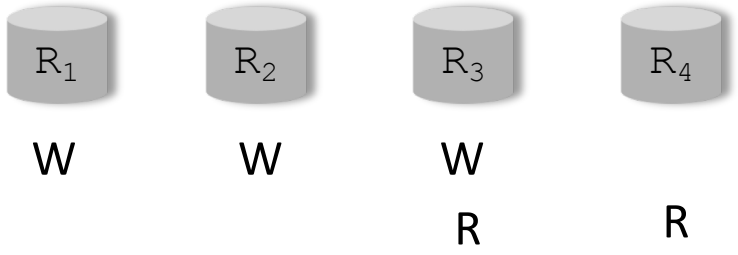


Overview of Consistency Protocols



Quorum-Based Protocols

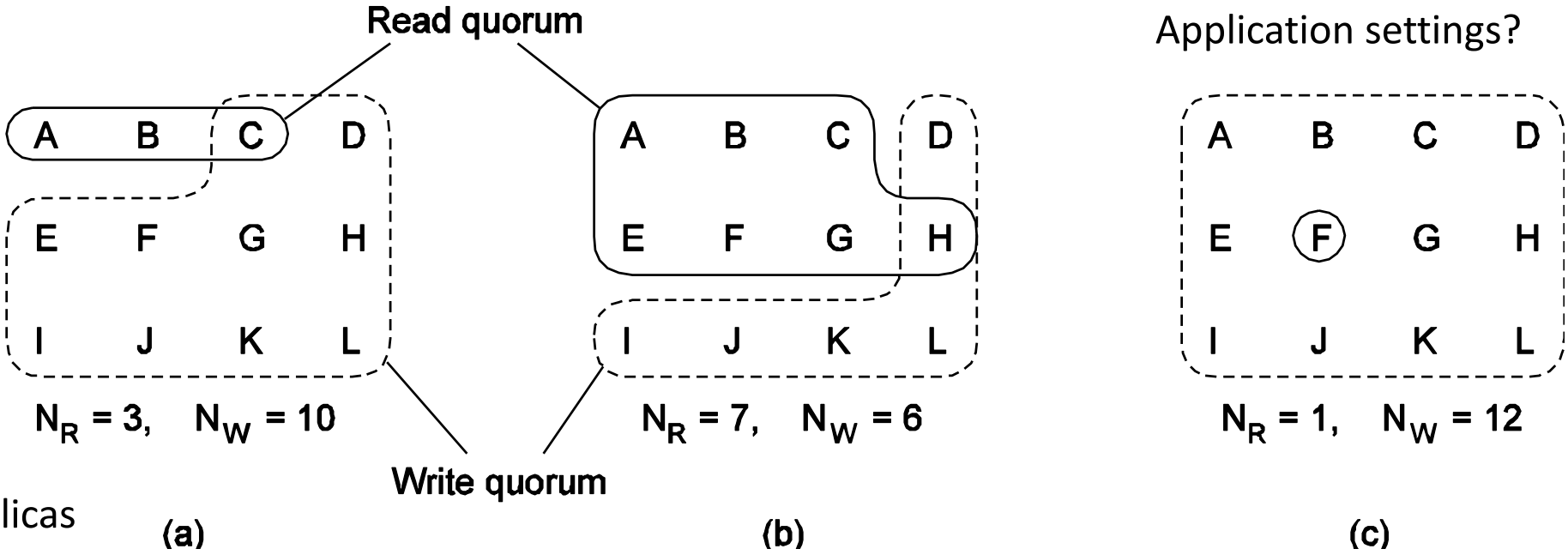
- Read / write at multiple replicas
- How to get majority vote?
  - To how many replicas shall I write to write to the majority?
  - From how many replica shall I read to catch the latest write at least once?
- Example: 4 replicas
  - Write to 3, read from 2
  - Sure to read latest write at least once



Quorum-Based Protocols

- Quorum-based protocols:
  - Need to read / write to many machines
  - Each write has a logical timestamp (version number)
- Read quorum (N_R) and write quorum (N_W), N replicas:
  - $N_R + N_W > N$
  - $N_W > N/2$ (see next slide)

ROWA: read one, write all
Application settings?



12 replicas

Quorum-Based Protocols

- Why: $N_R + N_W > N$
  - ensures a data item is not read and written by two transactions concurrently.
  - ensures that a read quorum contains at least one site with the newest version of the data item.
- Why: $N_W > N/2$
  - ensures that two write operations from two transactions cannot occur concurrently on the same data item.
- Together, the two rules ensure that one-copy serializability is maintained.

Quorum-Based Protocols

- Discussion
- Pros?
  - Fully distributed protocol, no single point failure
  - Can choose size of read and write quorums based on application requirements
    - Example high read to write ratio: ROWA
- Cons?
  - Complex to implement
  - Overhead and complexity: impacts performance

Questions

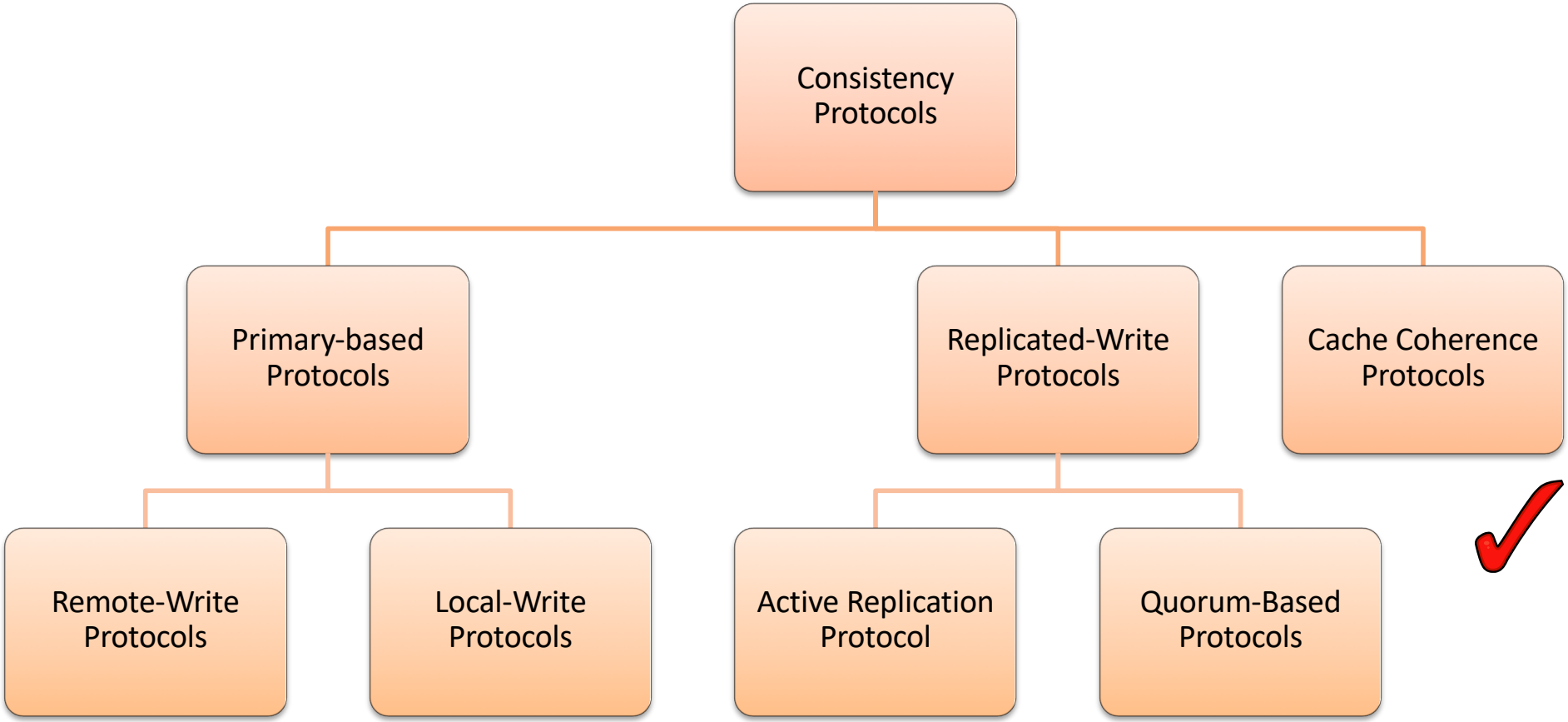
- Active Replication?
- Quorum Based?

- Questions
  - Which one has no single point of failure?
  - Which one is easier to implement?
  - Which one allows more reads per second?
  - Which one allows more writes per second?
- olafland.polldaddy.com/s/active-replication-vs-quorum-based

Answers

- Which one has no single point of failure:
  - quorum (and also active replication if we add re-election of sequence)
- Which one is easier to implement:
  - active replication: less complexity
- Which one allows more reads per second:
  - active: can read from any replica (when configured for ROWA, quorum can give similar performance)
- Which one allows more writes per second:
  - quorum: have to write to write quorum size, commonly less than all replicas to which I have to write when using active replication

Overview of Consistency Protocols



Cache Coherence Protocols

- Caches are special types of replicas
  - Typically, caches are client-controlled replicas
- Cache coherence refers to the consistency of data stored in caches
- How are the cache coherence protocols in shared-memory multiprocessor (SMP) systems different from those in Distributed Systems (DS)?
  - Coherence protocols in SMP assume cache states can be broadcasted efficiently
  - In DS, this is not possible because caches may reside on different machines

Cache Coherence Protocols (cont'd)

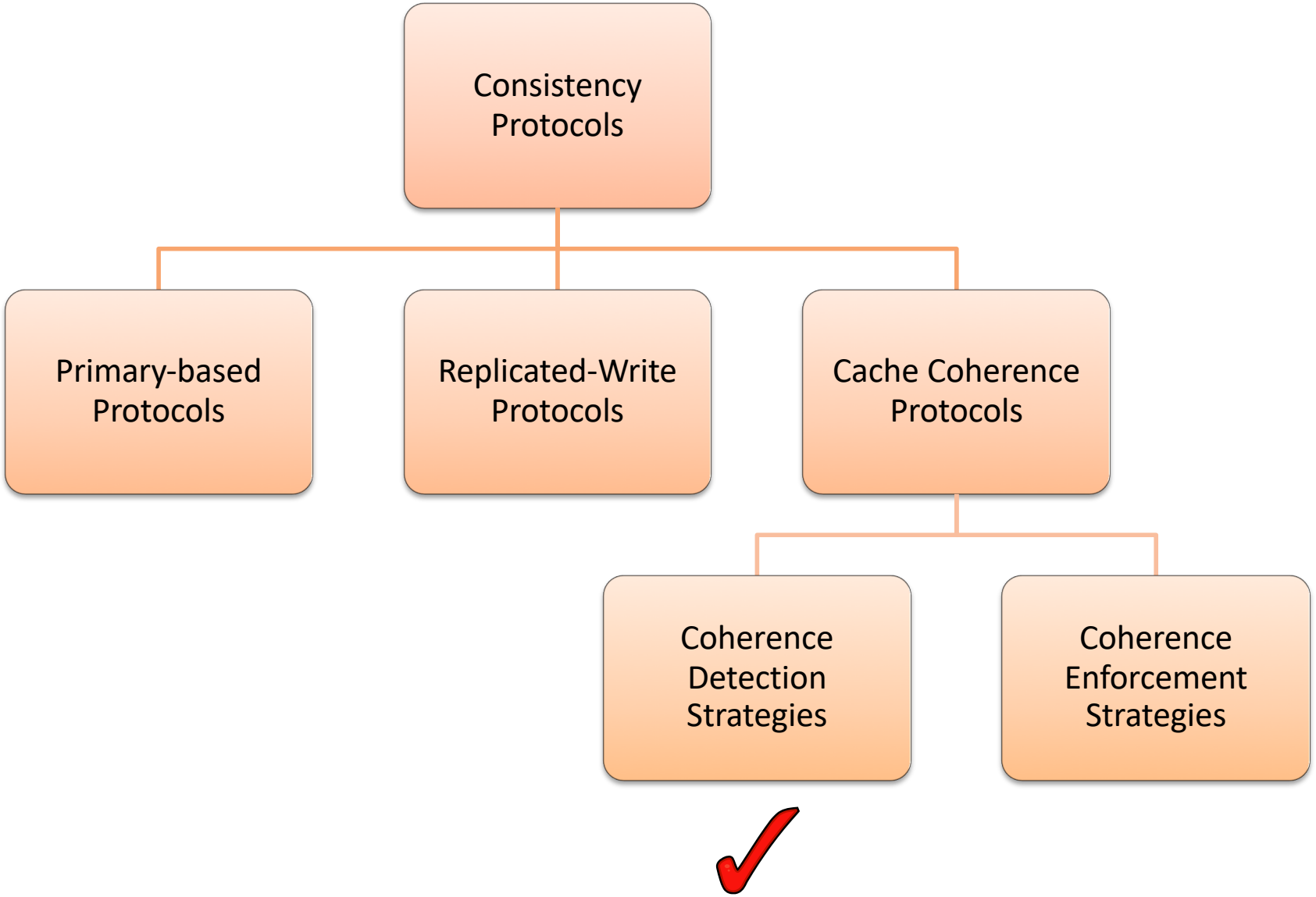
- Cache Coherence protocols determine how caches are kept consistent

- Caches may become inconsistent when data item is modified:
  1. at the server replicas, or
  2. at the cache

Two aspects of Cache Coherence Protocols

- In order to maintain consistent caches, we need to perform two operations:
  - Coherence detection strategies
    - Detect inconsistent caches
  - Coherence enforcement strategies
    - Update caches

Overview of Consistency Protocols



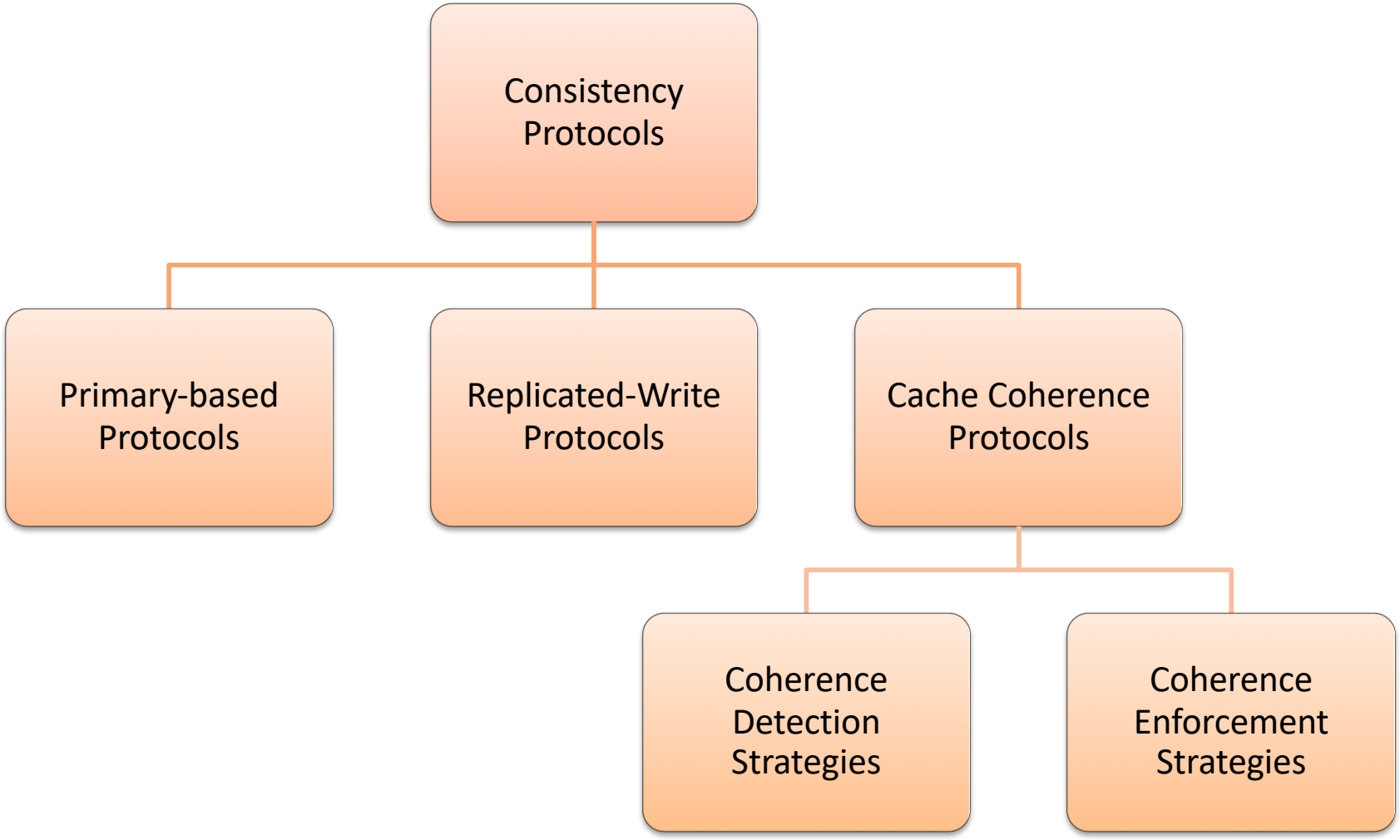
Coherence Detection Strategies

- Detection strategies deal with predicting when caches are inconsistent
- Since different replicas may be written by client processes, the protocol has to dynamically detect cache inconsistency

Coherence Detection Strategies

- In a distributed system, cache inconsistencies can be typically detected at three stages:
  1. Verify coherence before every access
     - Before every read or write operation
  2. Verify coherence before a write access
     - Cache coherence is checked before every write operation
  3. Verify coherence after a write access
     - First, an update is performed, and later cache consistency is verified
     - If cache was inconsistent
       - The write operation is rolled-back, and re-performed

Overview of Consistency Protocols



Coherence Enforcement Strategies

- Enforcement strategies determine how caches are kept consistent
- Caches may become inconsistent when data item is modified:
  1. at the server replicas, or
  2. at the cache

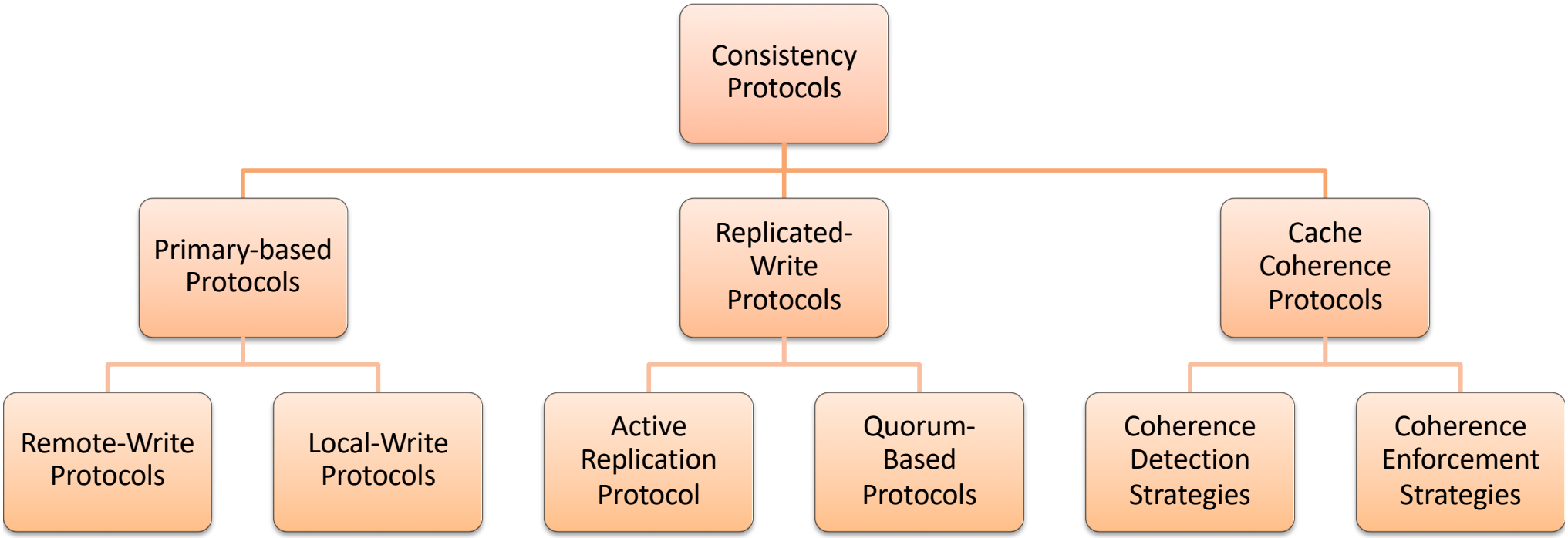
1. When Data is Modified at the Server

- Two approaches for enforcing coherence:
  1. Server-initiated invalidation
     - Here, server sends all caches an invalidation message when data item is modified
  2. Server updates the cache
     - Server will propagate the update to the cache

2. When Data is Modified at the Cache

- The enforcement protocol may use one of three techniques:
  i. Read-only cache
     - The cache does not modify the data in the cache
     - The update is propagated to the server replica
  ii. Write-through cache
     - Directly modify the cache, and forward the update to the server
  iii. Write-back cache
     - The client allows multiple writes to take place at the cache
     - The client batches a set of writes, and will send the batched write updates to the server replica

Overview of Consistency Protocols



Consistency and Replication – Brief Summary

- Replication improves performance and fault-tolerance
- However, replicas have to be kept reasonably consistent

Consistency Models

- A contract between the data-store and process
- Types: Data-centric and Client-centric

Replication Management

- Describes where, when and by whom replicas should be placed
- Types: Replica Server Placement, Content Replication and Placement

Consistency Protocols

- Implement Consistency Models
- Types: Primary-based, Replicated-Write, Cache Coherence

Case Study

EVENTUAL CONSISTENCY

Overview

- Until now: Completed good number of topics
  - Mutual Exclusion & Election
  - Naming
  - Clocks & Time
  - Consistency & Replication
- We said
  - Real systems base on combinations of buildings blocks from these
  - Now: Time to have a look at one system
    - More to come in the application section of the course

Ingredients

- Two main ones
  - Vector Clocks
  - Eventual Consistency
- Others
  - Unique names
  - Mutual Exclusion (via conflict detection)
- Keep the ingredients in mind for labs 3 and 4 ;-)

Eventual Consistency

- What is eventual consistency?
  - All replicas gradually converge to the same value
    - In the absence of updates
- Today: Discuss a protocol
  - Writes are *eventually* applied in total order
    - -> same order on all replicas
    - -> lead to the same value
    - -> eventual consistency
  - Reads might not see most recent writes in total order

Bayou

- System for eventual consistency
- Developed at Xerox PARC in 1997
  - One of the first good-working solutions to eventual consistency
  - Basis for many of today's protocols
  - Reasonable simple (compared to today's protocols)
    - Can discuss it in the remainder of this lecture
- Other things invented at Xerox PARC?
  - Laser printers
  - The graphical user interface (windows, icons, …)
  - Computer mouse
  - Ethernet
  - WYSIWYG text editor
  - …. strongly impacted modern computers and networks

Motivation & Goals

- What is this?
  - The predecessor of your smartphone

- Goal: Distributed Calendar
  - Access to shared resource with limited connectivity
  - Incl. conflict resolution etc.

- Why limited connectivity?
  - GSM / GPRS
    - Very expensive in 1997
  - WIFI, Bluetooth, …
    - Very sporadic in 1997

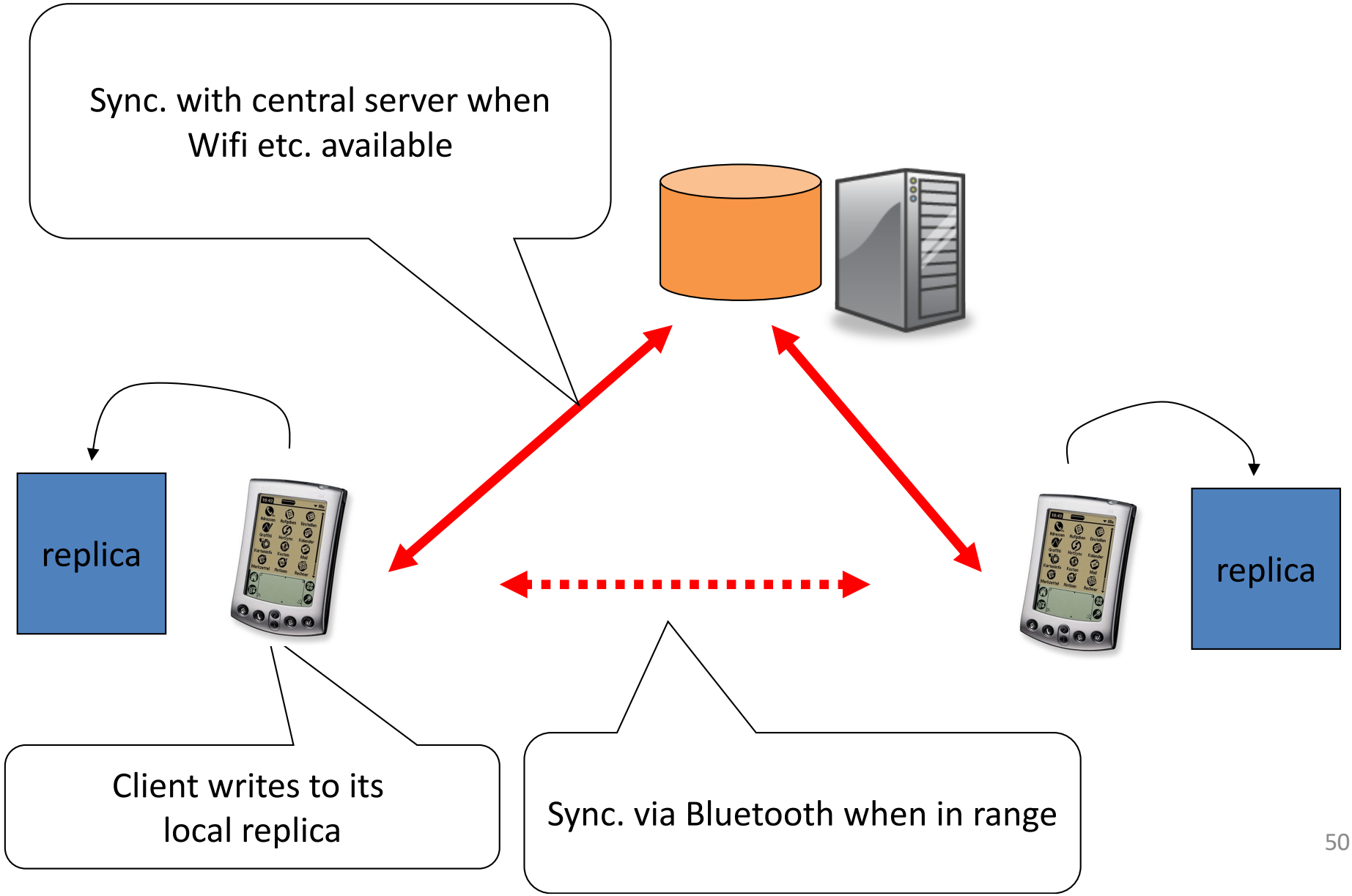
Motivating Scenario: Shared Calendar

- Goals
  - Calendar updates are made by several people
  - Want to allow offline updates
    - Conflicts can't be prevented
- Two possibilities:
  - Disallow offline updates?
  - Conflict resolution? 

Conflict Resolution

- Replication **not** transparent to application
  - Only the application knows how to resolve conflicts
  - Conflict: If it figures out
    - same room booked twice at same time
  - Resolution:
    - give priority to first booking, notify other booking
- Split of responsibility:
  - Replication system: propagates updates
  - Application: resolves conflict
  - Very common approach today, too
- Optimistic application of writes
  - Requires that writes be “undo-able” -> rollback
  - In case of conflict

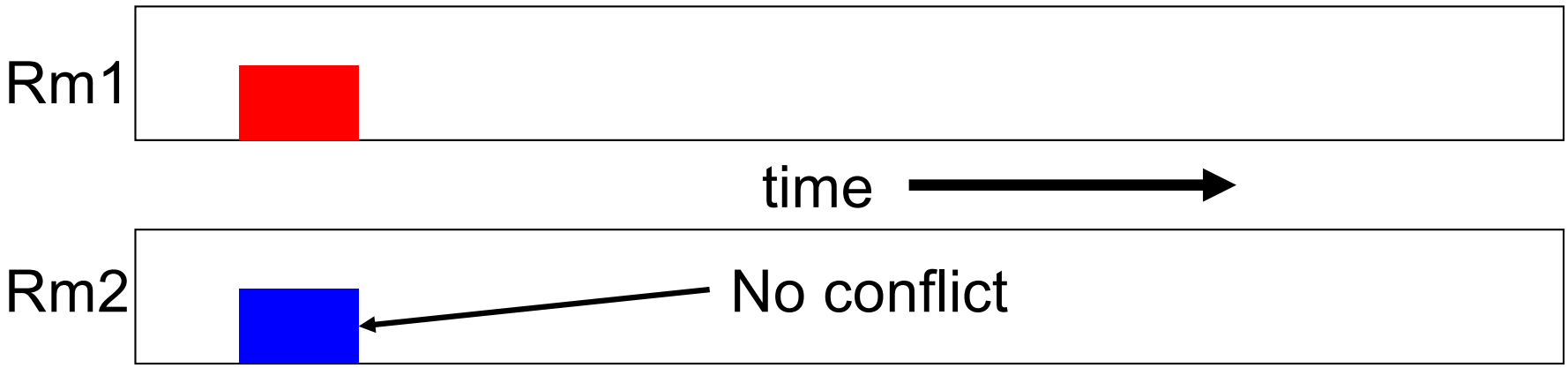
Basic Setting: Local Replicas



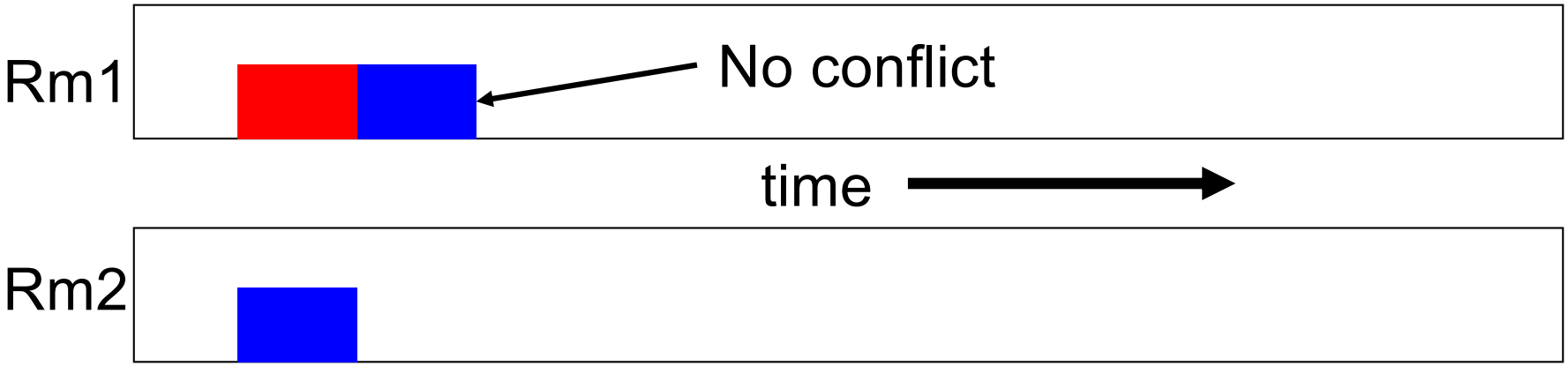
Meeting Room Scheduler

- Conflicts Scenarios:
  - Reserve same room at same time: conflict
  - Reserve different rooms at same time: no conflict
  - Reserve same room at different times: no conflict

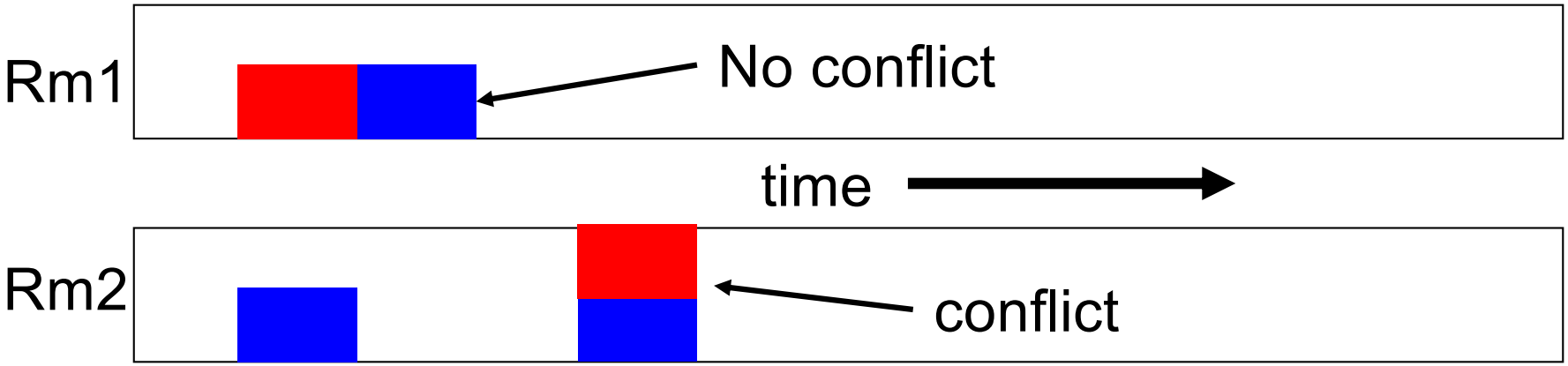
Only the application would know this!



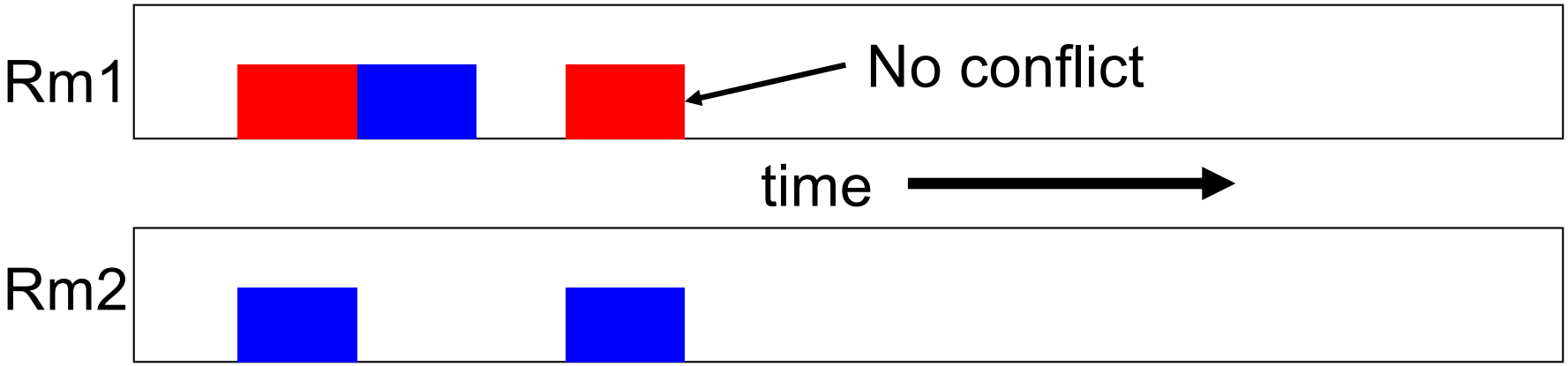
Meeting Room Scheduler



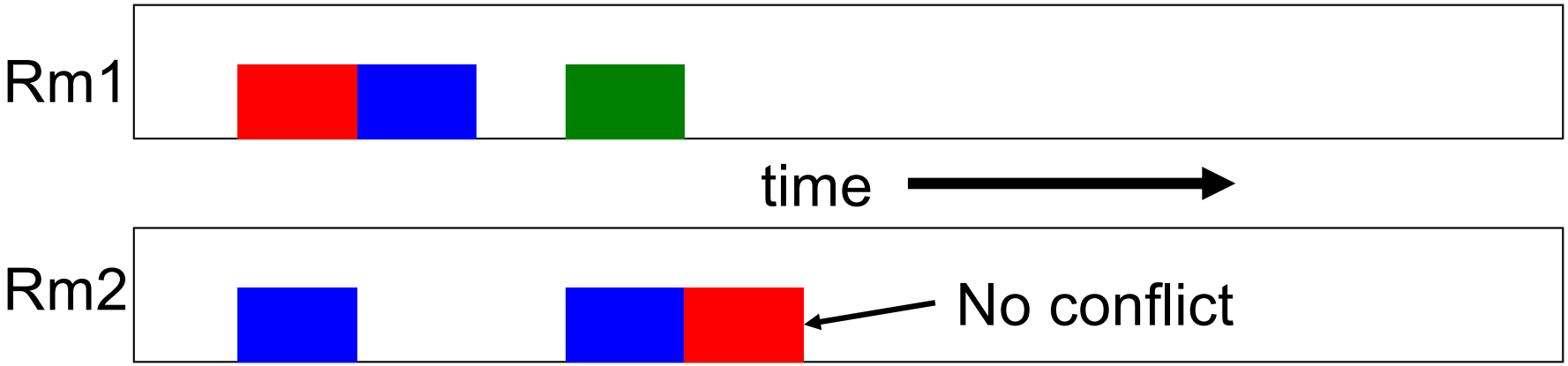
Meeting Room Scheduler



Meeting Room Scheduler



Meeting Room Scheduler

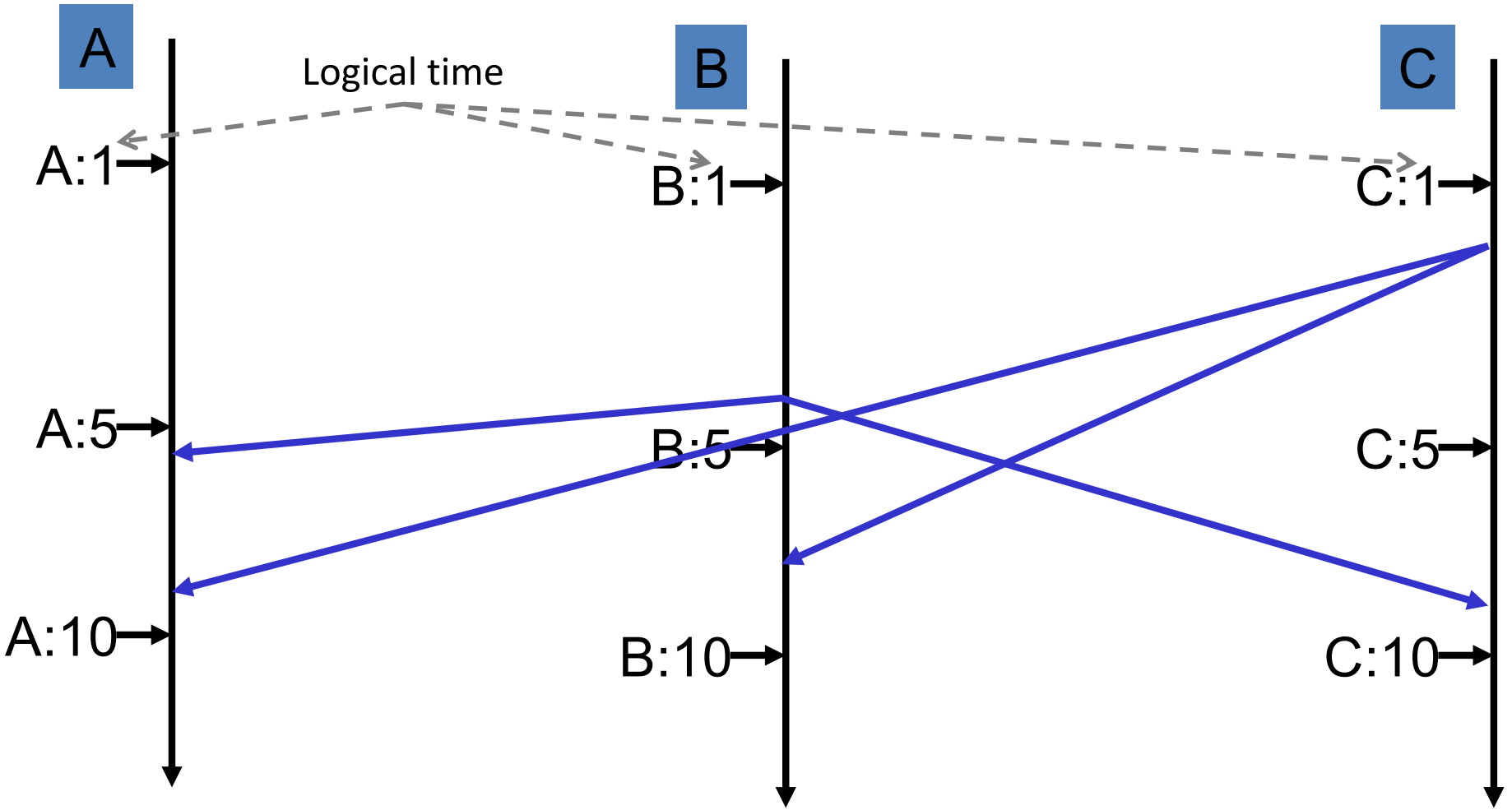


Handling Updates

- When a delayed update comes in
  - Example:
    - Booking got delayed in the network due to sporadic connectivity
    - And we also received more recent updates from other node

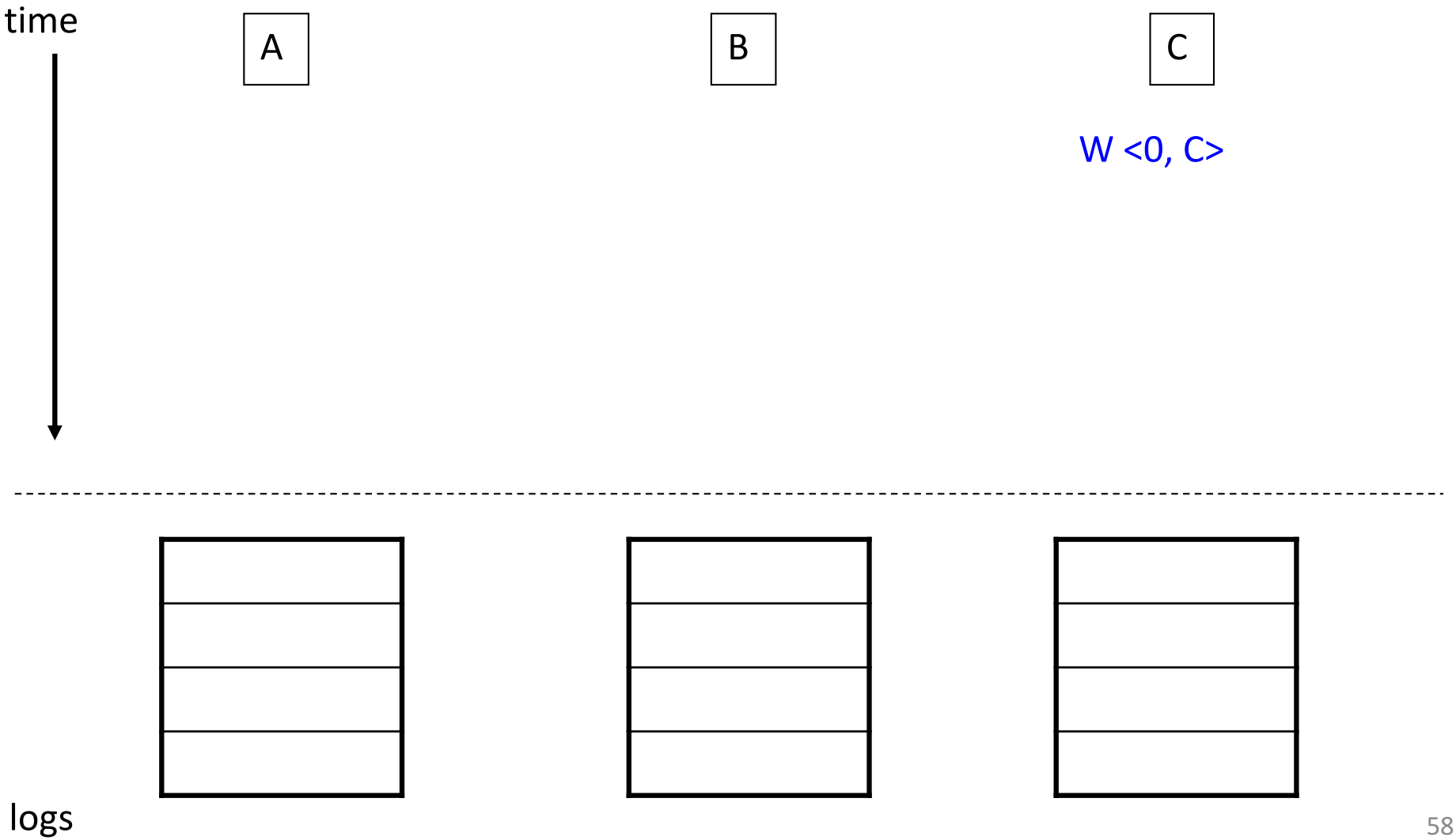
1. Rollback to update
   - How?
     - Keep logs of updates
2. Place it in the correct order and reapply log of updates
   - How?
     - Vector clocks ;-)
3. Resolve any conflicts
   - How?
     - Rule: Give priority to first booking, notify other booking
     - First booking: lower timestamp -> “happened first”

Example of an Undo

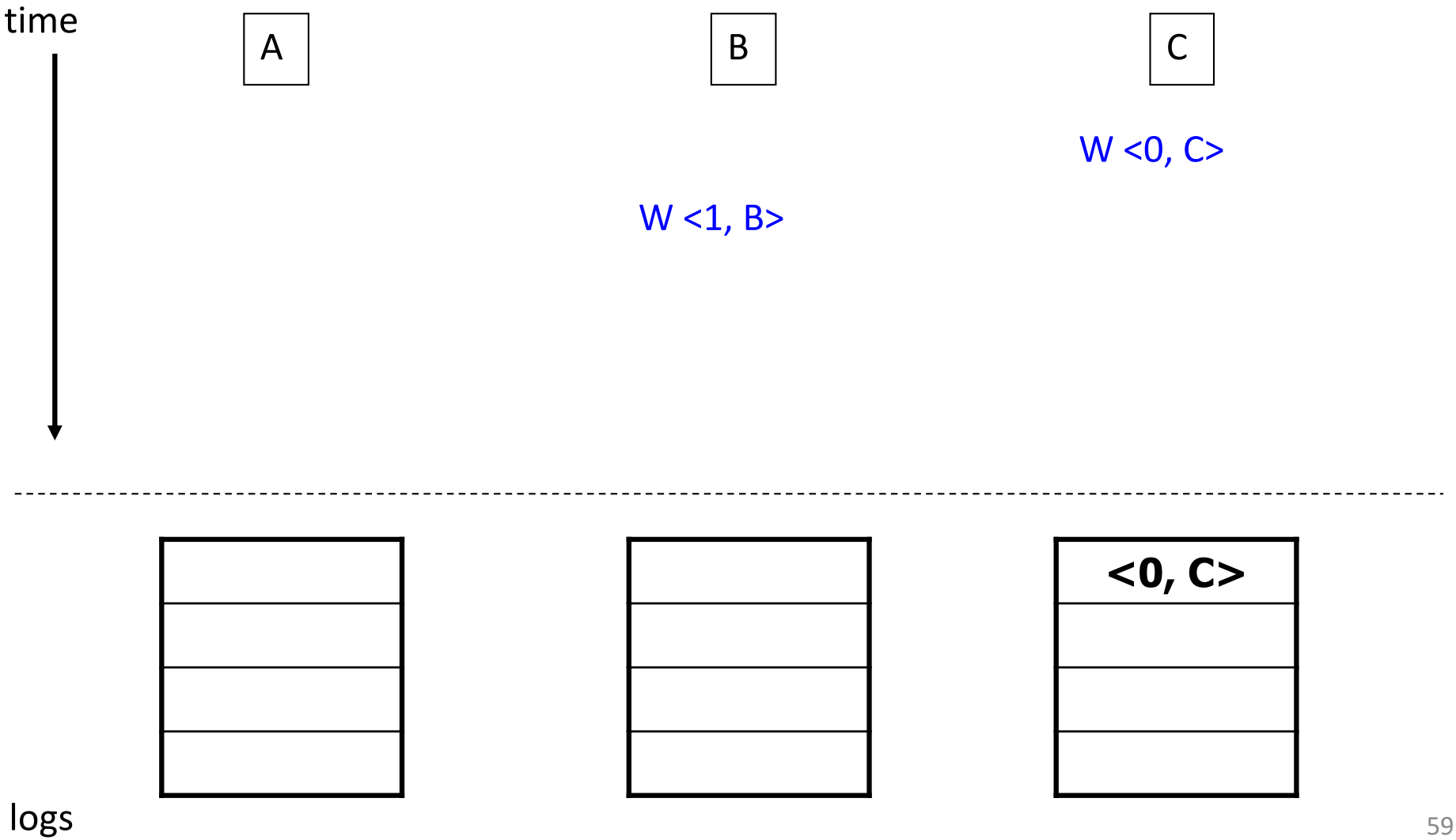


A will undo update from B, apply C and then B

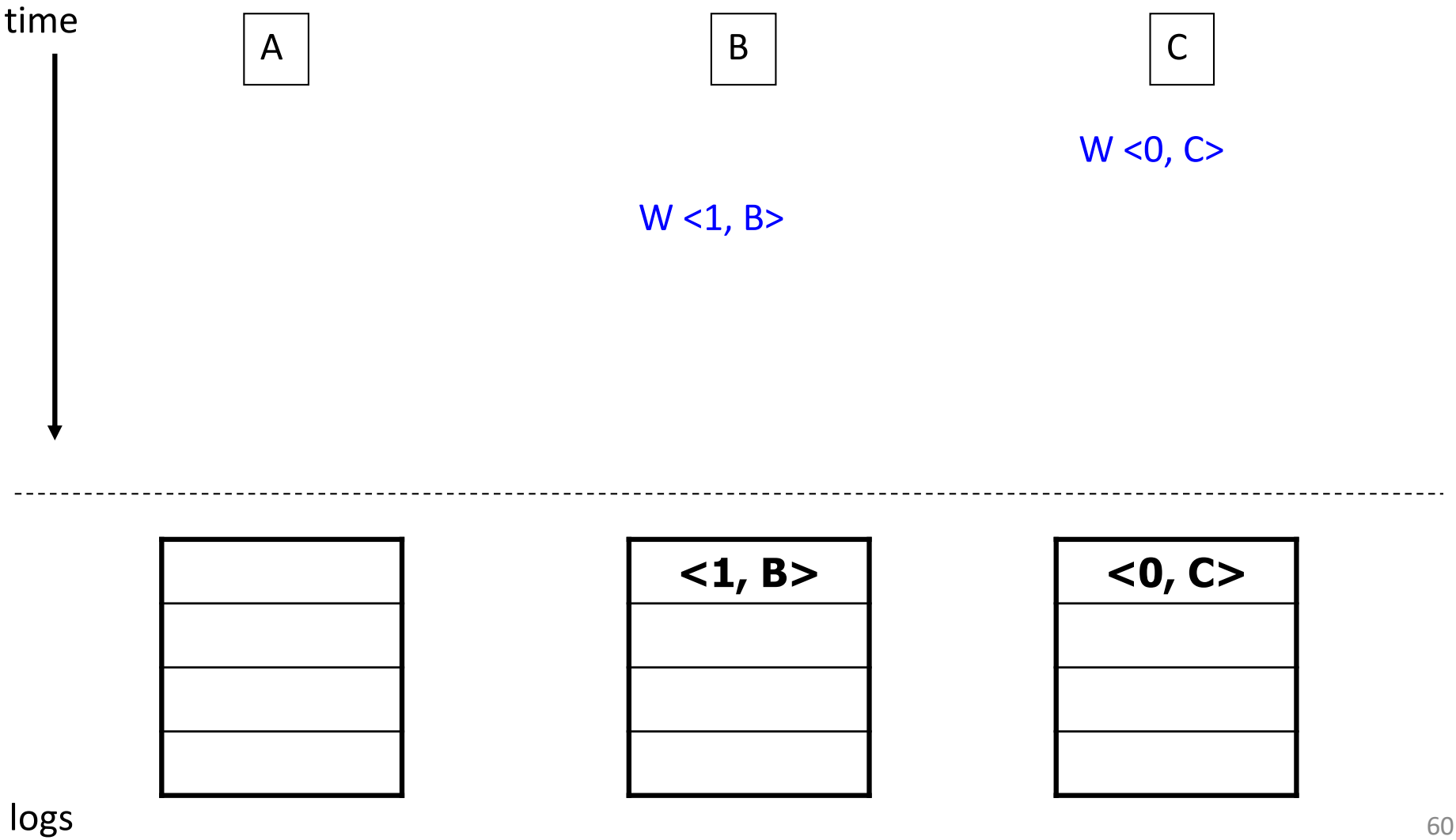
Example: Disagreement on Tentative Writes



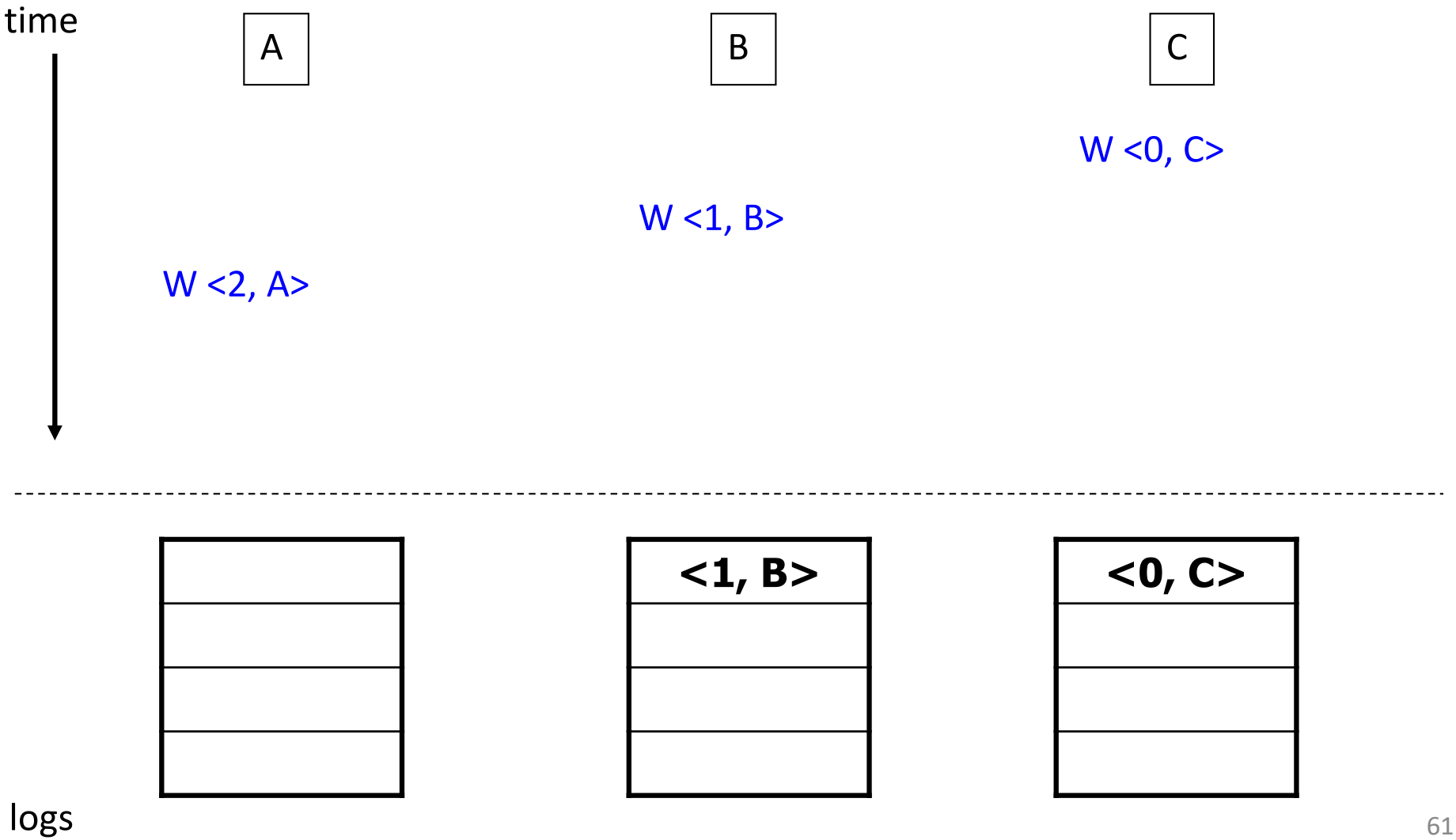
Example: Disagreement on Tentative Writes



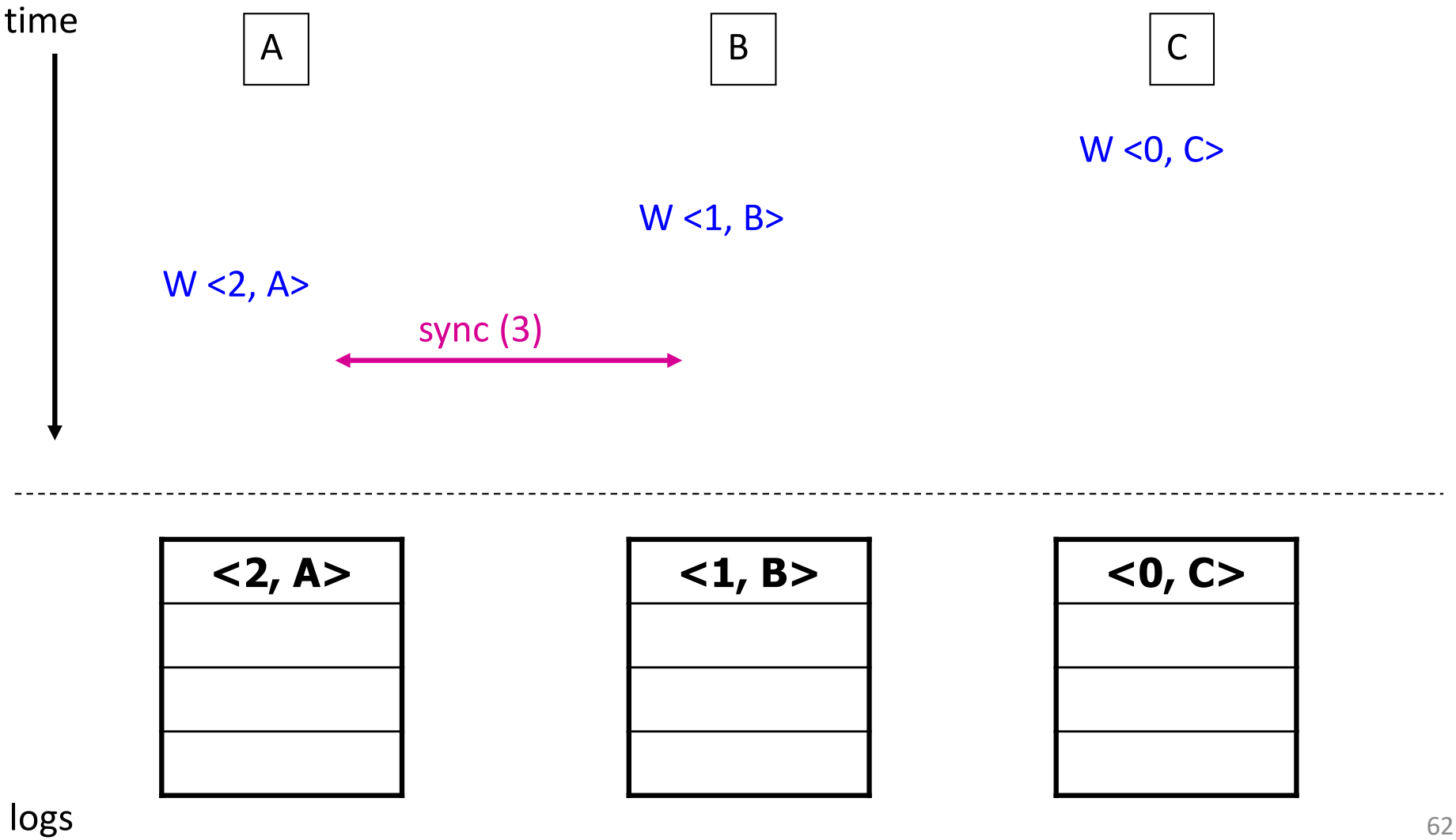
Example: Disagreement on Tentative Writes



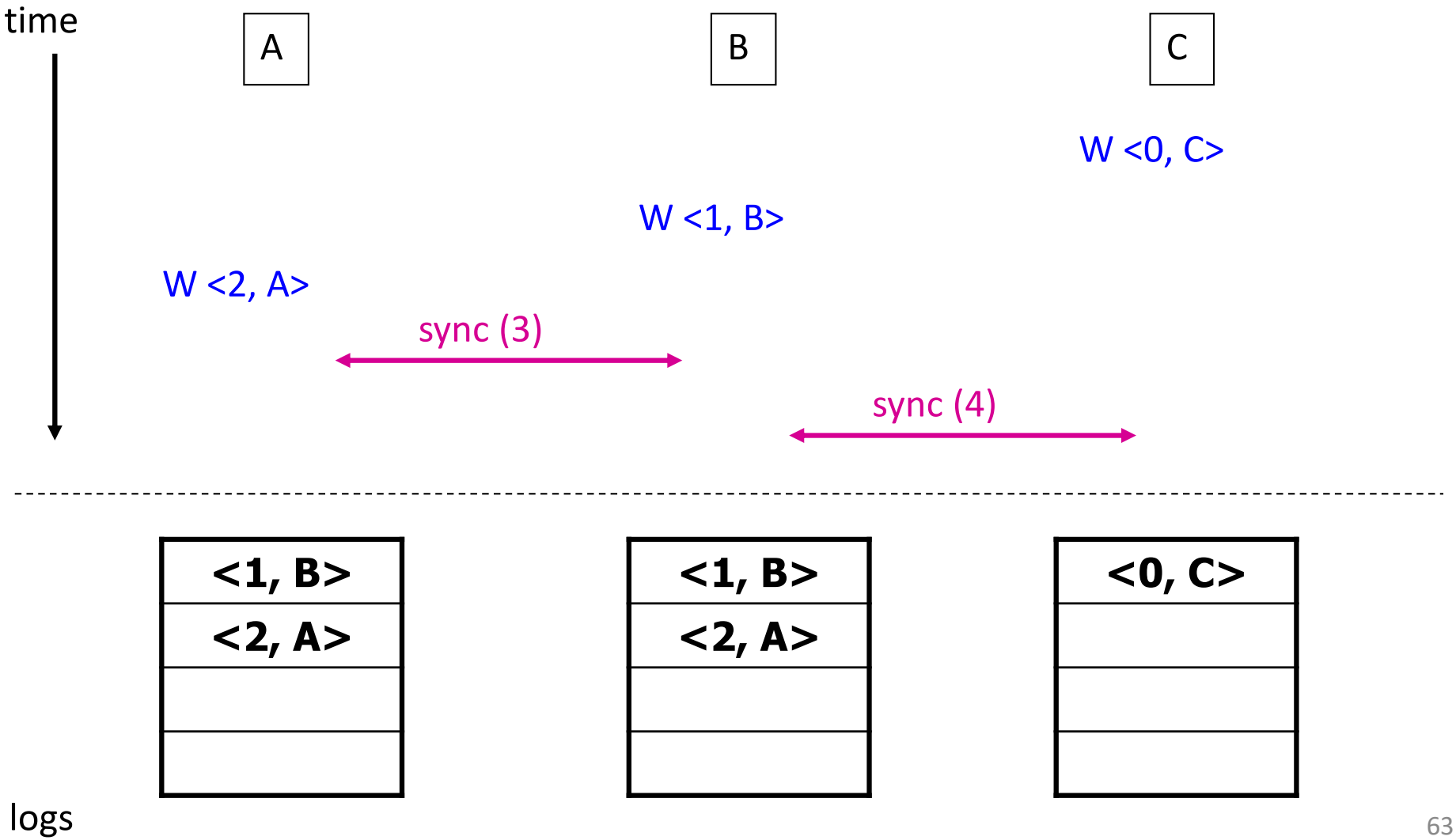
Example: Disagreement on Tentative Writes



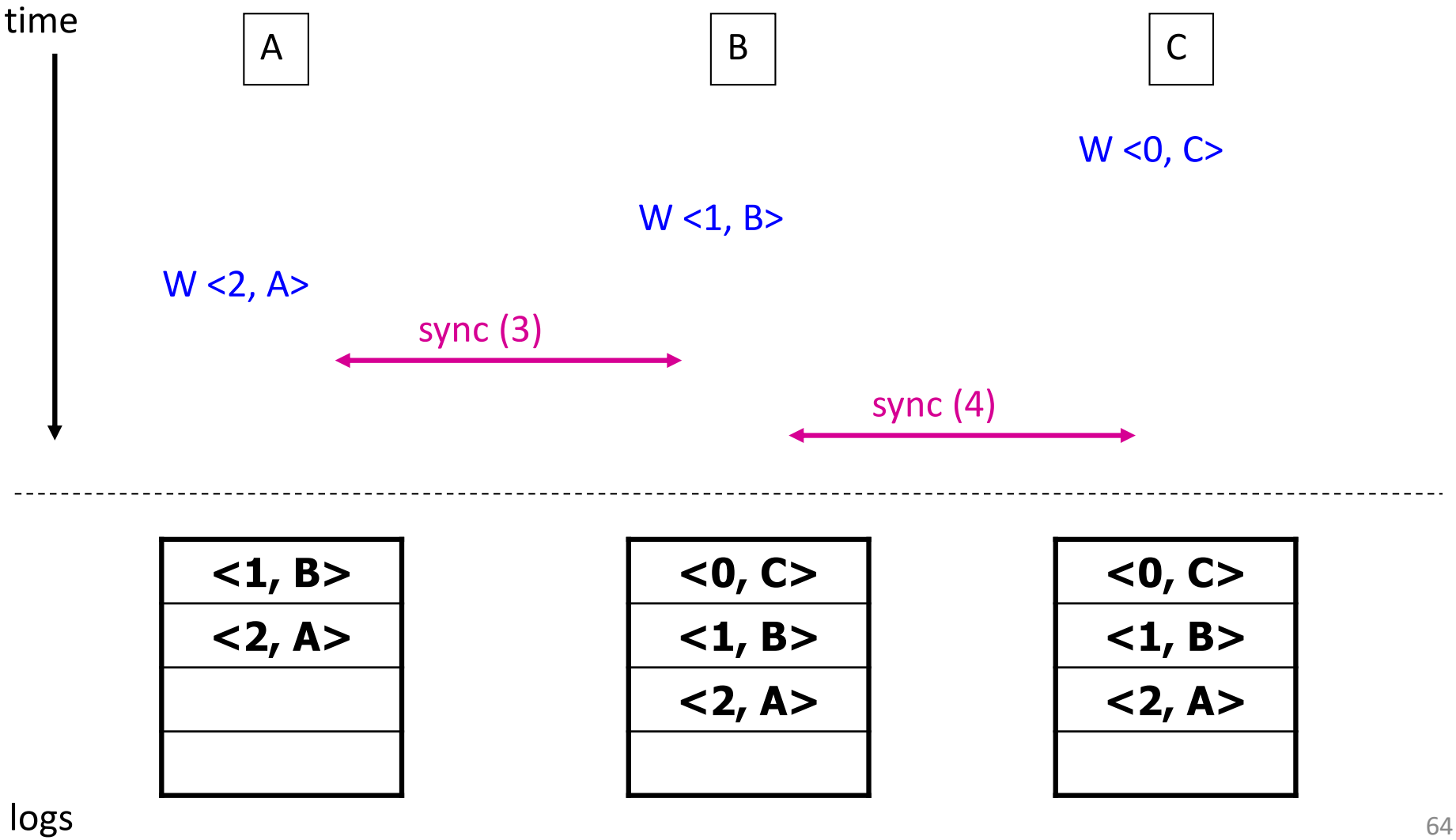
Example: Disagreement on Tentative Writes



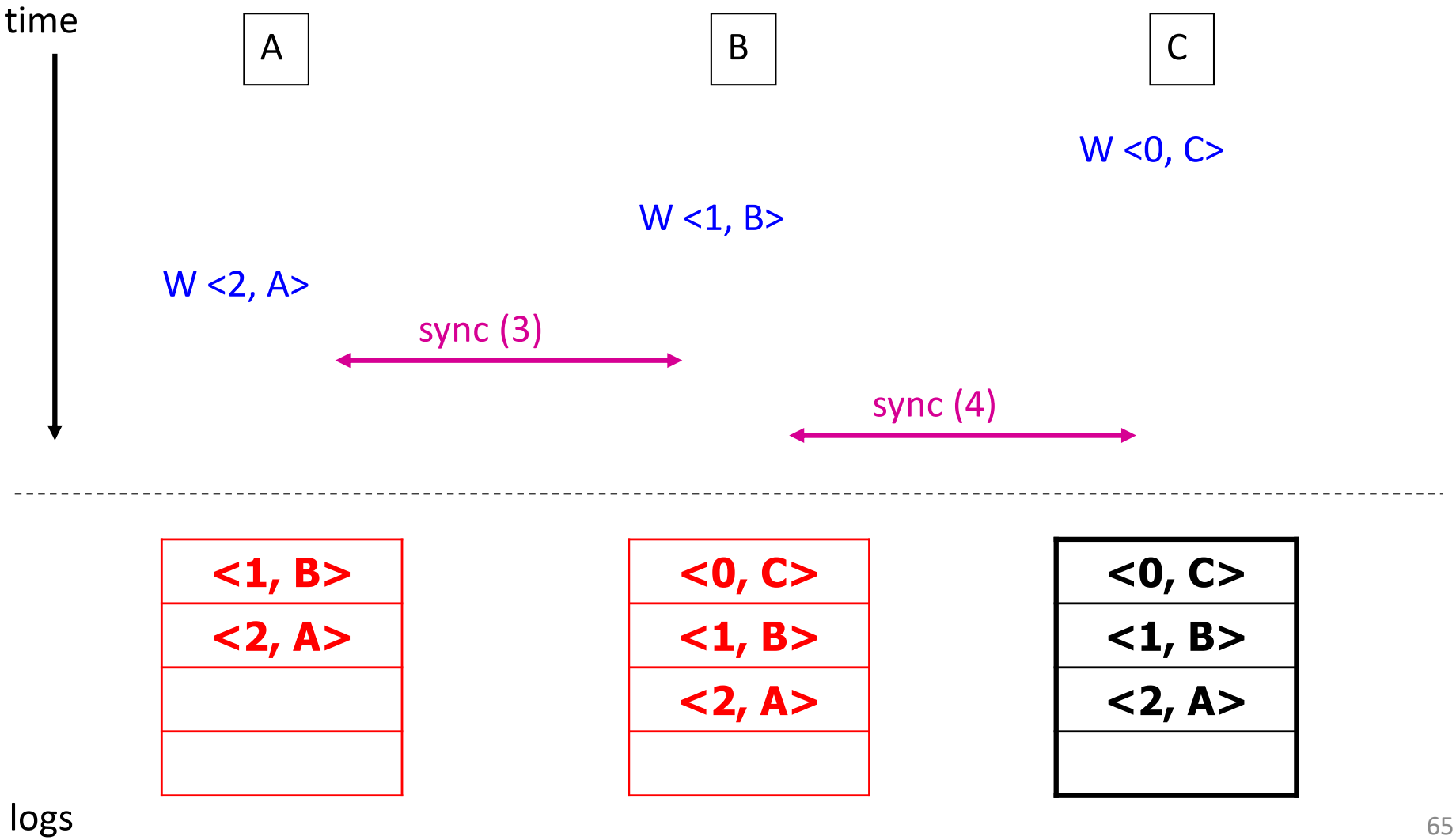
Example: Disagreement on Tentative Writes



Example: Disagreement on Tentative Writes



Example: Disagreement on Tentative Writes



Questions

- Summarize Bayou
  - Goals?
  - How does it work?

- When are updates propagated?
  - Eventually
- When are all conflicts detected?
  - Eventually
- What do we need Vector Clocks for?
  - To detect which bookings happened first, roll back
- How do we resolve conflicts?
  - Keep older booking, signal conflict to user
- When do I know that my booking has no conflicts?
  - When all previous ones arrived
- Is there are guaranteed time bound on this?
  - No, it will just happen eventually

Two Basic Elements

- Flexible update propagation
  - Eventually, all updates will reach all nodes
  - Eventually, all conflicts will be detected
- Dealing with inconsistencies
  - Detect in logs
  - Resolve conflicts by asking for help from user
- Lab 3: think of Bayou

Next Time

- Fault-tolerance
  - How to detect and deal with failures in Distributed Systems?

Questions?

In part, inspired from / based on slides from

- Vinay Kolar
- Brad Karp
- Scott Shenker and Ion Stoica