CS5223
Distributed Systems

Lecture 6: Replication and Consistency

Instructor: YU Haifeng

Roadmap

- Chapter 7 of textbook
- Consistency model – specifies what is consistent and what is not
- Consistency protocols – implements a certain consistency model
  - May use different protocols to implement the same model
- Replica placement
- Consistency is a large topic, we need 2 lectures and still will only have time to cover the major issues and not every detail in depth

Motivation for Data Replication

- Replication can be used for data or functionality
  - Data replication is more prevalent
- Reliability (durability)
  - Backup your data
- Availability
  - Being available to clients all the time
- Performance
  - Share the load
  - Being closer to clients using the data

Disadvantage of Data Replication: Consistency Maintenance

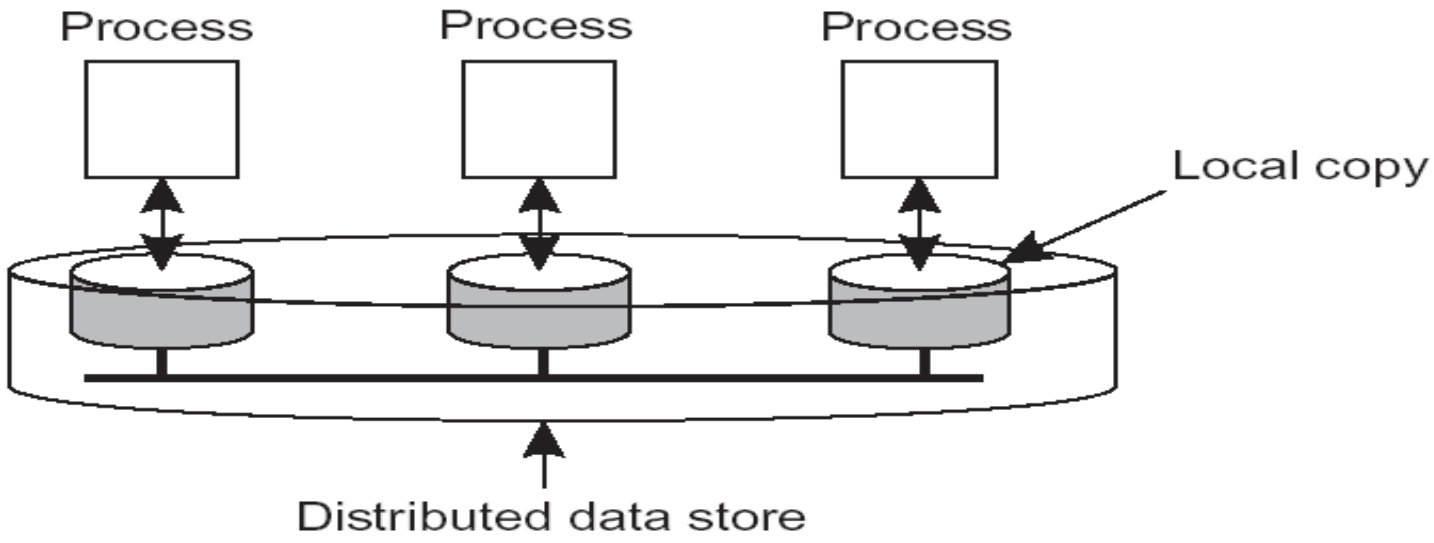
- Data, by definition, allow reads and writes
  - Reads and writes are very different for replicated data
  - If data only allow reads, then it is readonly
- Caching: A restricted form of replication
  - Usually refers to partial replication of readonly or rarely-updated data
  - But different people use the term differently in different context – important to know what exactly it refers to
- Consistency: If one copy is updated, then other copies need to be “in synch” with the update
  - By definition, readonly data do not have consistency issue

Complications of Consistency Maintenance

- Design complexity
  - Complexity is always bad
- Reduction on reliability / availability / performance gains
  - Reads on readonly data always enjoy these gains
  - Writes may or may not enjoy these gains
  - Reads on read/write data may not enjoy these gains
- Readonly data almost always benefit from replication
  - Only cost is design complexity and storage space

Model for Replicated Data

- Read operation: A single or multiple reads
  - Similar as a readonly transaction
- Write operating: A single or multiple reads/writes
  - Contains at least write, may contain read
  - Similar as an update transaction



What Is Consistency?

- Consistency – a term with thousand definitions
- Definition in this course:
  - Consistency model specifies what behavior is allowed when shared data are accesses by multiple node
  - When we say something is “consistent”, we mean it satisfies the specification (according to some given spec)
- Specification:
  - No right or wrong – anything can be a specification
- But to be useful, must:
  - Be sufficiently strong – otherwise the shared object cannot be used in a program
  - Can be implemented (efficiently) – otherwise remains a theory
  - Often a trade-off

The Consistency Spectrum

strong

easy-to-use

low performance

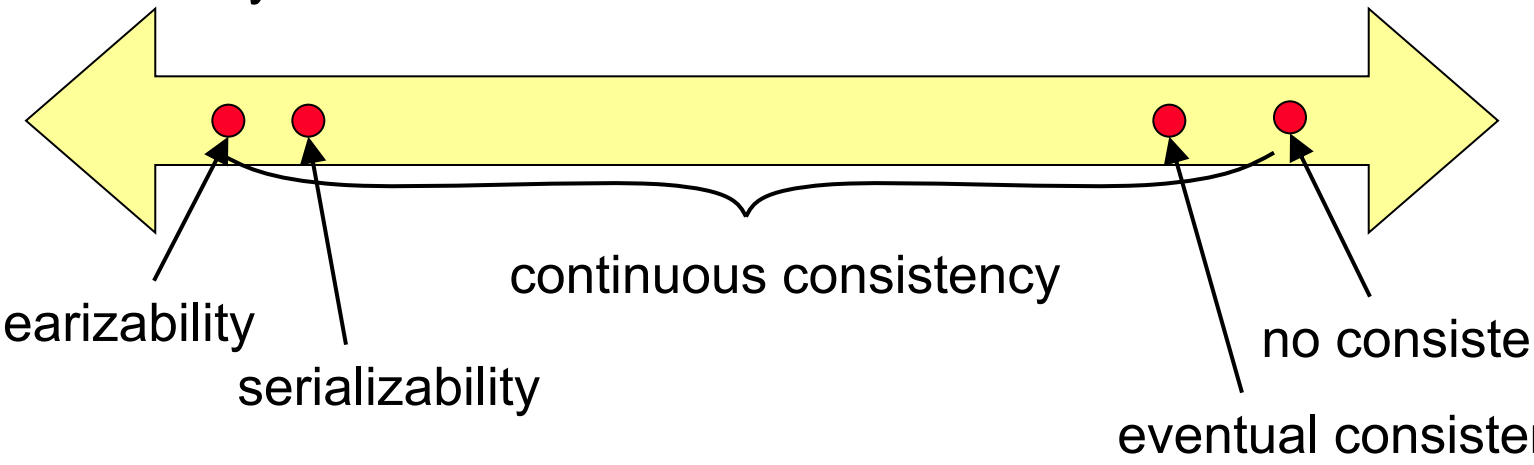
low availability

weak

hard-to-use

high performance

high availability



Note:

Huge number of consistency models have been defined by people

Not all consistency models are comparable

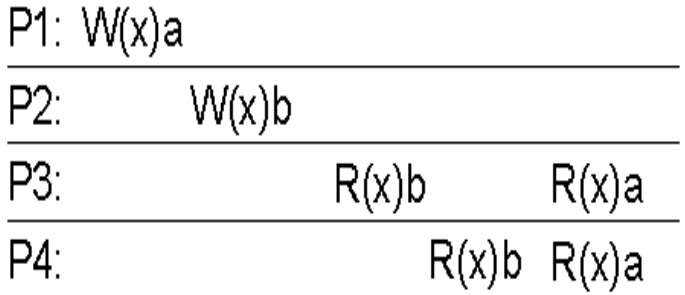
Continuous consistency does not capture all consistency models

Sequential Consistency

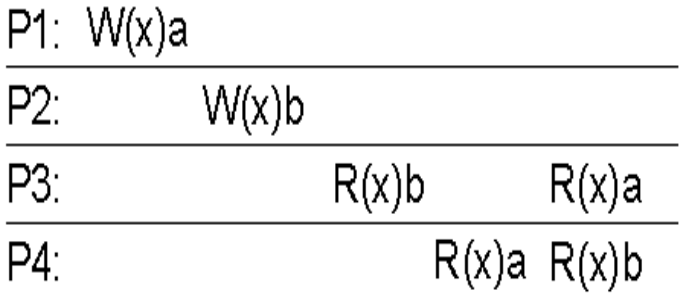
- Perhaps the most widely used consistency model
  - Multiprocessors usually guarantees sequential consistency
- First defined by Lamport
  - "…the results … is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by the program"

- Use sequential order as a comparison point
- Focus on results only – that is what users care about
- Require that program order is preserved
- One-copy serializability in database context

Examples



(a)

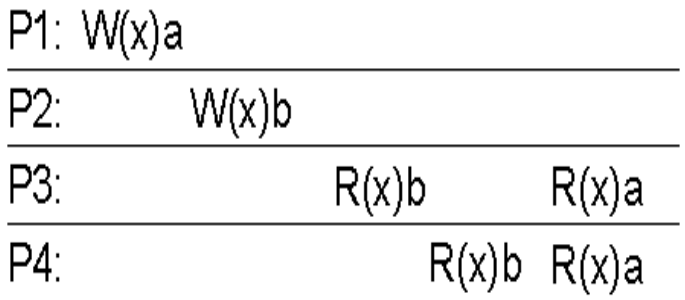


(b)

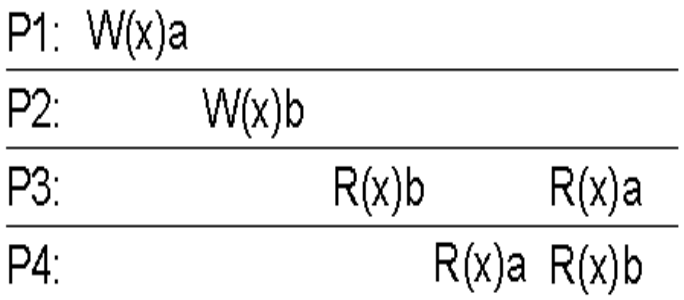
(a) = P2-W(x)b P3-R(x)b P4-R(x)b P1-W(x)a P3-R(x)a P4-R(x)a

OK for P2-W(x)b be ordered before P1-W(x)a according to the definition of sequential consistency – your program must accept such possibility

Examples



(a)



(b)

(b) = P2-W(x)b P3-R(x)b P4-R(x)b P1-W(x)a P3-R(x)a P4-R(x)a

P4-R(x)a cannot appear after P4-R(x)b

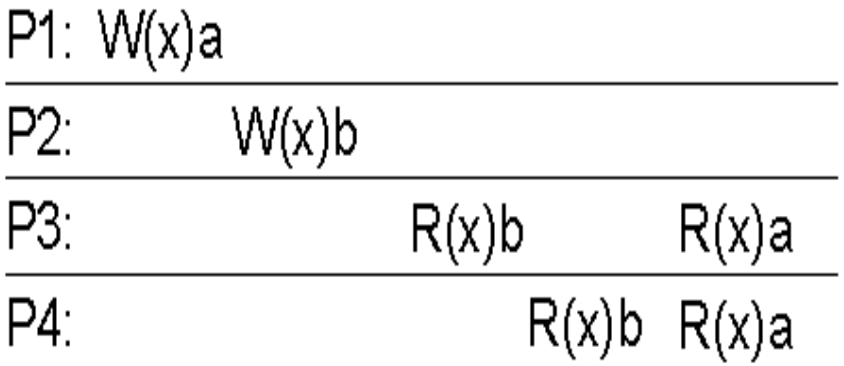
- But still don't know if it is sequentially consistent – why?
  - Enumerate all 6! orderings

Linearizability – A Stronger Model

- The strongest consistency model define ever
- Also called one-copy serializability + external consistency

- Sequential consistency allows reading stale data – sometimes undesirable
  - Example?
- Linearizability captures real-time
  - An operation O2 that occurs after O1 in real-time must be ordered after O1
- Linearizability necessarily implies sequential consistency

Examples



(a)

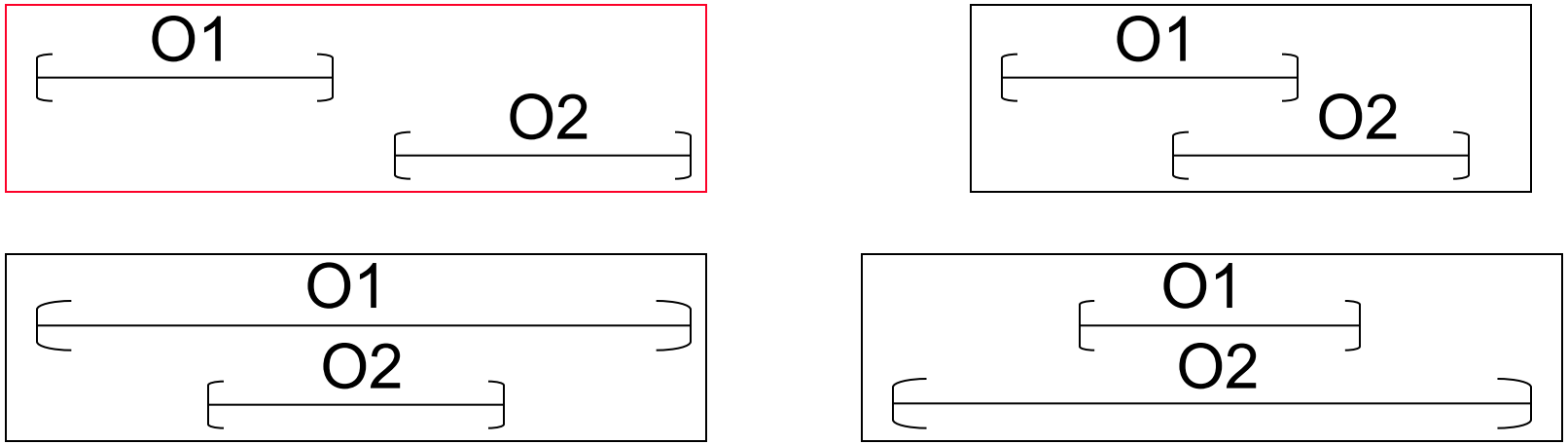
(a) = P2-W(x)b P3-R(x)b P4-R(x)b P1-W(x)a P3-W(x)a P3-R(x)a P4-R(x)a

Not OK for P2-W(x)b be ordered before P1-W(x)a

- But still don't know if it is linearizable – why?
  - Enumerate all 6! orderings

Linearizability – More precise

- An operation O2 that occurs after O1 in real-time must be ordered after O1
- What do we mean by "occurs after O1 in real-time"?
  - Each operation takes non-zero time to execute



Eventual Consistency

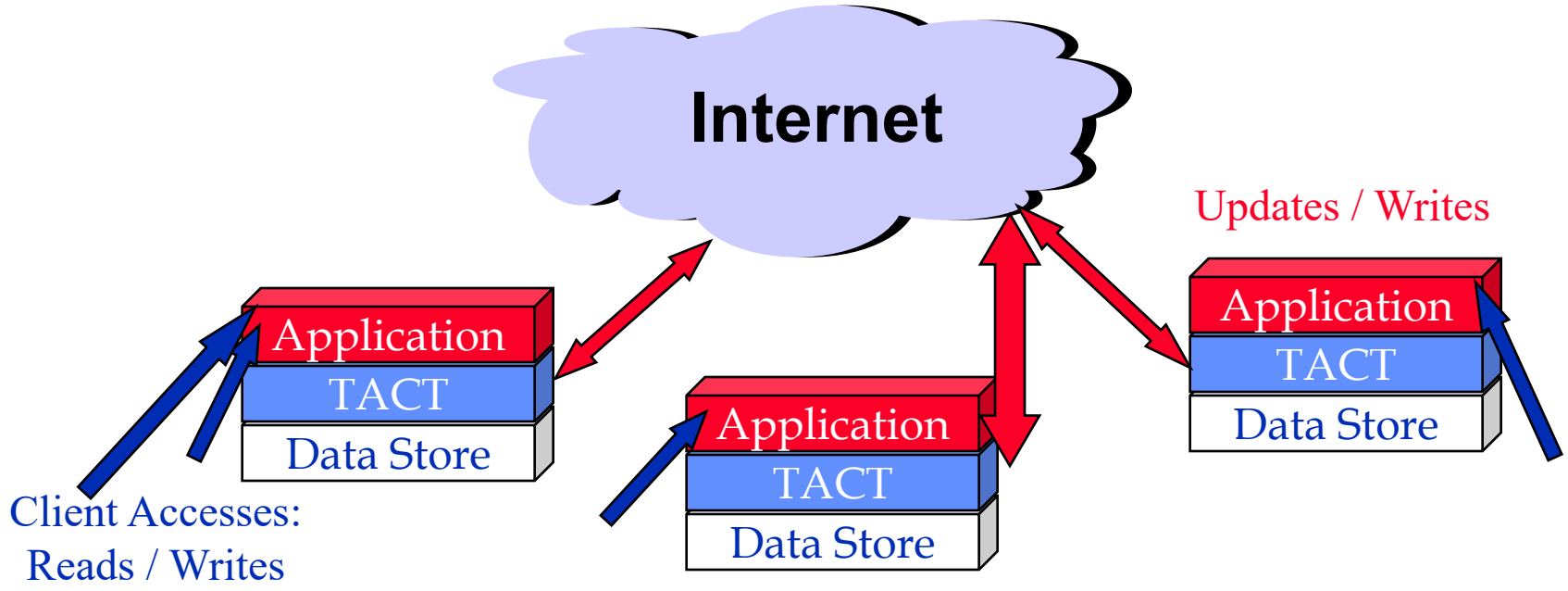
- Eventual consistency (optimistic consistency)
  - No guarantee on the values observed by reads and writes
  - But if we stop doing updates to the data, all replicas will eventually be the same (i.e., converge)
  - The state of the replica depends on the sequence of write operations
- Very weak guarantee but still sufficient in many cases
  - DNS replication/caching
  - Web page replication/caching
- Allows extremely good performance/availability

Motivation for Continuous Consistency

- In many applications, we want something in the middle
  - Don't want to design a new consistency model for every application
  - Don't want to implement a new consistency model for every application
  - Sometimes consistency needs may change on-the-fly – example?
- Continuous consistency
  - Capture the consistency spectrum using a single consistency model and consistency protocol
  - Allow adjustment of consistency "strength" dynamically
- Many continuous consistency models have been proposed
  - We will focus on the model developed in TACT (as described in the textbook)

System Model for Continuous Consistency in TACT

- Data store fully replicated
- Each replica may accept reads operations or write operations
- Writes are re-executed at each replica
- Writes may be undo or redo
- Replica convergence achieved by propagating writes – eventual consistency always ensured



Intuition for Defining Continuous Consistency

- *Observed result*: The result returned to client
- *Ideal result*: The ideal result perfect consistency were maintained

- Continuous consistency should quantify the "distance" between observed result and ideal result
  - Since eventual consistency is always ensured, we can conceptually define a "final image" – performing the operation against the final image would give us the "ideal result"
  - There can be multiple possible final images – all of them are good – but one and only one will be picked as reference
  - The system does not know the final image

Quantifying Consistency

- Three consistency metrics:
  - With respect to some final image – the reference image
  - **Numerical Error:** How many writes are missing?
  - **Order Error:** How many writes are out of order? (According to serialization order in the reference image)
  - **Temporal Error:** How stale (in real-time)?
  - Largely orthogonal but not entire orthogonal
- Consistency specified as bounds on the three metrics

Consistency Metric Example

Replica A

W3 (*accepted by A*)

(Absolute) Numerical Error = 2
(from W1 and W2)
(Relative) Numerical Error = 2/3
(from W1 and W2)
Order Error = 1
(from W3)
Temporal Error = currenttime - accepttime of W1

Replica B

W1 (*accepted by B*)
W2 (*accepted by B*)
W3 (*accepted by A*)

(Absolute) Numerical Error = 0
(Relative) Numerical Error = 0
Order Error = 0
Temporal Error = 0

Assuming reference image is W1W2W3

Numerical Weight

Replica A

W3: *X* += 5

(Absolute) Numerical Error = 17

(from W1 and W2)

(Relative) Numerical Error = 17/22

(from W1 and W2)

Replica B

W1: *X* += 7
W2: *X* += 10
W3: *X* += 5

(Absolute) Numerical Error = 0

(Relative) Numerical Error = 0

Weight can be subjective

Weight not restricted to numerical data

Extremes of the Continuous Consistency Model

- If none of NE, OE, TE are bounded:
  - Eventual consistency
- If NE = 0, OE = 0, TE =0:
  - Linearizability

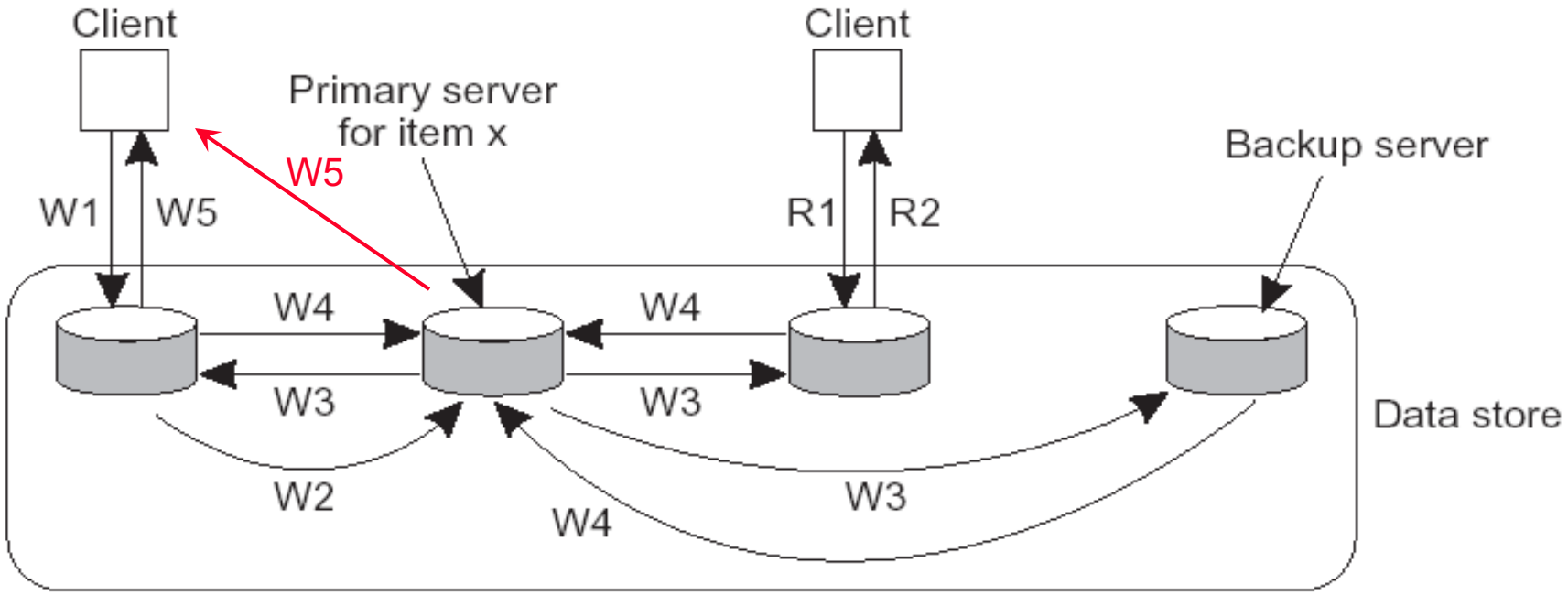
- By tuning NE, OE and TE, applications obtain fined-grained consistency levels and corresponding performance / availability

Consistency Protocols

- Linearizability / Sequential consistency
- Eventual consistency
- Continuous consistency

Linearizability/Sequential Consistency: Primary-based Protocol

- 1 primary replica, n-1 backup replicas
  - For now assume the primary replica is on some fixed node
  - For now assume the backup replicas are fixed as well
- Reads are process locally by each replica (primary or backup)
- All writes are forwarded to the primary (potentially via some backup replica)
  - Backup replicas never process writes directly



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

This example from the textbook only ensures serializability (why?)

The red W5 will give us linearizability

Generalization of Primary-based Protocol

- Dynamically create/destroy backup replicas
  - Create a new backup on a client's machine if the client accesses the data frequent enough
  - The primary replica needs to maintain a directory (i.e., which are the backup replicas)
  - Destroy a backup replica if it is not used for long (why?)
- Dynamically change a backup replica to a primary replica (and the old primary becomes backup)
  - Why do we want to do this?
  - Need a handshake with the old primary

Generalization of Primary-based Protocol

- Invalidating backup replicas instead of updating them
  - At every write, the primary can invalidate the backup replicas
  - An invalidated backup replica needs to get the up-to-date data from the primary before it can processing reads
  - Typically invalidation is not on the entire data store, but rather on much smaller granularity (e.g., individual objects)

Application of Primary-based Protocol: Software Distributed Shared Memory

- Software DSM is a middleware that provides a “shared-memory” abstraction for distributed applications running on a cluster
  - Hardware DSM usually refers to multiprocessor systems with distributed shared memory

- Motivation for software DSM:
  - Clusters are cheaper than multiprocessor systems and have more computation power
  - A lot of old scientific computing programs has been developed for shared-memory multiprocessors
  - They all use shared memory instead of message passing
  - Nobody really understand these programs – the cost of updating them is too large
  - Software DSM allows clusters to run them without modification

Discussion on Primary-based Protocols

- Very simple – good
- Used widely

- Performance is typically the focus
  - Improves read performance but hurts write performance
  - Many workloads (e.g., file systems) are dominated by reads
  - Do not use primary-based protocol if your workload is mostly writes!
  - More replicas $\Rightarrow$ Better read performance and worse write performance

- Availability
  - Good read availability
  - Poor write availability

Dealing with Failures in Primary-based Protocol

- Primary-based protocol typically cannot deal with any failures
  - If any replica fails, writes cannot be done – a serious problem (worse than having poor write performance)
  - Extremely difficult to deal with primary replica failure
- So we will assume below that the primary does not fail

- If a backup replica fails – we can simply ignore it (i.e., not pushing writes to or waiting for ack from it)
  - Challenge: Cannot tell if it indeed fails or it is just slow or network partition
  - What if we mistakenly treat a partitioned backup replica as failed?

Dealing with Failures in Primary-based Protocol

- Leases: A very simple and useful technique
  - Limited support to deal with backup replica failure / network partition
  - Assumes that clocks rate drifts are bounded
- Primary periodically gives each backup replica a lease
  - "I will always consider you as a backup for 10 minutes from now." – 10 minutes is the lease duration
  - Lease is renewed before expiration (either the primary can automatically issues a new lease or the backup may request one)
- A backup replica "kills itself" if it does not have a valid lease
  - More precisely – "do not process any reads"
- A primary can continue if it cannot reach a backup sufficient long
  - E.g., 15 minutes

Linearizability/Sequential Consistency: Replicated State Machine Protocol

- Motivations:
  - Want to deal with primary failures – remember that availability is one of the motivation for replication
- Replicated state machine:
  - All replicas are primary
  - Each replica is considered as a deterministic state machine
  - Each read/write operation consider as an “instruction”
  - All state machines needs to execute the same instructions in the same order
- Each read/write operation is broadcast to all replicas
  - All replicas have the same instructions

Linearizability/Sequential Consistency: Replicated State Machine Protocol

- Need to have an agreed ordering of all the instructions
  - Can be any order, but all replicas must use the same order
  - How?
  - The key issue in replicated state machine – needs fancy protocols to do so (in a fault-tolerant way)
- In primary-based protocol, the order is chosen by the unique primary replica
  - The primary replica is a "serialization point" or "sequencer"
- Replicated state machine with n replicas can usually tolerate n/2 replica failures
  - Availability!

Linearizability/Sequential Consistency: More Advanced Protocol

- Primary-based protocol
  - Good performance, bad availability
- Replicated state machine
  - Bad performance, good availability
- More advance protocol
  - Have a single primary, but can regenerate the primary if it fails
  - Allow new nodes to dynamically join the replica group

History Readings (Non-compulsory)

- “Implementing fault-tolerant services using the state machine approach: a tutorial”, ACM Computing Surveys, Volume 22 , Issue 4 (December 1990).

Eventual Consistency: The Anti-entropy Protocol

- System model:
  - Data store fully replicated
  - Each replica may accept reads operations or write operations
  - Writes are re-executed at each replica
  - Writes may be undo or redo
  - Each replica has a write log – log of all writes applied
  - Writes are gossiped from replica to replica

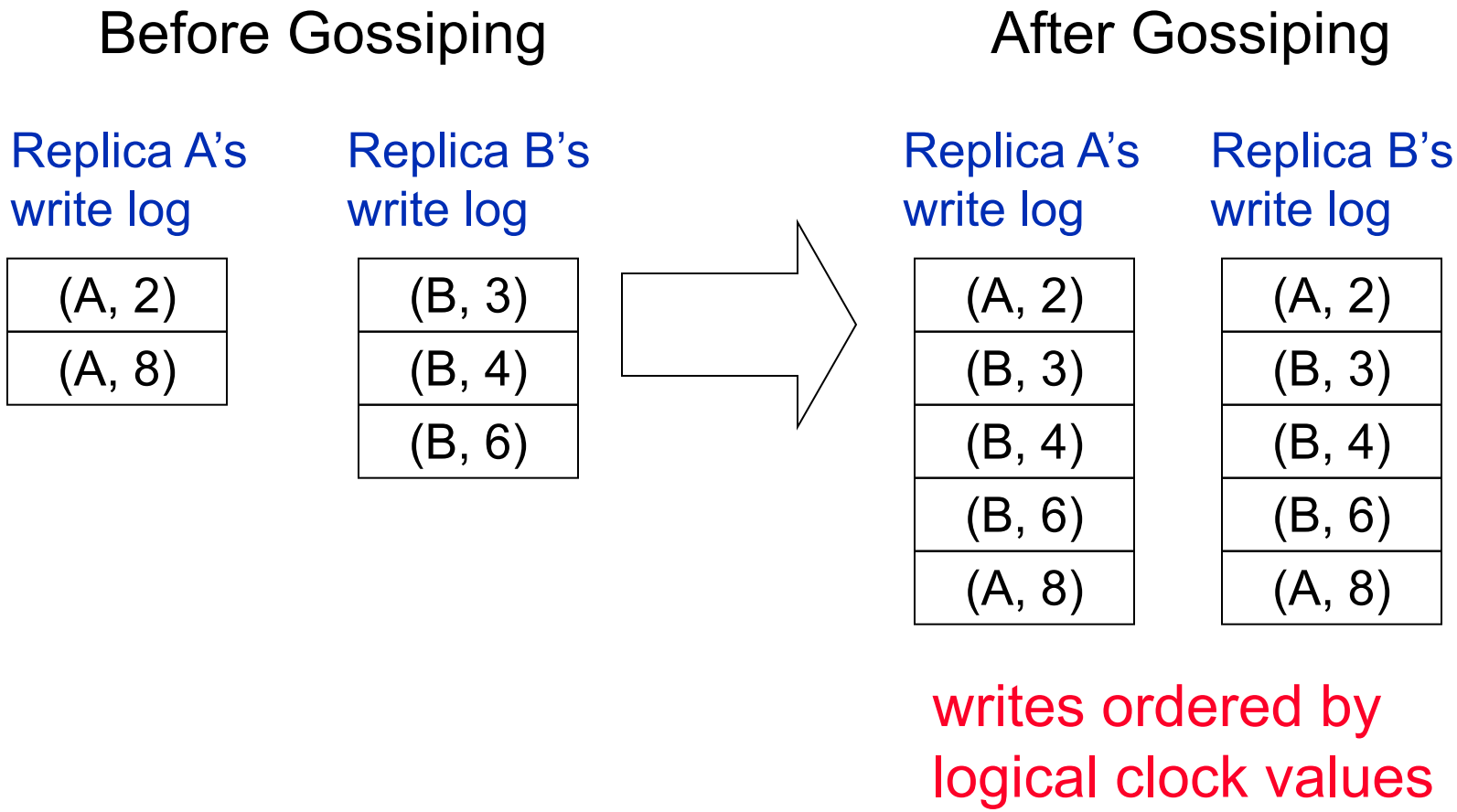
- Goal: Eventual consistency
  - All replicas see all writes – already ensured by gossiping
  - All replicas apply them in the same order (the serialization order) – this is the key

Generating a Total Order Among Writes

- If no restriction on the serialization order
  - Trivial (how?)

- Sometime the serialization order needs to preserve happened-before relation among writes
  - W1 creates a file and W2 deletes the file, then W1 should be ordered before W2 (what if we swap them?)
  - Order writes according to logical clocks

Example

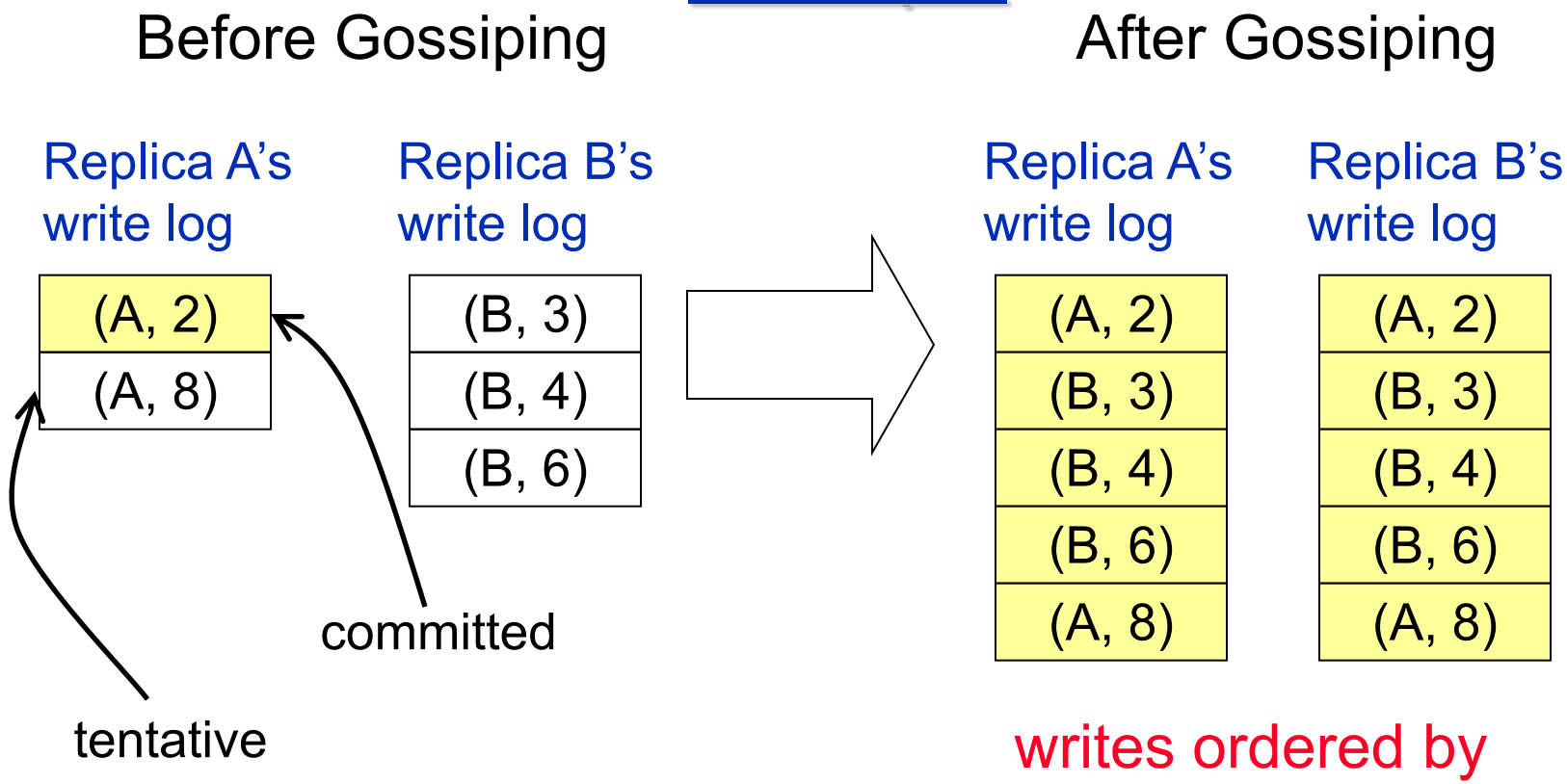


- What if logical clock values tie?

Committed vs. Tentative Writes

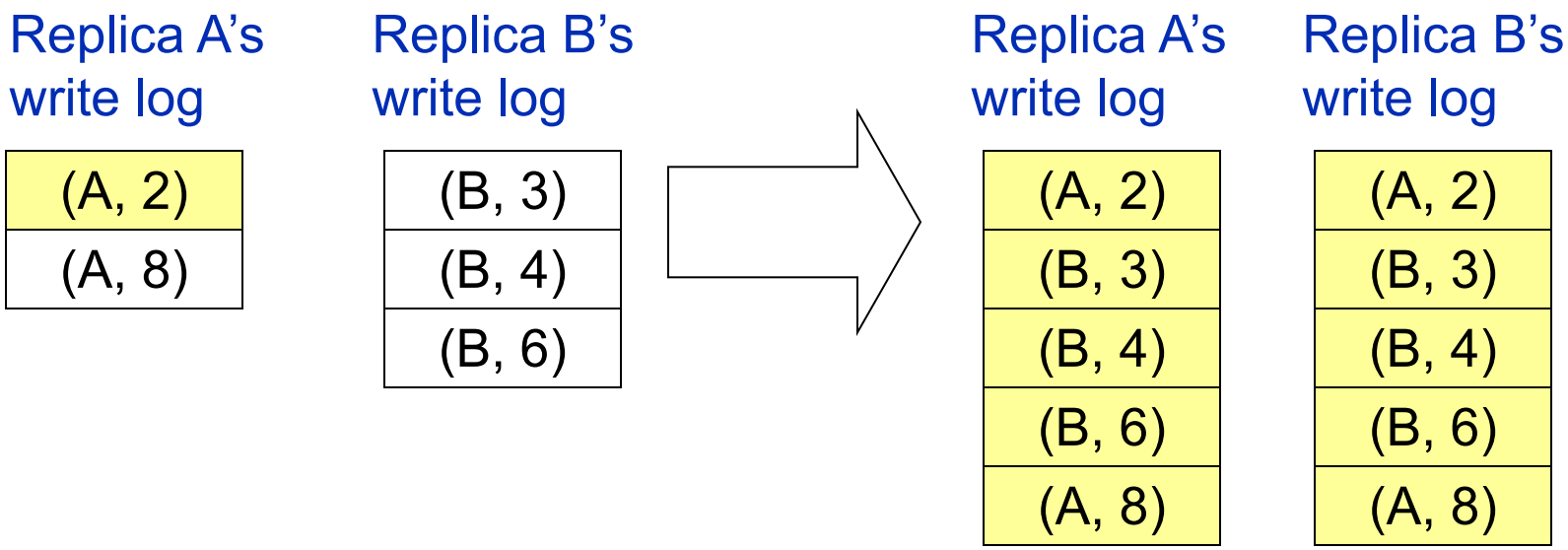
- With eventual consistency, sometimes a user wants to know whether the effect of a write has been finalized
  - Example: Imagine that writes are bookings to airline tickets
- Committed write: Its position in the write log will never change again
  - Its effect is guaranteed – if the write returns OK, you will get the ticket
- Tentative write: Its final position is still to be determined
  - Its effect may change

Example


Before Gossiping
After Gossiping
Replica A's write log
Replica B's write log
(A, 2)
(A, 8)
(B, 3)
(B, 4)
(B, 6)
committed
tentative
Replica A's write log
Replica B's write log
(A, 2)
(B, 3)
(B, 4)
(B, 6)
(A, 8)
(A, 2)
(B, 3)
(B, 4)
(B, 6)
(A, 8)
writes ordered by logical clock values

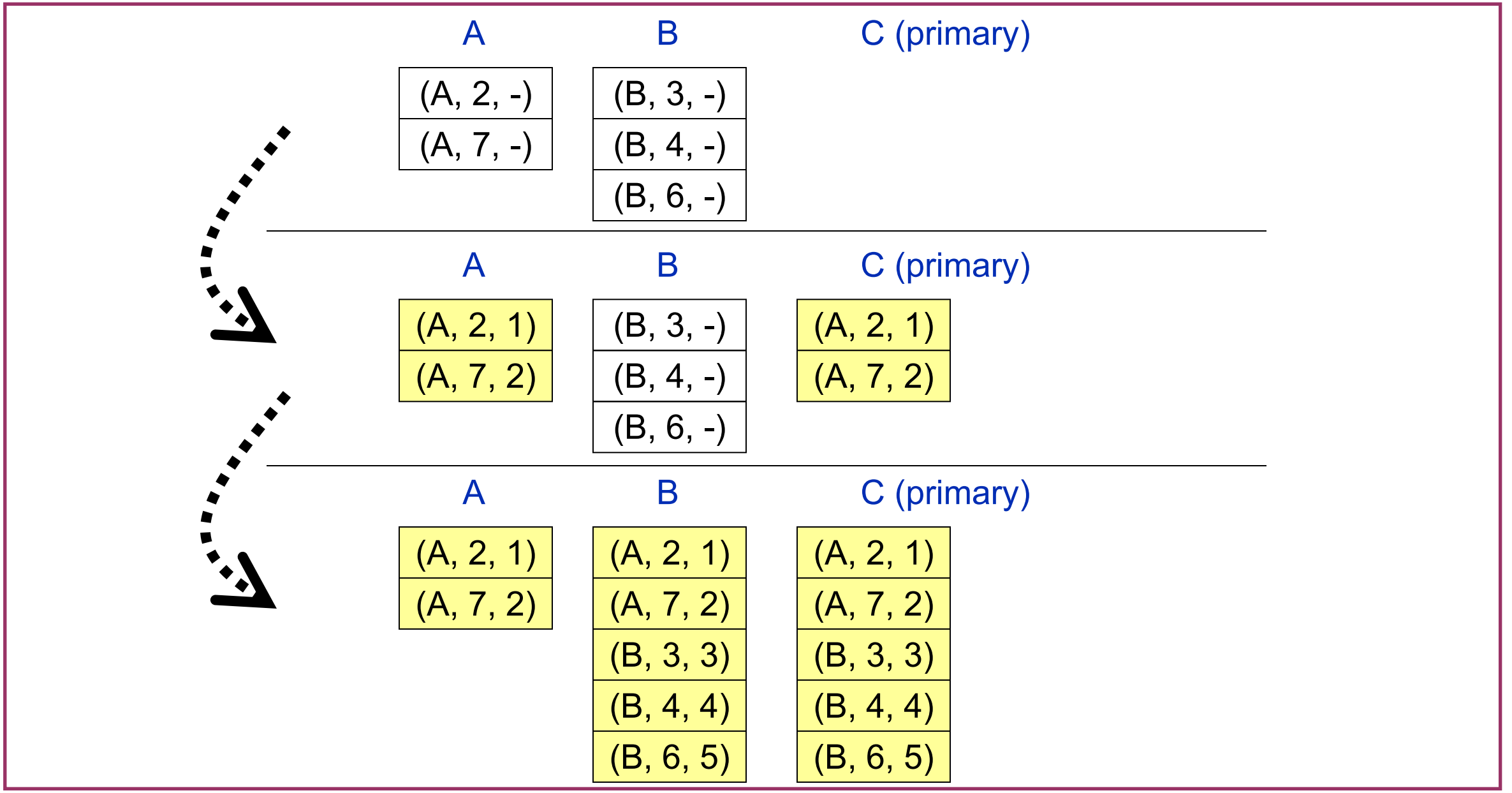
Detecting “Holes” in Serialization Order

- Determining whether a write is committed or tentative is surprisingly hard
  - How can Replica A be sure that (A,2) is a committed write?



Use a Sequencer to Detect "Holes"

- Use a distinguished/primary replica as serialization point or sequencer
- Assign consecutive integers as "commit number" for writes
  - "commit number" = position in the final serialization order
- (A, 2, 1)
  - A is the accepting replica
  - 2 is the logical clock value
  - 1 is the commit number


A
B
C (primary)
(A, 2, -)
(A, 7, -)
(B, 3, -)
(B, 4, -)
(B, 6, -)
A
B
C (primary)
(A, 2, 1)
(A, 7, 2)
(B, 3, -)
(B, 4, -)
(B, 6, -)
(A, 2, 1)
(A, 7, 2)
A
B
C (primary)
(A, 2, 1)
(A, 7, 2)
(A, 2, 1)
(A, 7, 2)
(B, 3, 3)
(B, 4, 4)
(B, 6, 5)
(A, 2, 1)
(A, 7, 2)
(B, 3, 3)
(B, 4, 4)
(B, 6, 5)

History Readings (Non-compulsory)

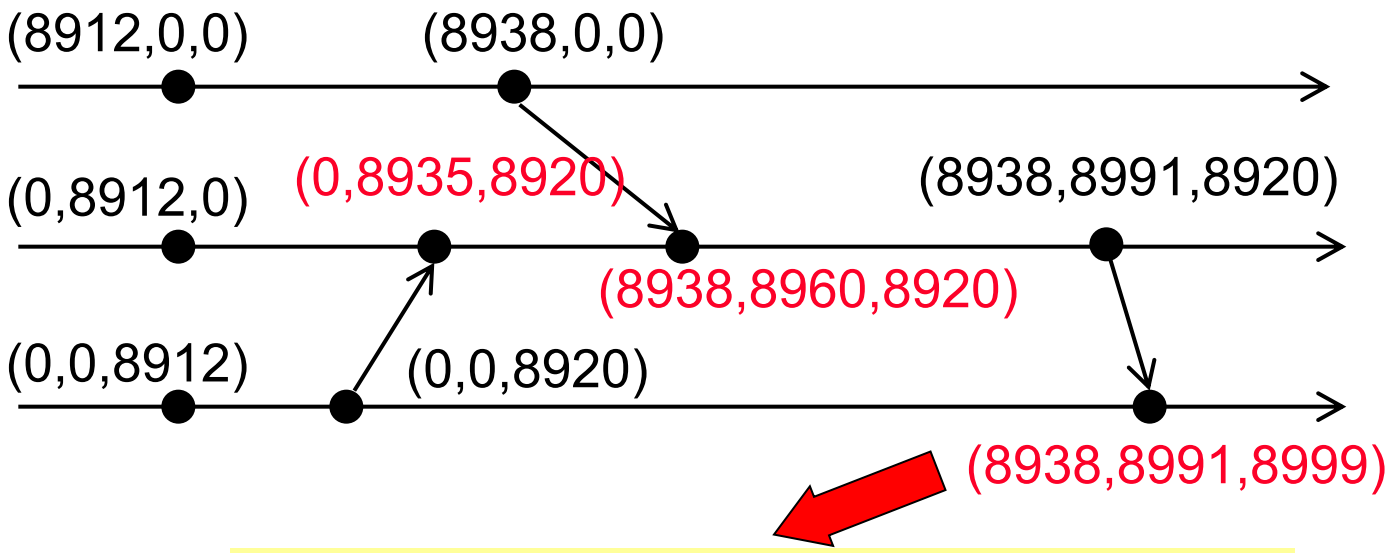
- “Flexible Update Propagation for Weakly Consistent Replication” by K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. In Proceedings of the 16th ACM Symposium on Operating Systems Principles, October 5-8, 1997.

Consistency Protocols for Continuous Consistency

- Order error = # tentative writes
  - Any protocol for reducing number of tentative writes can be used for reducing order error
  - We discusses a protocol already…

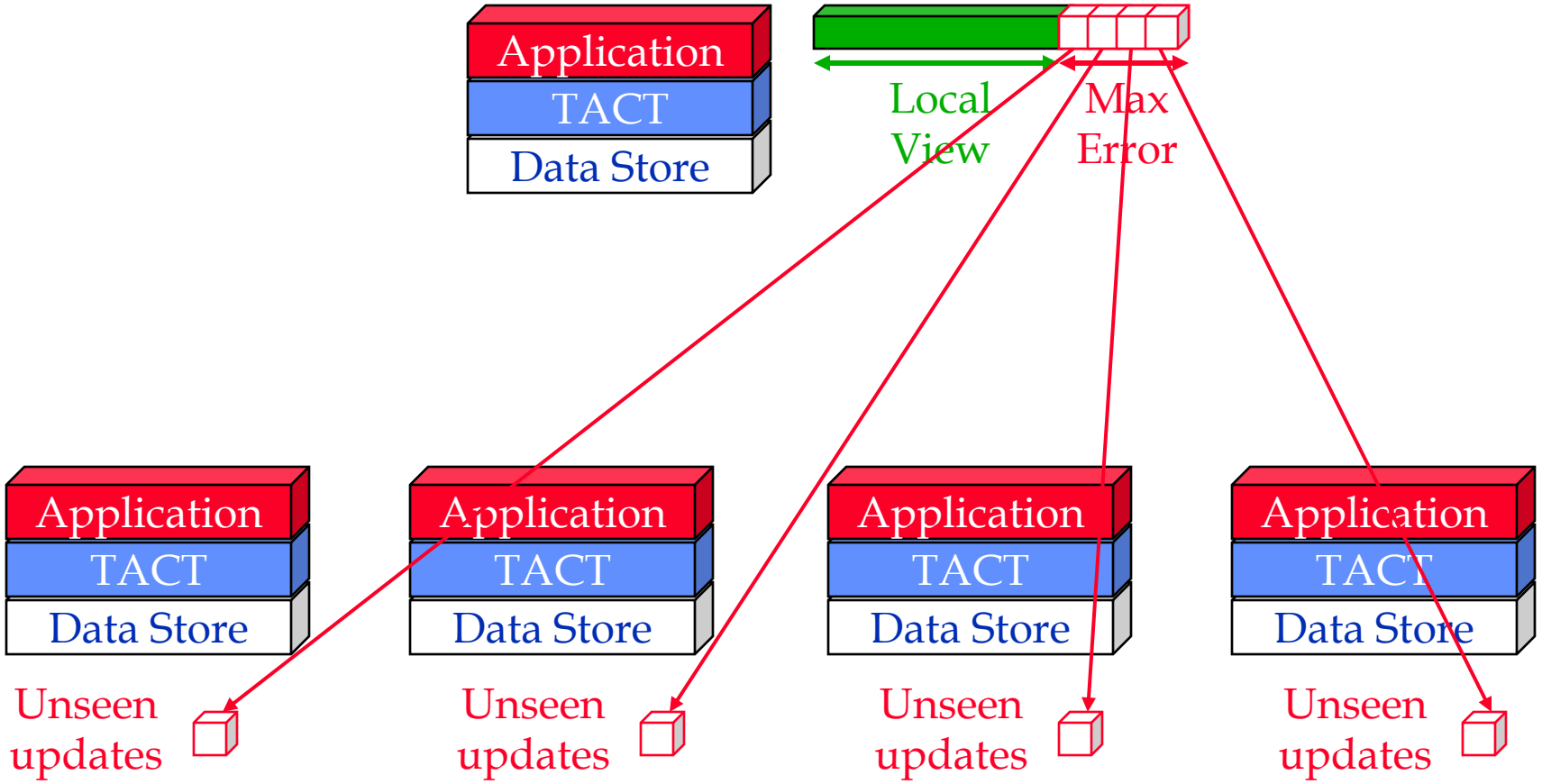
Consistency Protocols for Continuous Consistency

- Temporal error = staleness of the data
- Idea:
  - Real-time vector clock – similar as vector clock
  - Assume loosely synchronized clocks

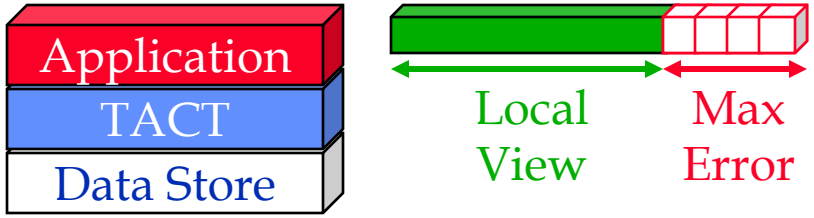


I have seen everything before real-time 8938 ⇒
staleness = current_time – 8938

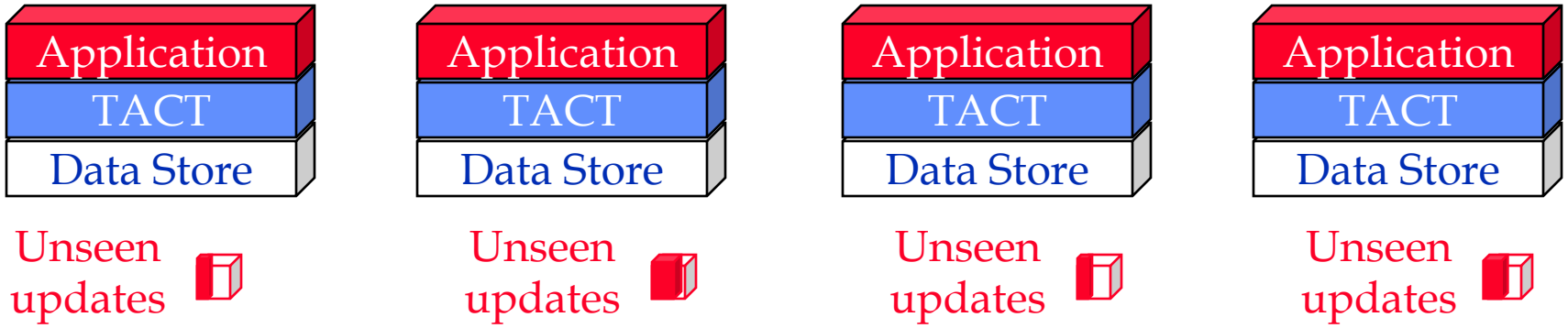
Numerical Error Bounding Intuition



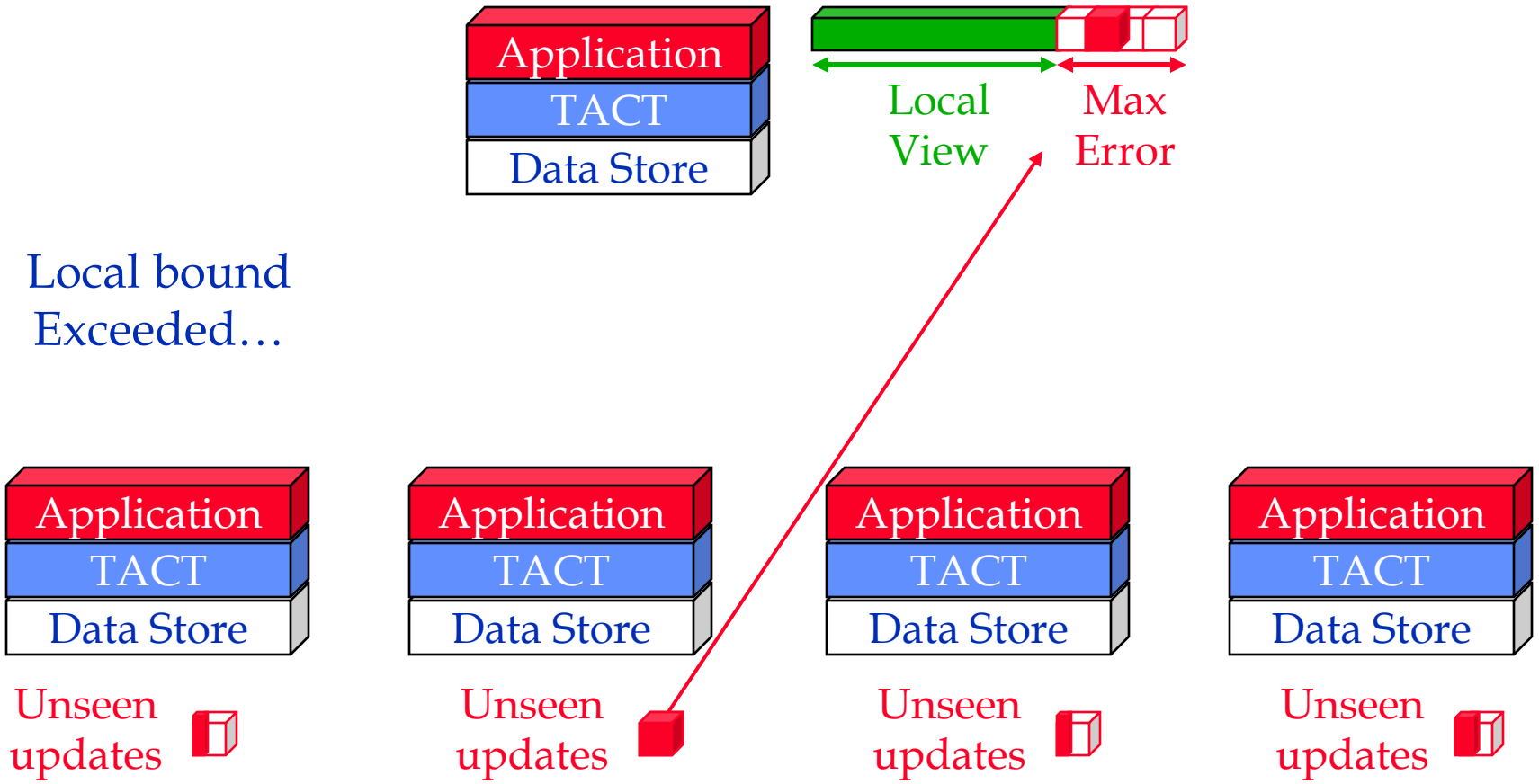
Numerical Error Bounding Intuition



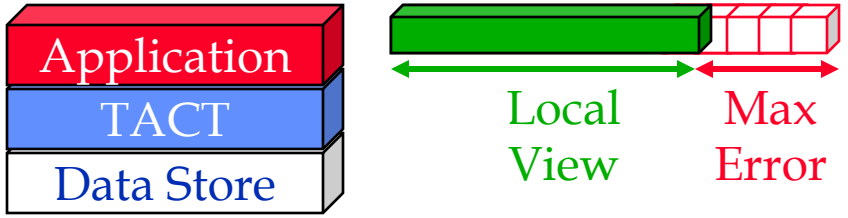
Incoming updates…



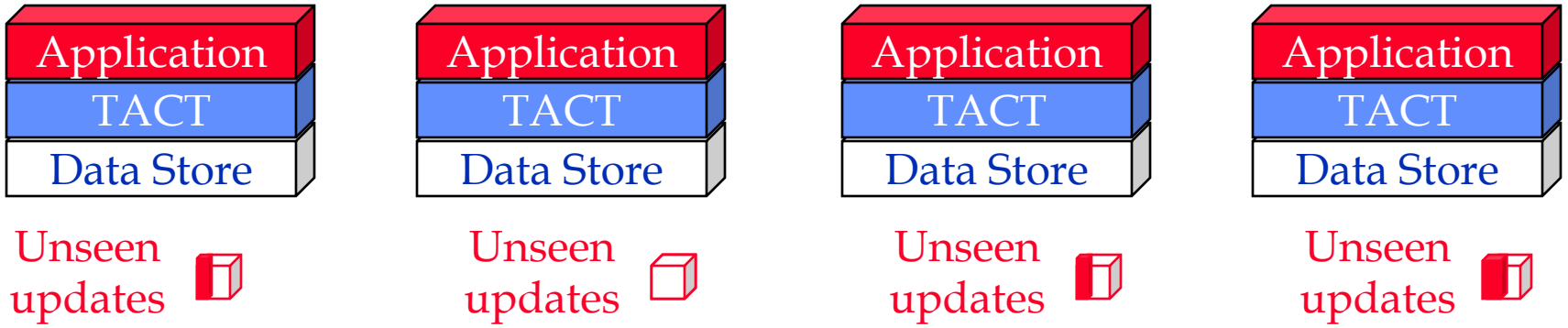
Numerical Error Bounding Intuition



Numerical Error Bounding Intuition



Push updates



Replica Placement

- How many replicas do we use?
  - Each replica comes with a cost
  - Does the benefit justify the cots?

- Where should each replica be?
  - Always want to be close to clients
  - But clients are distributed $\Rightarrow$ Being close to one necessarily means being far away from the other

Replica Placement

- Huge number of replica placement algorithms for readonly data
  - Web proxy caching being an example target application
  - Replica placement is often NP-hard
  - Most solutions are ad hoc and require global knowledge
  - In practice, only simple heuristics are used
- Replica placement for read/write data is much harder than for readonly data
  - Only small number of algorithms for such data
  - Still NP-hard and solutions are ad hoc

Summary

- Consistency model – specifies what is consistent and what is not
- Consistency protocols – implements a certain consistency model
  - May use different protocols to implement the same model
- Replica placement