



# Feasibility of Consistent, Available, Partition-Tolerant Web Services

Meng Wang

Jingxin Feng

Feb 14, 2011



# Overview

- Background
- Formal Model
- Analysis in Asynchronous Networks
- Analysis in Partially Synchronous Networks
- Conclusion
- Other opinions

# Background

- What do you expect for web services?



## This wiki has a problem

Sorry! This site is experiencing technical difficulties.  
Try waiting a few minutes and reloading.

(Cannot connect to the database server: No working slave server: Unknown error (10.0.2.500))

You can try: searching via Google is the meantime.  
Note that this indexes of our content may be out of date.



Wikipedia  WWW

## Background (cont. )



Conjecture by Eric Brewer, at PODC 2000 :  
It is impossible for a web service to provide  
following three guarantees:

- Consistency
- Availability
- Partition-tolerance



## Background (cont. )

- Consistency - all nodes should see the same data at the same time.
- Availability - node failures do not prevent survivors from continuing to operate
- Partition-tolerance - the system continues to operate despite arbitrary message loss



# Background (cont. )

- CAP Theorem
  - Conjecture since 2000
  - Established as theorem in 2002: Lynch, Nancy, and Seth Gilbert. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, v.33(2), 2002, p. 51-59.



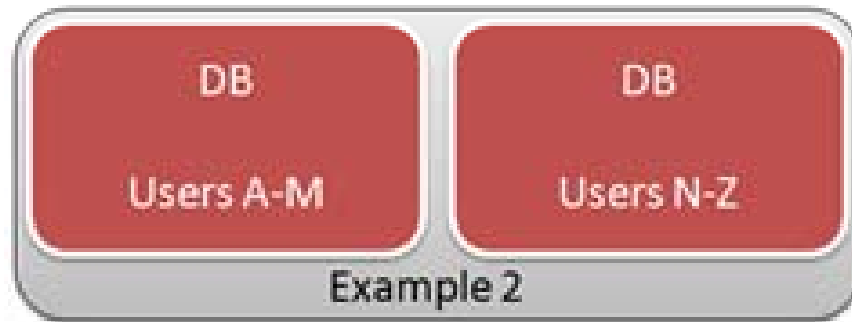
# Formal Model

- Atomic/ Linearizable Data Objects
  - Something like ACID, but not quite...
  - Under this guarantee, there must exist a total order on all operations such that each operation looks as if it were completed at a single instant.

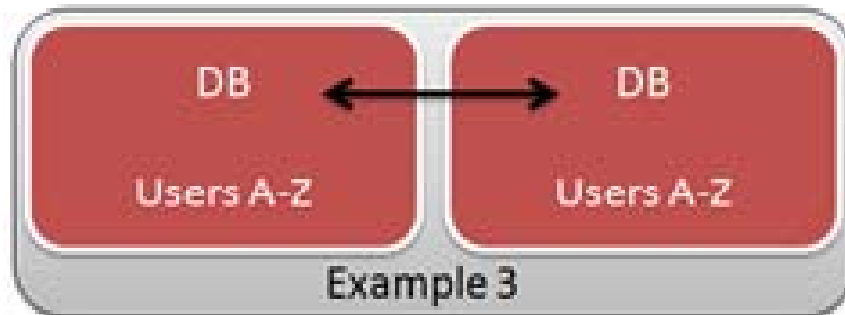
# Formal Model (Cont.)



Consistent



Consistent



Need some work...





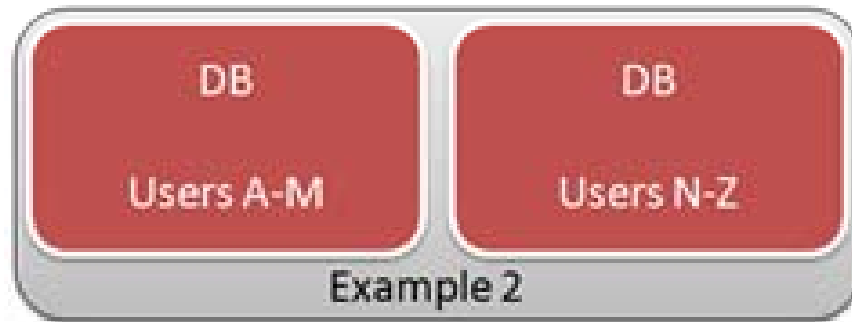
## Formal Model (Cont.)

- Available Data Objects
  - Every request received by a non-failing node in the system must result in a response.
  - That is, any algorithm used by service must eventually terminate.

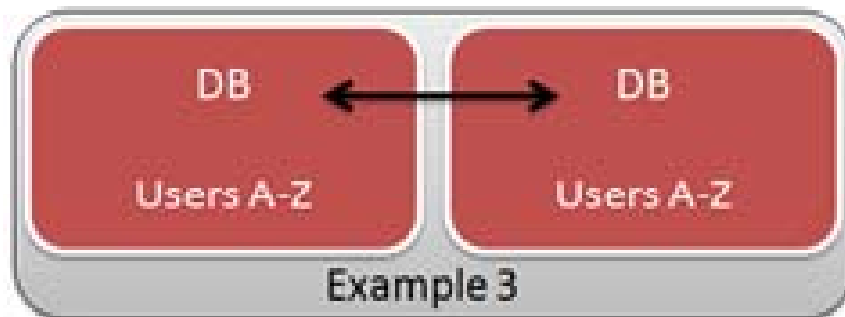
# Formal Model (Cont.)



Not highly available



Not highly available



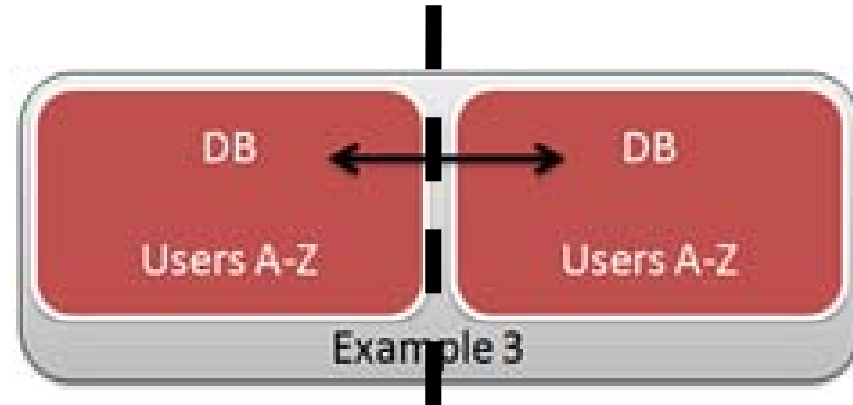
Highly available

# Formal Model (Cont. )

- Partition Tolerance
  - Partition: all messages sent from one node in one component to nodes in another component are lost.
  - Partition Tolerance : No set of failures less than total network failure is allowed to cause the system to respond incorrectly

# Formal Model (Cont.)

- Partition Tolerance



- The atomicity requirement implies that every response will be atomic, even though arbitrary messages sent as part of the algorithm might not be delivered
- The availability requirement therefore implies that every node receiving request from a client must respond, even through arbitrary messages that are sent may be lost
- Can we?



# Asynchronous Network

- Theorem

It is impossible in the asynchronous network model to implement a R/W data object that guarantees the following properties:

- Availability
- Atomic consistency

In all fair executions (including those in which messages are lost.)

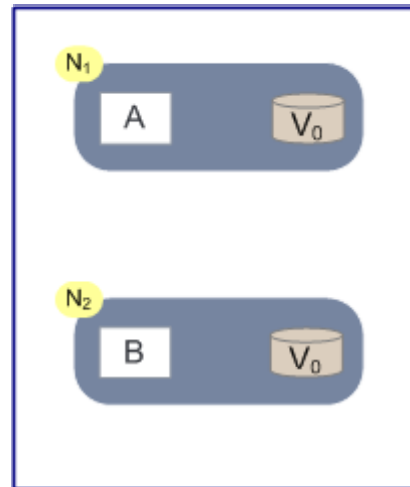
# Asynchronous Network

- There is no clock
- Nodes must make decisions based only on messages received and local computation.



# Asynchronous Network

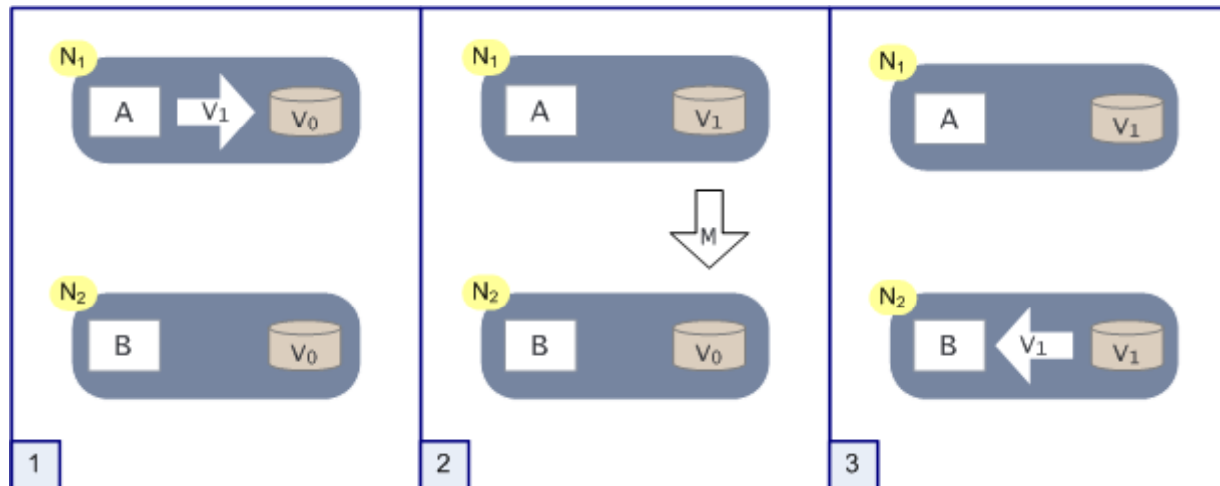
- Data model



The diagram above shows two nodes,  $N_1$  and  $N_2$ . They both share a piece of data  $V$ , which has a value  $V_0$ .  $A$  writes new values of  $V$  and  $B$  reads values of  $V$ .

# Asynchronous Network

- In a sunny day

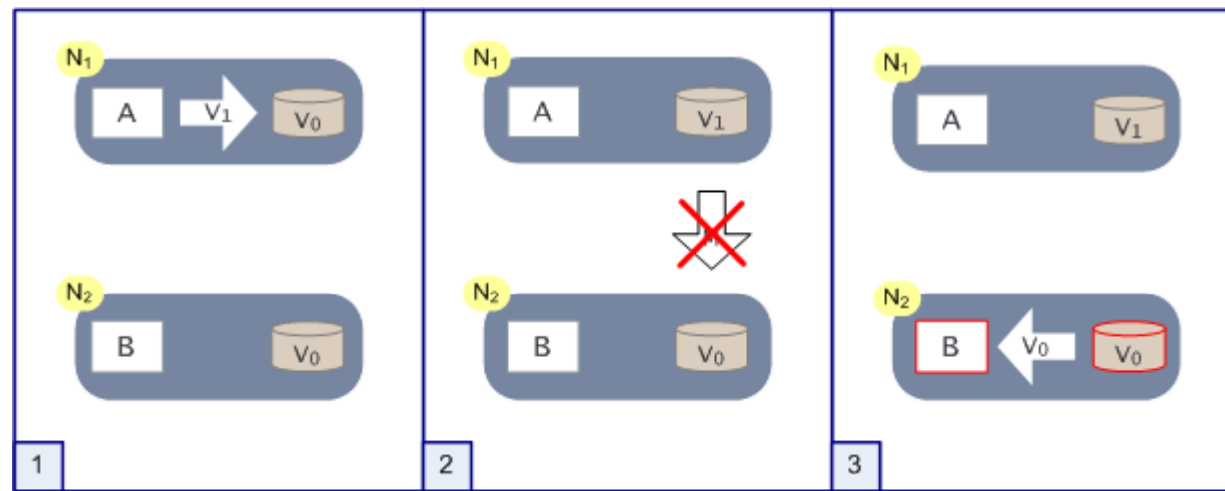


(1) First  $A$  writes a new value of  $V$ , which we'll call  $V_1$ . (2) Then a message ( $M$ ) is passed from  $N_1$  to  $N_2$  which updates the copy of  $V$  there. (3) Now any read by  $B$  of  $V$  will return  $V_1$ .



# Asynchronous Network

- However...



If the network partitions (that is messages from  $N_1$  to  $N_2$  are not delivered) then  $N_2$  contains an inconsistent value of  $V$  when step (3) occurs.

# Asynchronous Network

- Corollary

It is impossible in the asynchronous network model to implement a R/W data object that guarantees the following properties:

- Availability
- Atomic consistency in fair executions in no messages are lost.

Delayed or lost?



# Solution in Asynchronous Network

- Drop partition tolerance

If you want to run without partitions, you have to stop them happening. One way to do this is to put everything (related to that transaction) on one machine.

- Example: Only one node maintains the value of an object. No replicas.



## Solution in Asynchronous Network

- Drop availability
  - Trivial systems: ignores all the requests.
  - Or just wait on encountering a partition event until data is consistent.




# Solution in Asynchronous Network

- Drop Consistency
  - Trivial systems: just return what you have now...
  - Or “Eventually Consistent”



# Partially Synchronous Network Model

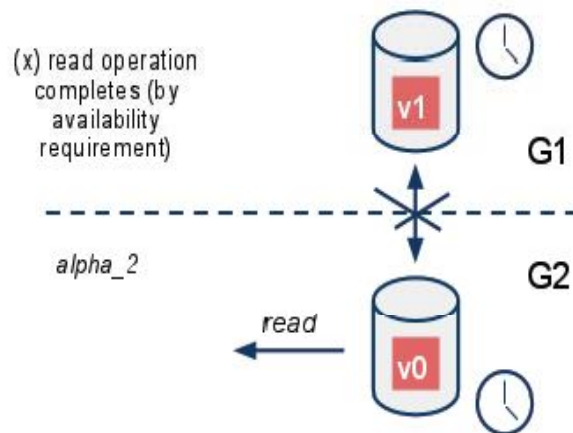
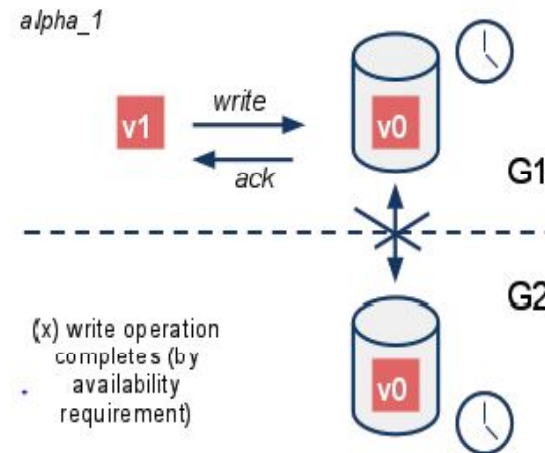
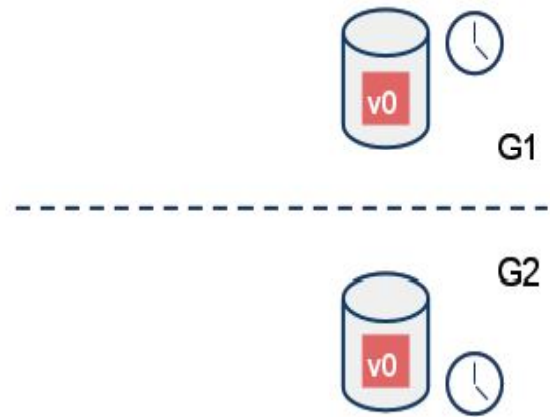
- In the real world, most networks are not purely asynchronous
- In partially synchronous model – every node has a clock and all clocks increase(roughly) at the same rate
- Assume that every message is either delivered within a given, known time  $T_{msg}$  or it is lost
- Also, every node processes a received message within a given, known time  $T_{local}$  and local processing time is 0.



## Partially Synchronous Networks : Impossibility Result

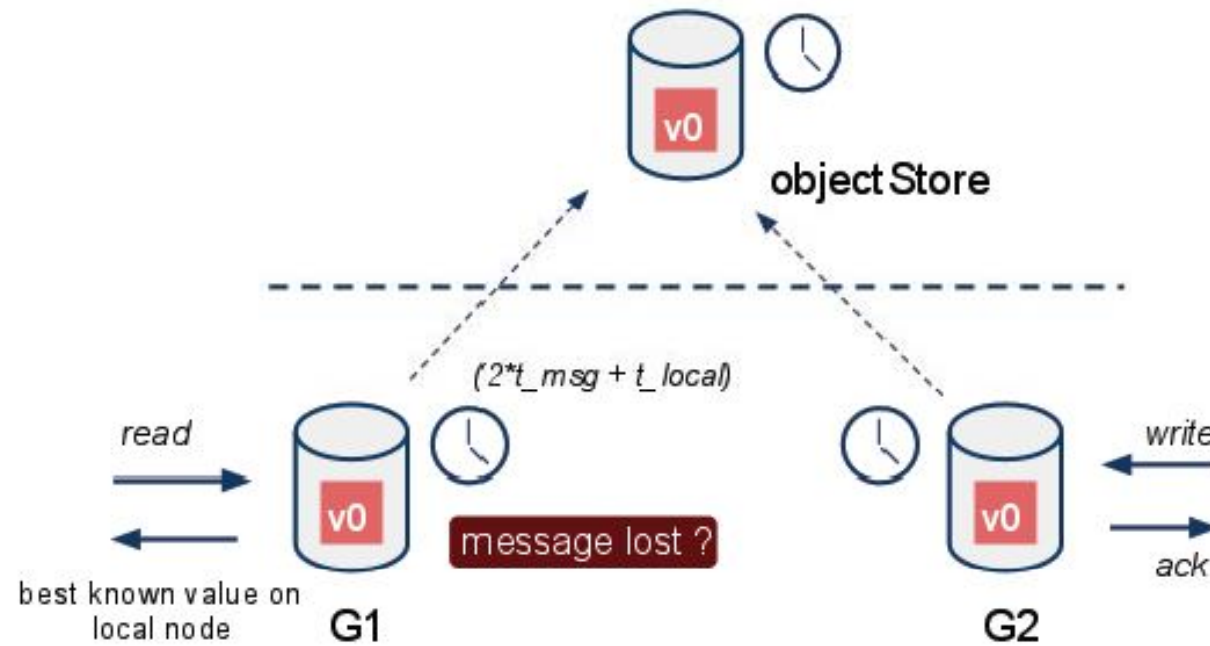
- Theorem 2: It is impossible in the partially synchronous network model to implement a read/write data object that guarantees the following properties:
  - Availability
  - Atomic consistencyin all executions (even those in which messages are lost)

# Proof Partially Synchronous Networks

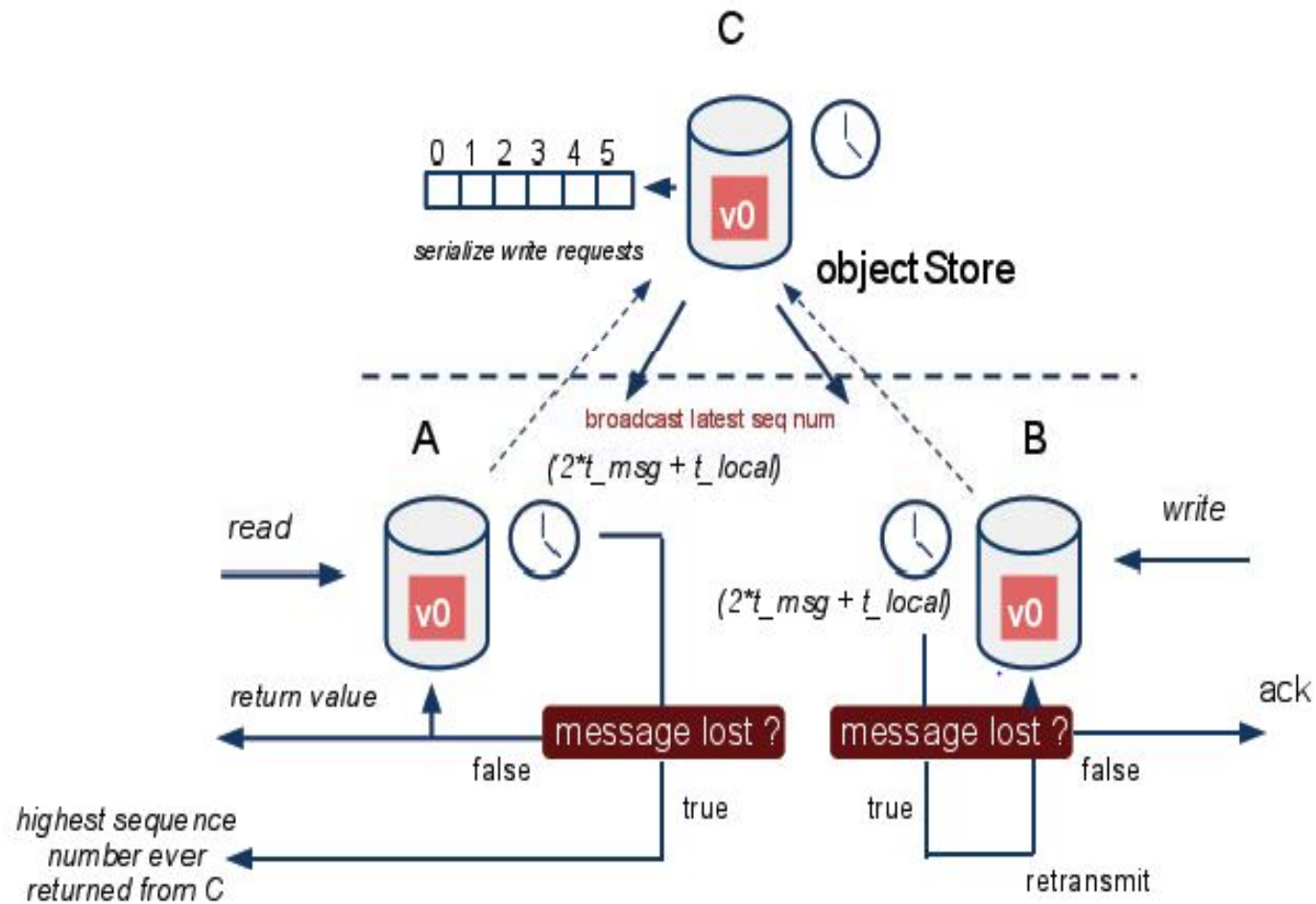




# Solutions in the Partially Synchronous Model

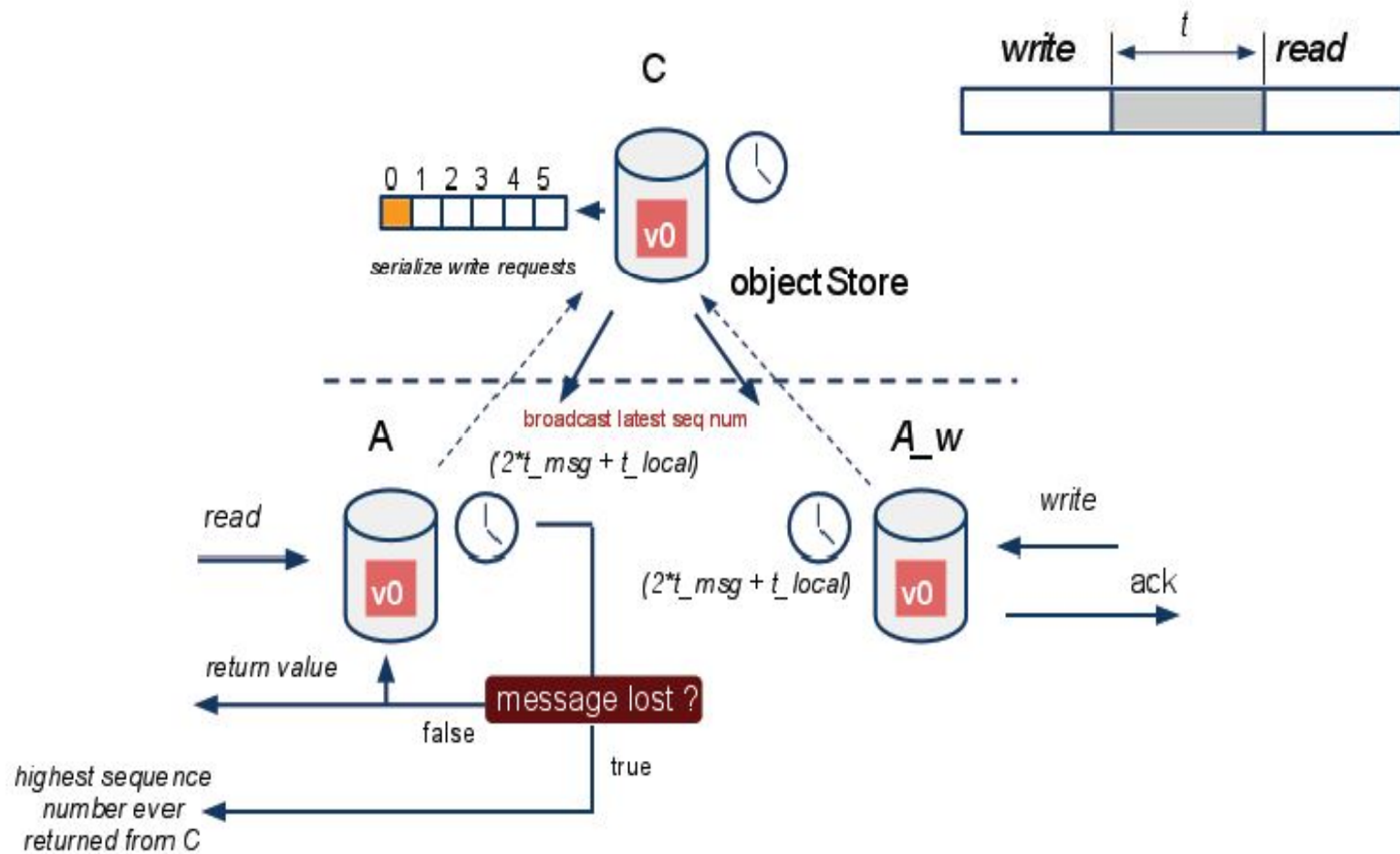


# Weaker Consistency Conditions



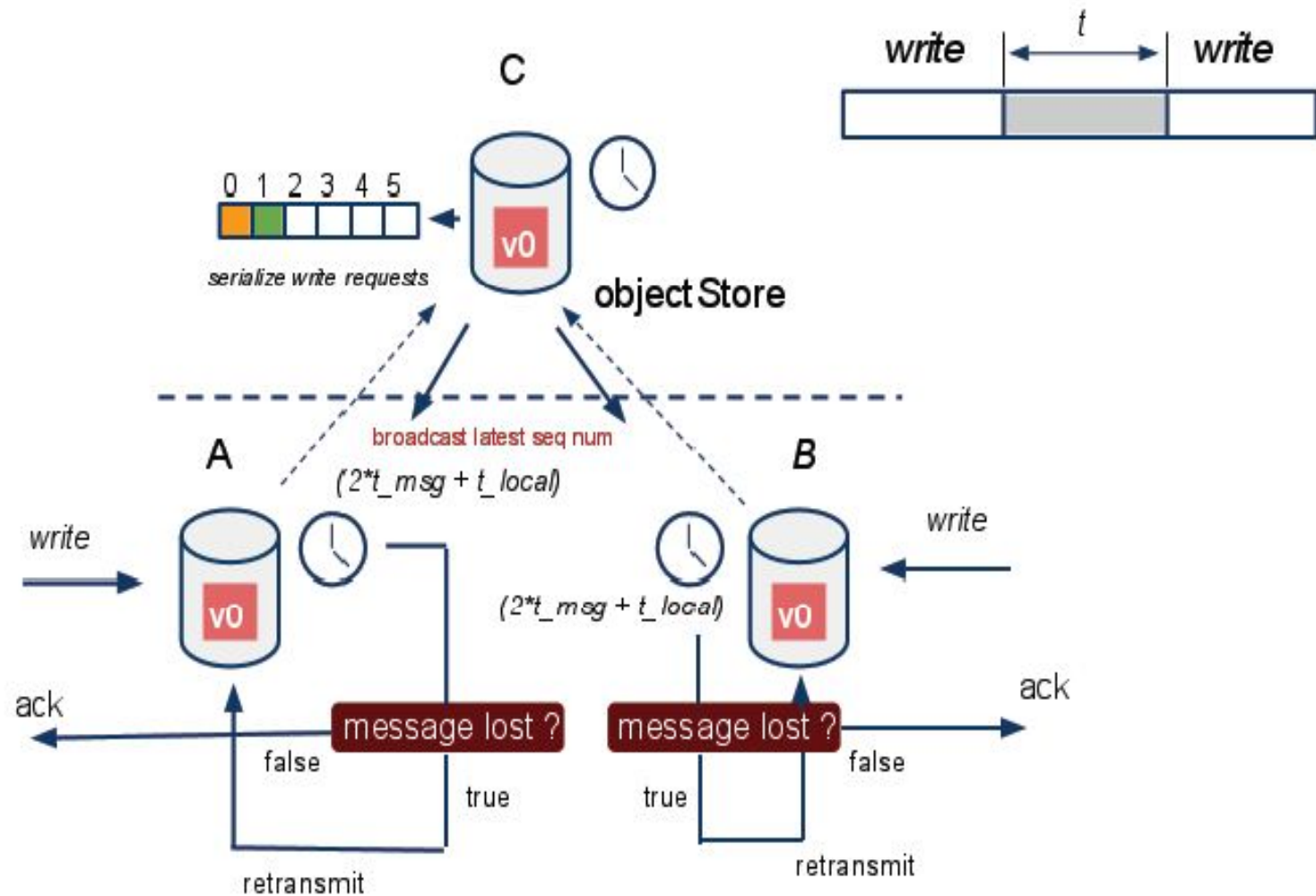
# Weaker Consistency Conditions

## write followed by read



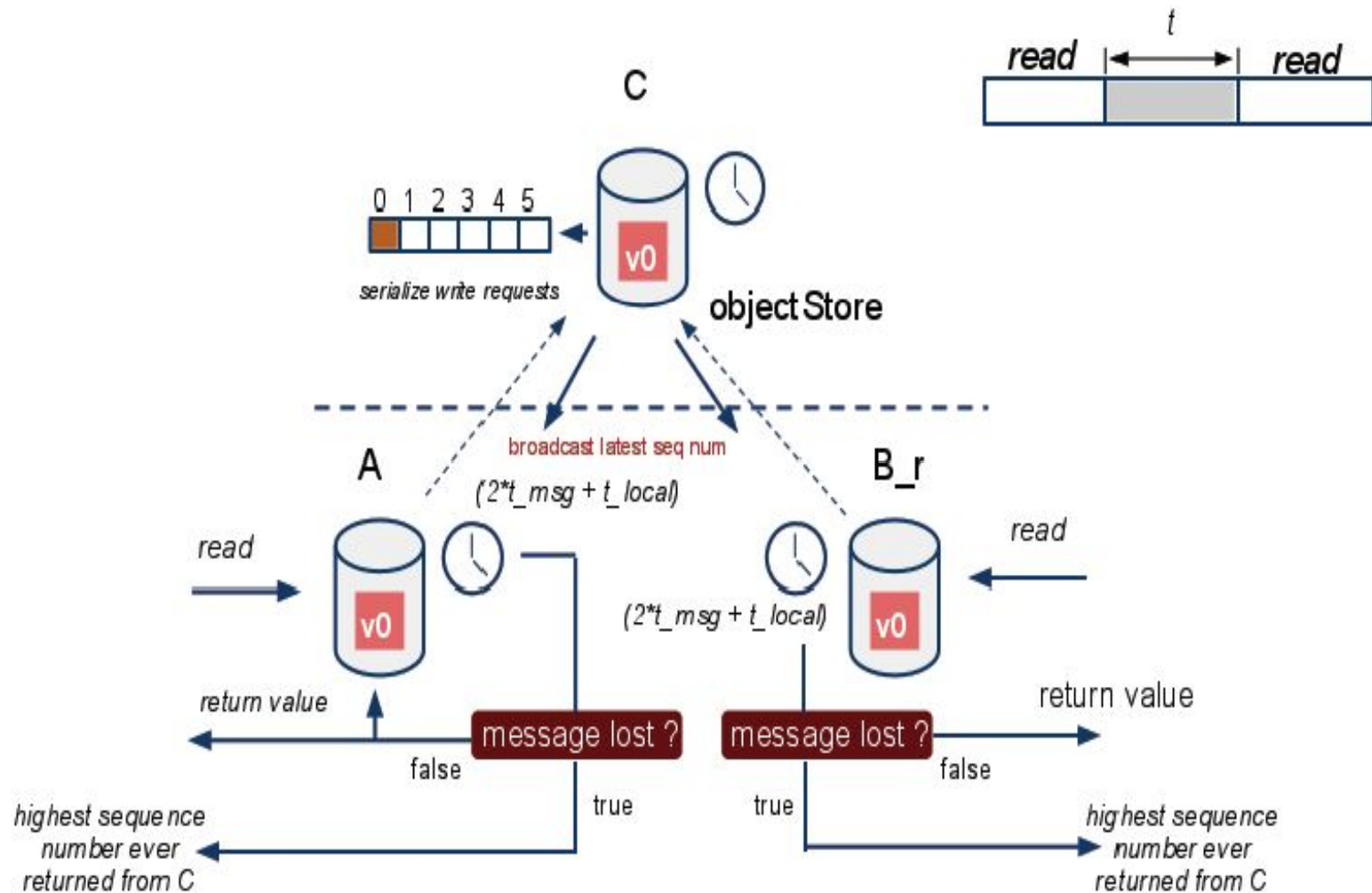
# Weaker Consistency Conditions

## write followed by write



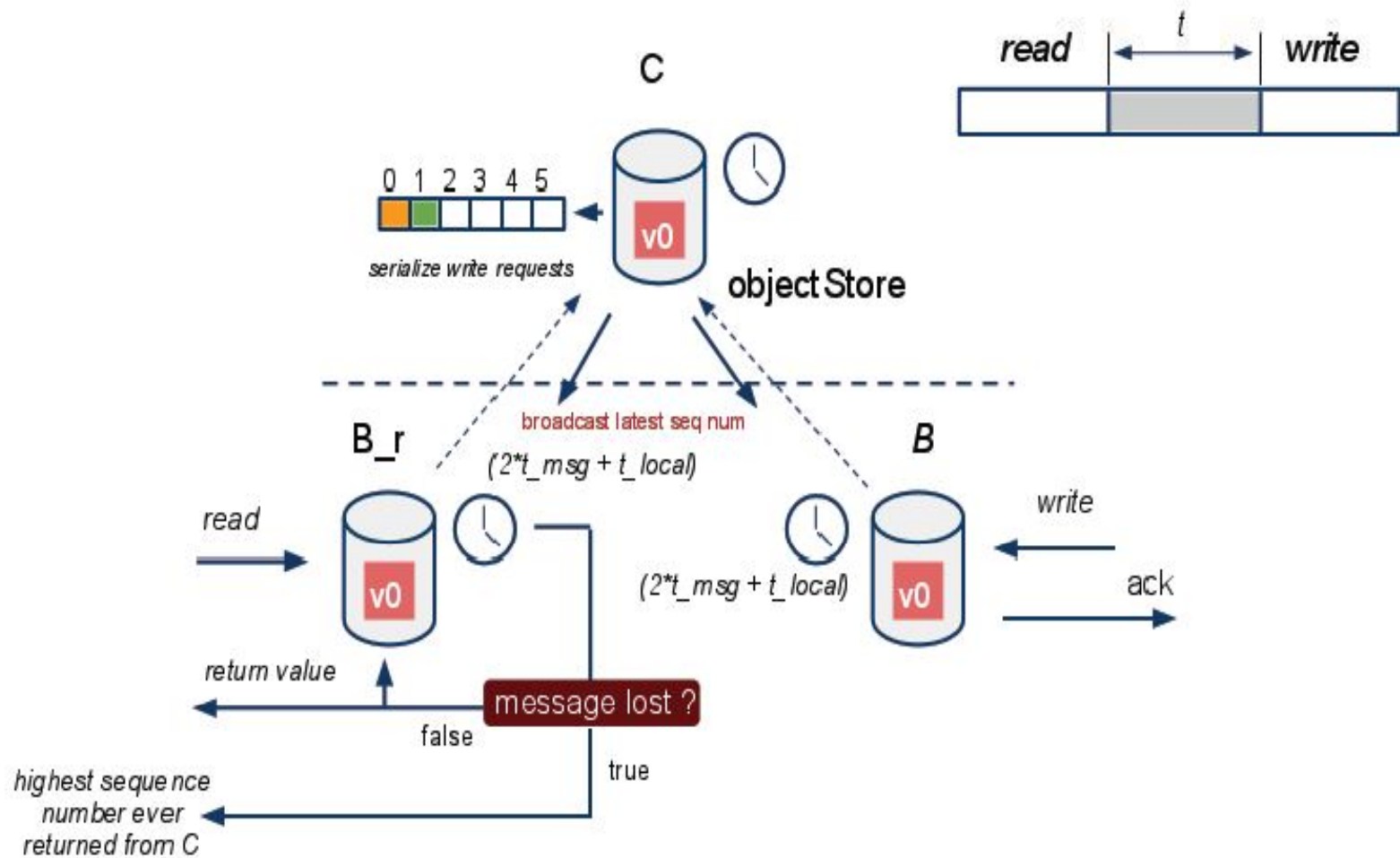
# Weaker Consistency Conditions

## read followed by read



# Weaker Consistency Conditions

## read followed by write

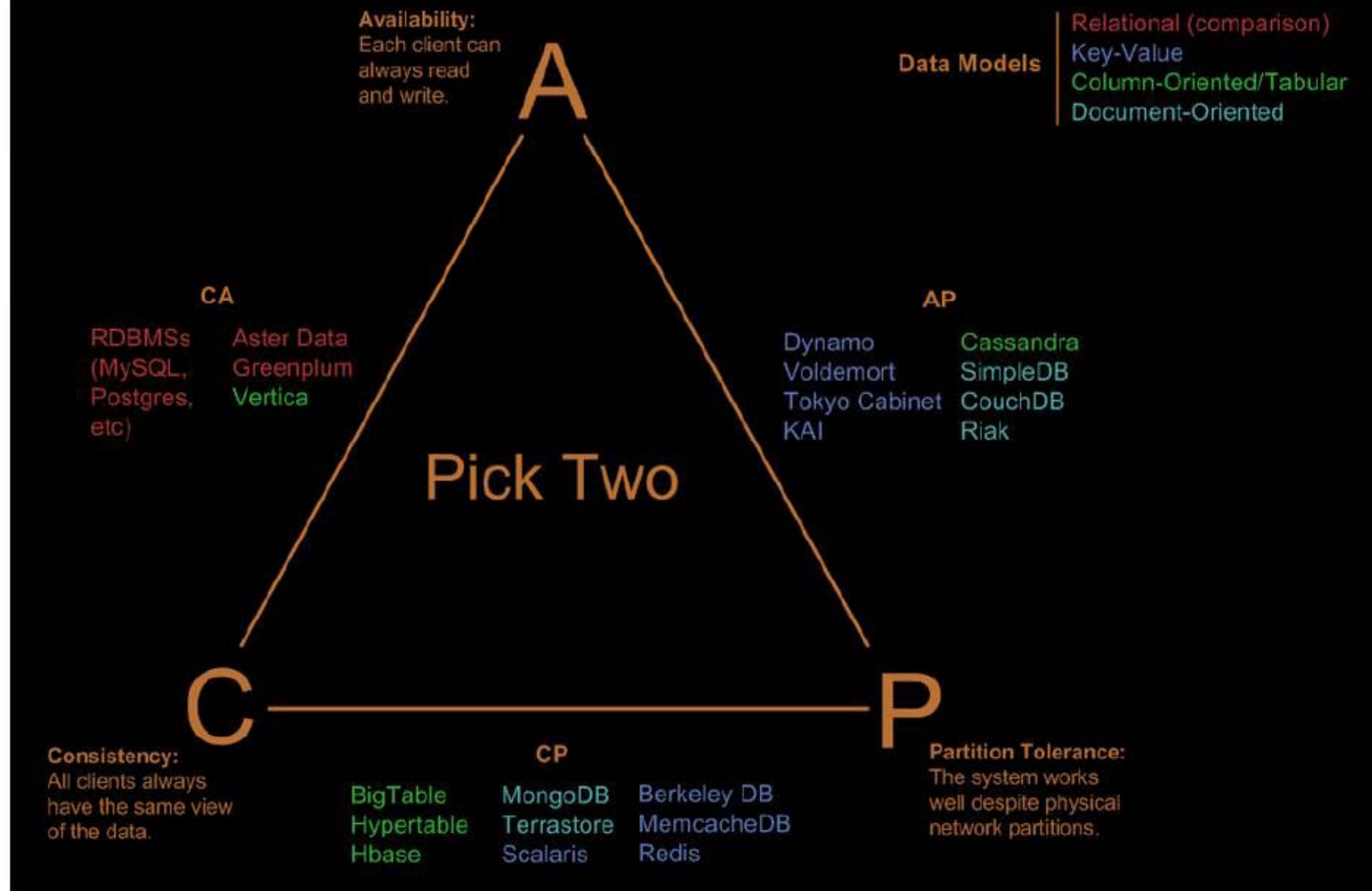




# Conclusion

- It is impossible to reliably provide atomic, consistent data when there are partitions in the network.
- It is feasible, however, to achieve any two of the three properties.
- In partially synchronous modes, it is possible to achieve a practical compromise between C and A.

# Visual Guide to NoSQL Systems







## Other opinions

- In the NoSQL community, this theorem has been used as the justification for giving up consistency.
- Eventually consistency, i. e., when network connectivity has been re-established and enough subsequent time has elapsed for replica cleanup. The justification for giving up C is so that the A and P can be preserved.

# Other opinions



- Michael Stonebraker
  - The CAP Theorem analysis is suspect, and that recovery from errors has more dimensions to consider.



# Errors in database

- 1. Application errors.
  - The application performed one or more incorrect updates.
  - Generally, this is not discovered for minutes to hours thereafter.
  - The database must be backed up to a point before the offending transaction(s), and subsequent activity redone.

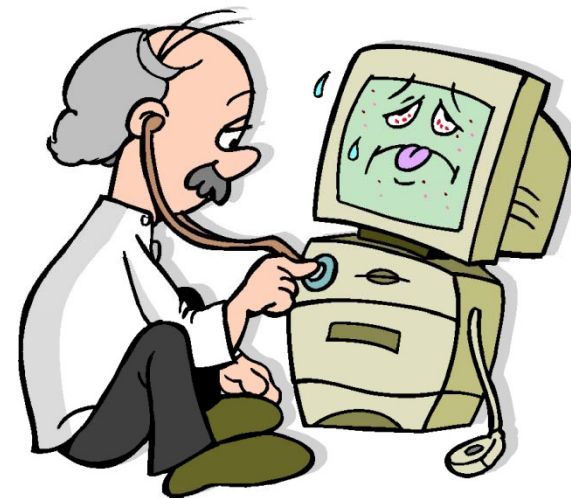
# Errors in database

- 2. Repeatable DBMS errors
  - The DBMS crashed at a processing node.
  - Executing the same transaction on a processing node with a replica will cause the backup to crash.



# Errors in database

- 3. Unrepeatable DBMS errors
  - The database crashed, but a replica is likely to be ok.



# Errors in database

- 4. Operating system errors.
  - The OS crashed at a node, generating the “blue screen of death.”



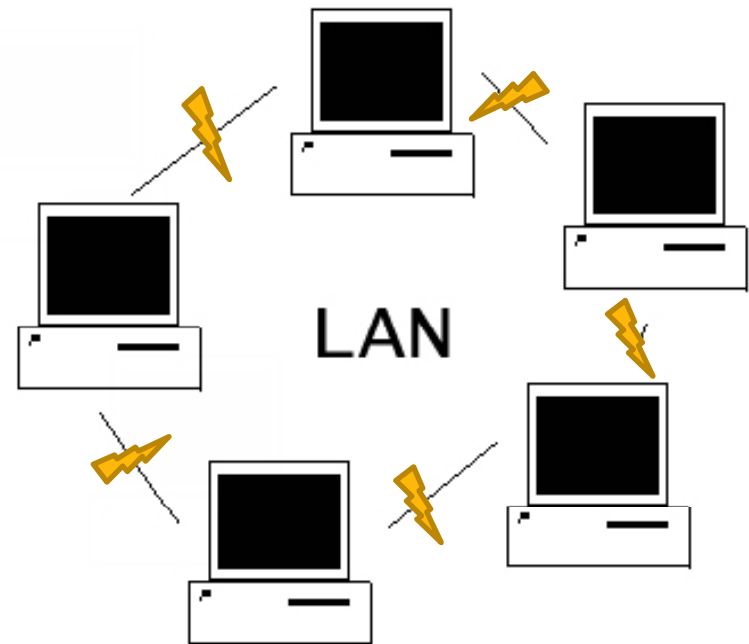


# Errors in database

- 5. A hardware failure in a local cluster.
  - These include memory failures, disk failures, etc. Generally, these cause a “panic stop” by the OS or the DBMS.
  - However, sometimes these failures appear as (3) Unrepeatable DBMS errors.

# Errors in database

- 6. A network partition in a local cluster
  - The LAN failed and the nodes can no longer all communicate with each other.





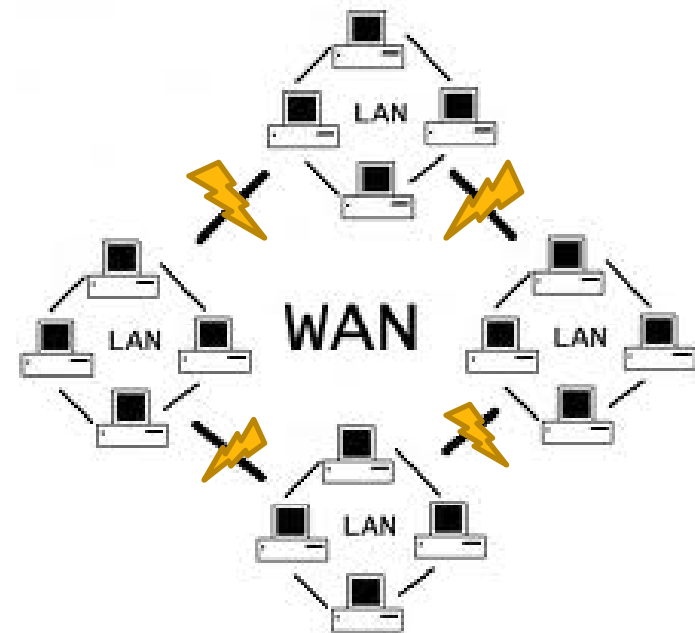
# Errors in database

- 7. A disaster.
  - The local cluster is wiped out by a flood, earthquake, etc. The cluster no longer exists.



# Errors in database

- 8. A network failure in the WAN connecting clusters together.
  - The WAN failed and clusters can no longer all communicate with each other.





# Errors in database

- First, note that app error and repeatable DBMS error will cause problems with any high availability scheme.
- In these two scenarios, there is no way to keep going. Also, replica consistency is meaningless; the current DBMS state is simply wrong.



# Errors in database

- In a disaster, data will only be recoverable if a local transaction is only committed after the assurance that the transaction has been received by another WAN-connected cluster.
- Few application builders are willing to accept this kind of latency.
- The performance penalty for avoiding it is too high, so designers choose to suffer data loss in this situation.



# Errors in database

- As such, errors 1, 2, and 7 are examples of cases for which the CAP theorem simply does not apply. Any real system must be prepared to deal with recovery in these cases. The CAP theorem cannot be appealed to for guidance.



# Errors in database

- A partition in WAN is quite rare
- Moreover, the most likely WAN failure is to separate a small portion of the network from the majority.
- It seems unwise to give up consistency all the time in exchange for availability of a small subset of the nodes in a fairly rare scenario.



# Errors in database

- Lastly, consider a slowdown either in the OS, the DBMS, or the network manager.
- Why? Skew in load, buffer pool issues...
- How to deal with? Fail the offending component?
- No! You push load to others in a high workload situation.
- Solution:
  - one should write software that can deal with load spikes without failing
  - good monitoring software will help identify such problems early
  - self-reconfiguring software that can absorb additional resources quickly



## Other opinions

- In summary, one should not throw out the C so quickly, since there are real error scenarios where CAP does not apply and it seems like a bad tradeoff in many of the other situations.