MDCC: Multi-Data Center Consistency

Authors: Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, Alan Fekete
Presenters: Yan-Yu Huang, Wei-Ting Ho, Po-Hsuan Chen

Outline

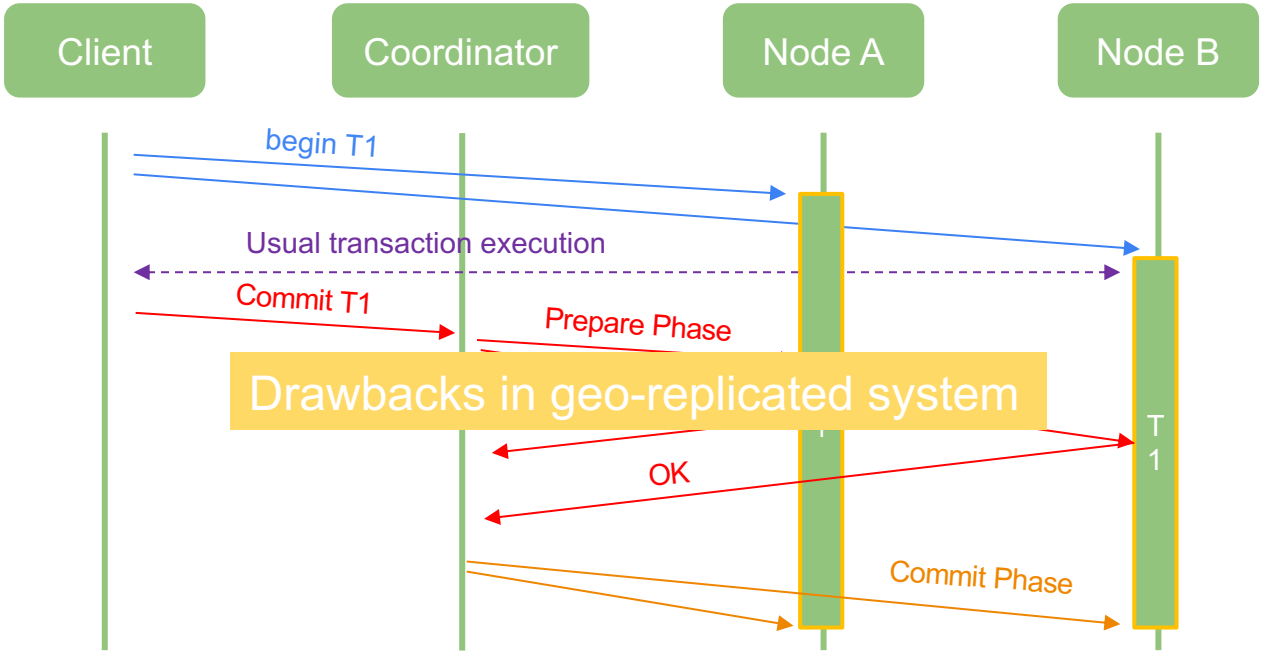
- **Introduction**
- **Classic Paxos**
- **MDCC**
  - Transaction Support
    - Multi-Paxos
  - Transactions Bypassing the Master
    - Fast Paxos
  - Commutative Updates
    - Generalized Paxos
- **Consistency Guarantees**
- **Evaluation**
- **Conclusion**

Introduction

- Databased Backed Application
  - Replicate data across multiple data centers
  - Keep replicas synchronized and consistent
- Geo-Replication
  - High network latency
  - Need to reduce the number of message round-trips

Background– 2 Phase Commit (2PC)

- Two Phases : Prepare Phase + Commit Phase



Background - Paxos

- Google Mega-store / Google Spanner / Paxos-CP
  - Based on Paxos
  - Scalability bottleneck

- Still rely on 2 phase commit
- Require 2 message rounds

MDCC (Multi-Data Center Consistency)

an optimistic commit protocol for geo-replicated transactions.

MDCC - Multi-Data Center Consistency

- Requires only 1 message round
- Strong consistency
- Low latency

Classic Paxos

Paxos Basics

What is Paxos?

Achieving consensus among replicas

Features of Paxos

- Tolerate failures
  - Lost, duplicated, re-ordered messages
- Not consider Byzantine problem

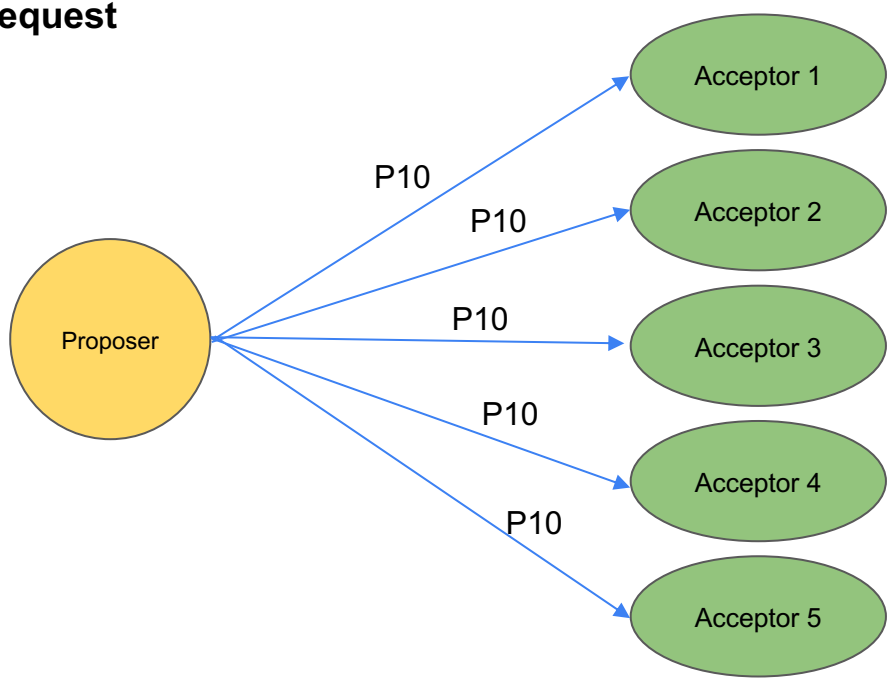
Paxos Basics

Paxos define four roles:

- Clients (app-servers)
- Proposers (masters)
- Acceptors (storage nodes)
- Learners (all nodes)

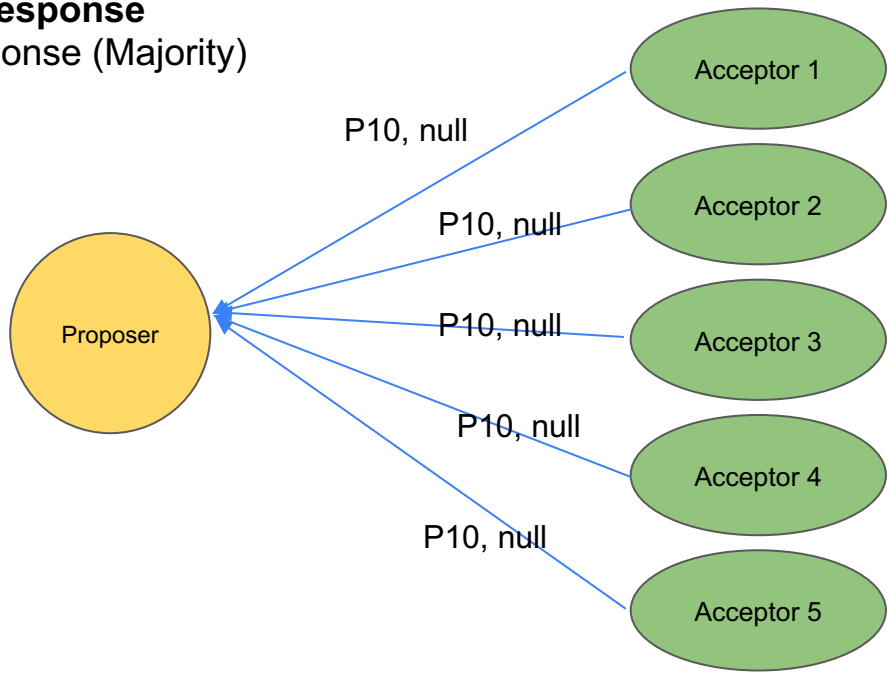
Classic Paxos - Prepare phase

Proposer sends prepare request



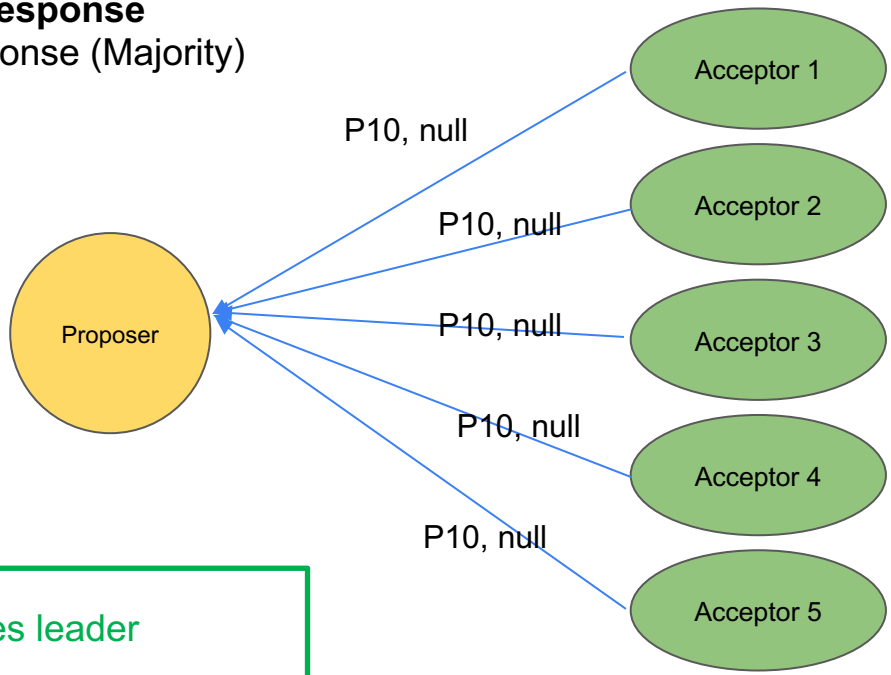
Classic Paxos - Prepare phase

Acceptors send prepare response
More than 3 Acceptors response (Majority)



Classic Paxos - Prepare phase

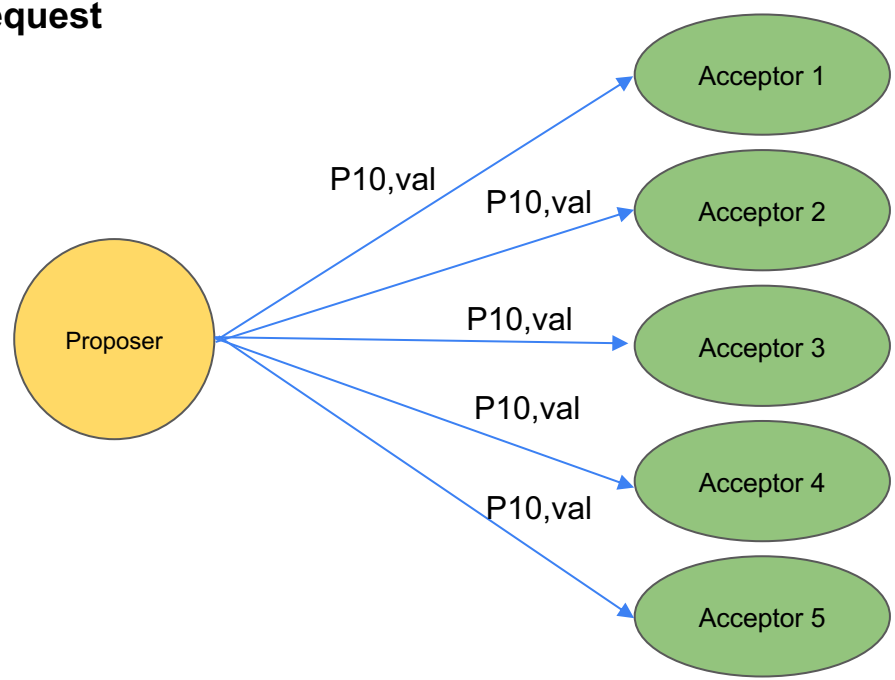
Acceptors send prepare response
More than 3 Acceptors response (Majority)



Proposer becomes leader

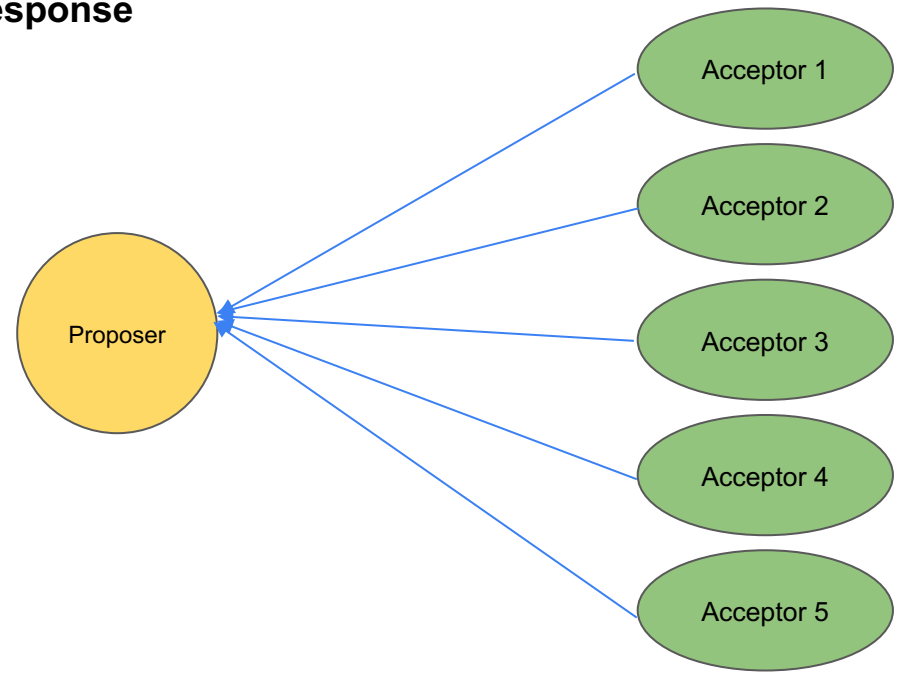
Classic Paxos - Accept phase

Proposer sends accept request



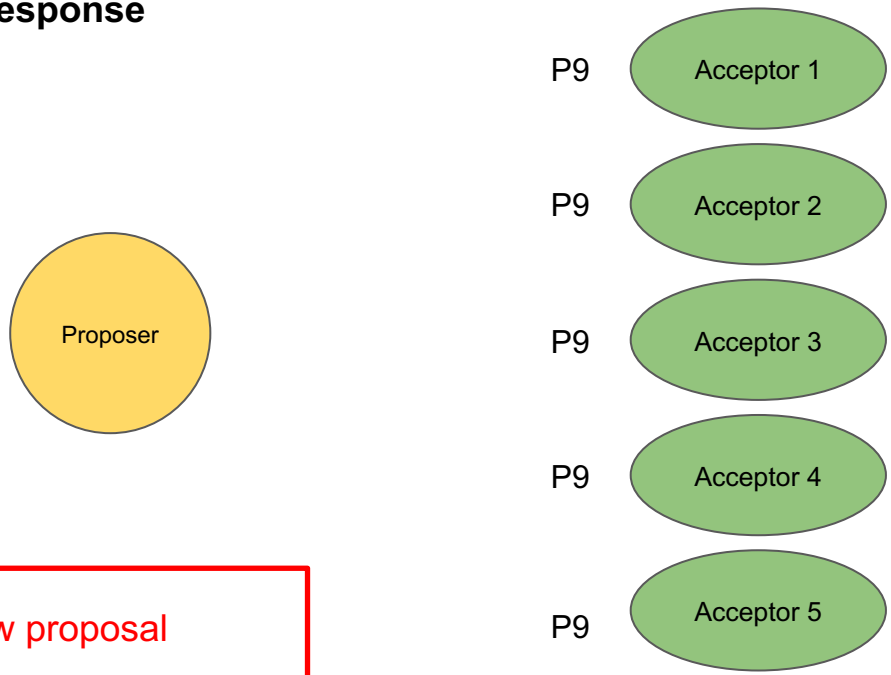
Classic Paxos - Accept phase

Acceptors send accept response



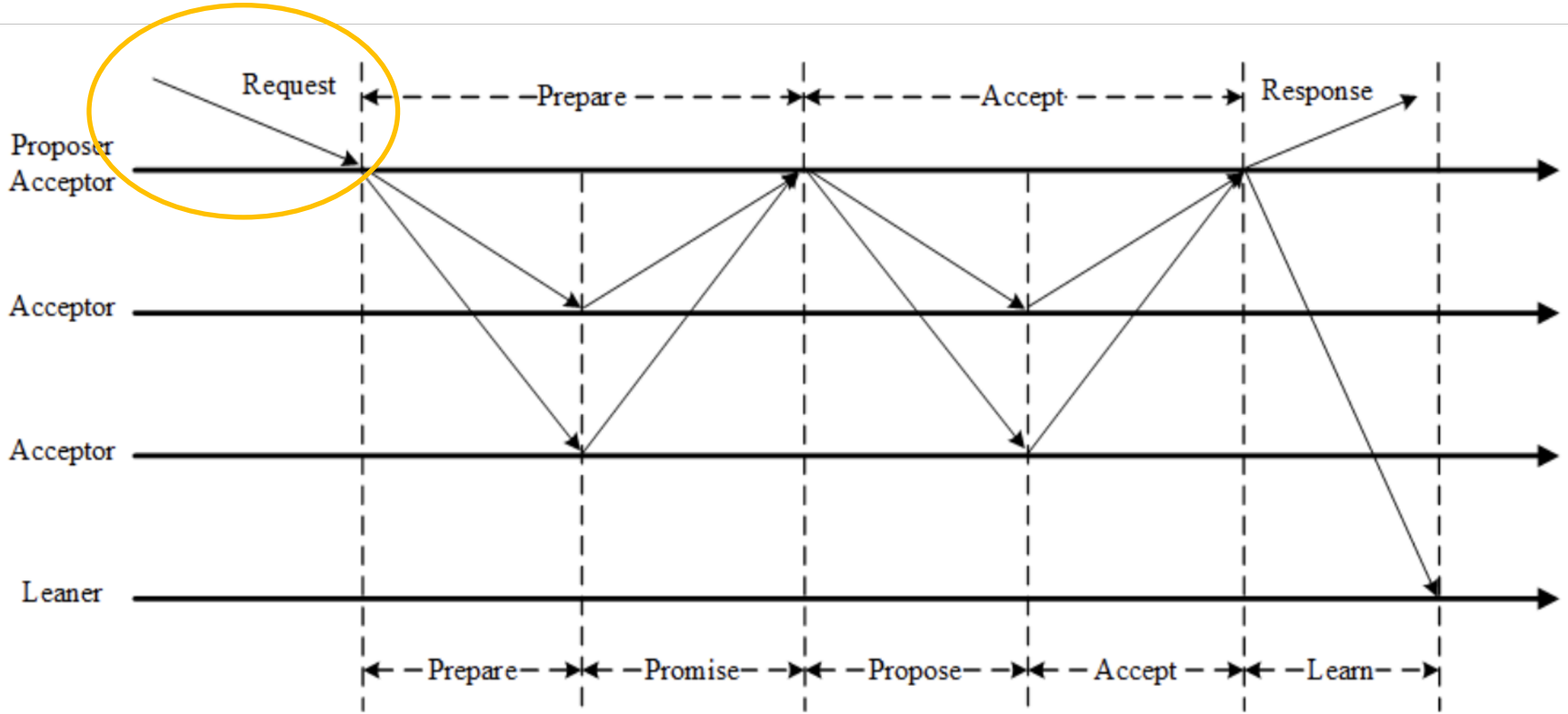
Classic Paxos - Prepare phase

Acceptors send prepare response

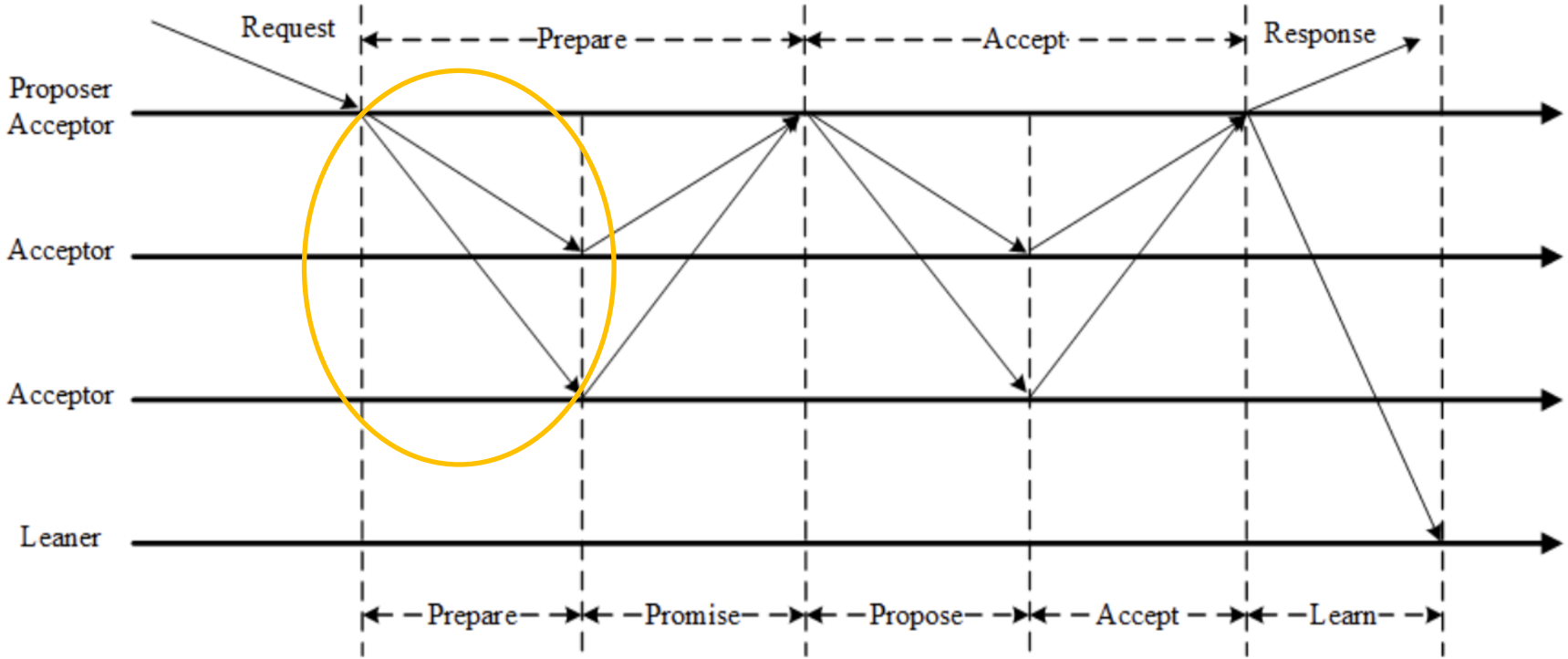


Time out! Send new proposal

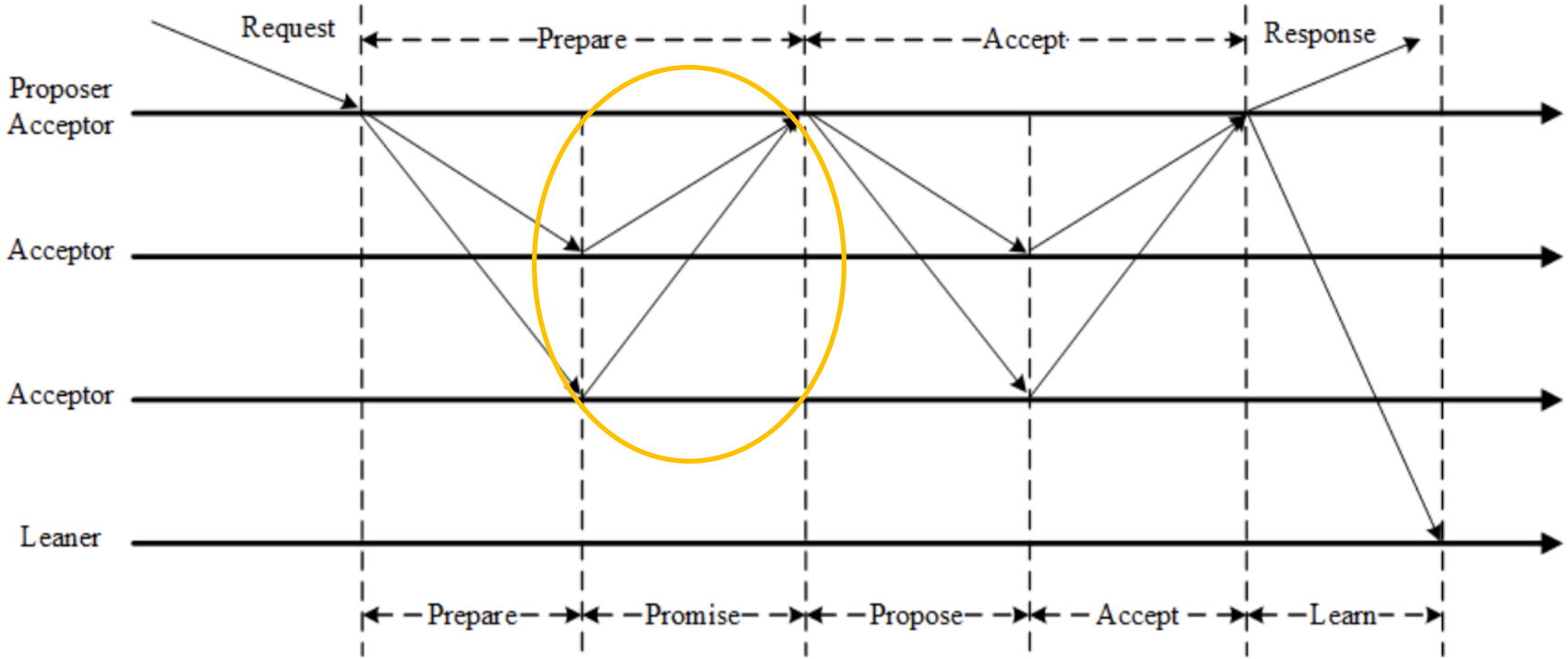
Classic Paxos



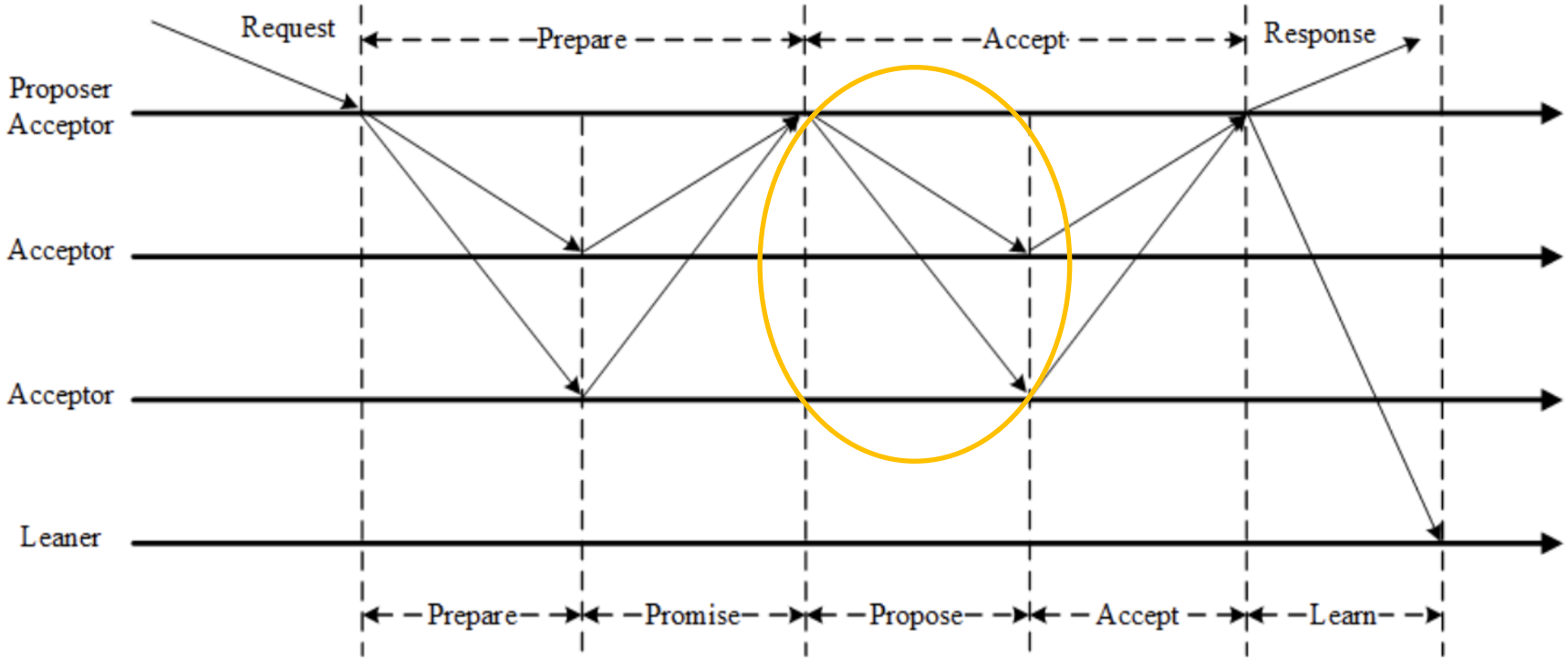
Classic Paxos



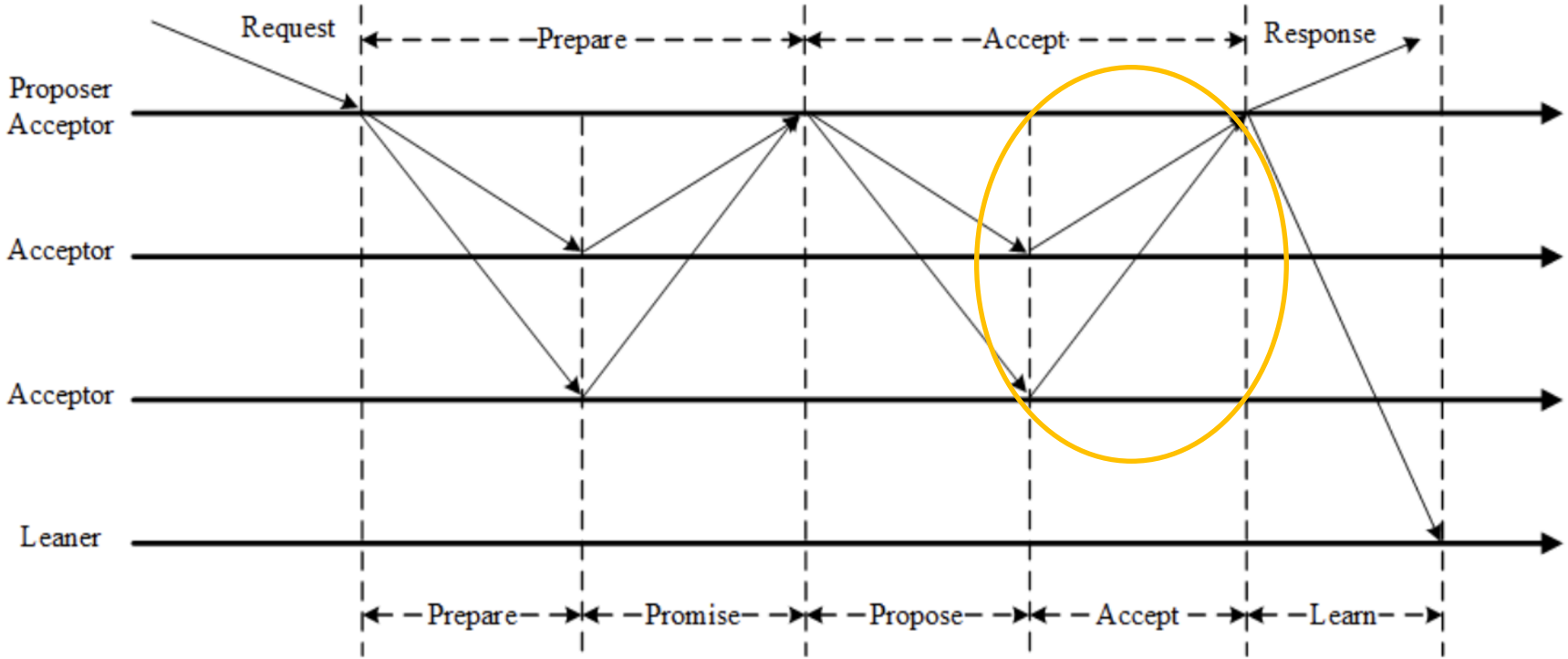
Classic Paxos



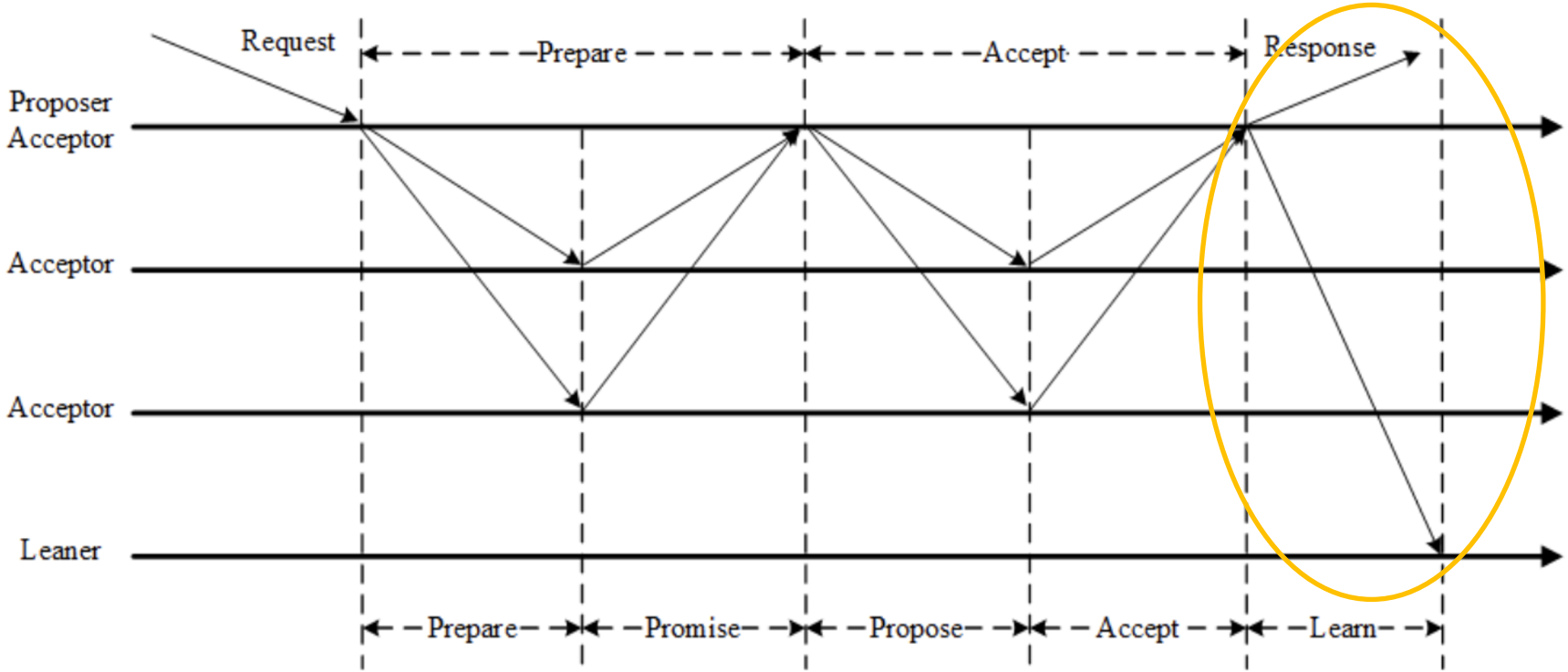
Classic Paxos



Classic Paxos



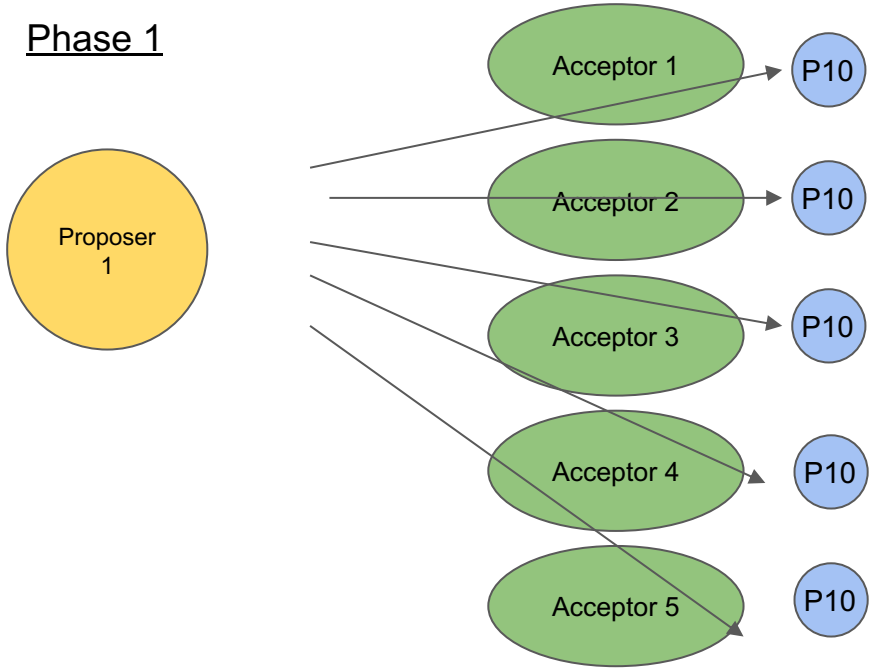
Classic Paxos



Multi-Paxos

Multi-Paxos

- Reserve Leadership
  - One Phase 1 of Classic Paxos
  - Write quickly with only Phase 2



No need in the future !

MDCC - Transactions Support

Transaction Support

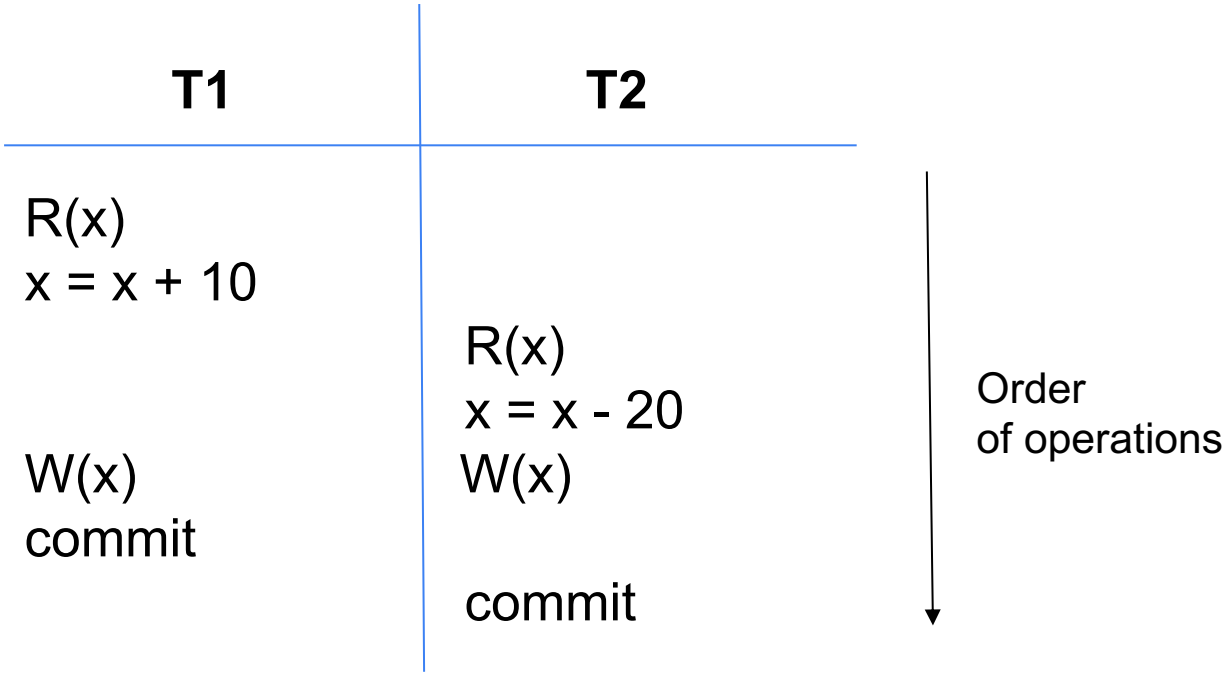
- Extension of Multi-Paxos supports multi-record transactions
  - Ensure **atomic durability**
  - Detect **write-write conflict**

- To guarantee consistency
  - Accept an ***option***
    - Not writing value directly
    - After committing transaction, notify storage nodes to execute options

Write – Write Conflict

Write-write conflict

Database x: 100



Write-write conflict

Database x: 100

T1		T2
R(x)	100	
x = x + 10		
		R(x)
		x = x - 20
W(x).		W(x)
commit		
		commit

Write-write conflict

Database x: 100

T1		T2
R(x)	100	
x = x + 10	110	
		R(x)
		x = x - 20
W(x)		W(x)
commit		
		commit

Write-write conflict

Database x: 100

T1		T2	
R(x)	100		
x = x + 10	110		
		R(x)	100
		x = x - 20	
W(x)		W(x)	
commit			
		commit	

Write-write conflict

Database x: 100

T1		T2	
R(x)	100		
x = x + 10	110		
		R(x)	100
		x = x - 20	80
W(x)		W(x)	
commit			
		commit	

Write-write conflict

Database x: 100

T1		T2	
R(x)	100		
x = x + 10	110		
		R(x)	100
		x = x - 20	80
W(x)	110	W(x)	80
commit			
		commit	

Write-write conflict

Database x: 110

T1		T2	
R(x)	100		
x = x + 10	110		
		R(x)	100
		x = x - 20	80
W(x)	110	W(x)	80
commit	110		
		commit	

Write-write conflict

Database x: 80

T1		T2	
R(x)	100		
x = x + 10	110		
		R(x)	100
		x = x - 20	80
W(x)	110	W(x)	80
commit	110		
		commit	80

Write-write conflict - Solution

Two Phase Lock

- Shared lock (Read lock)
  - read data items only
- Exclusive lock (Write lock)
  - read and write data items
  - owned by only one transaction at a time

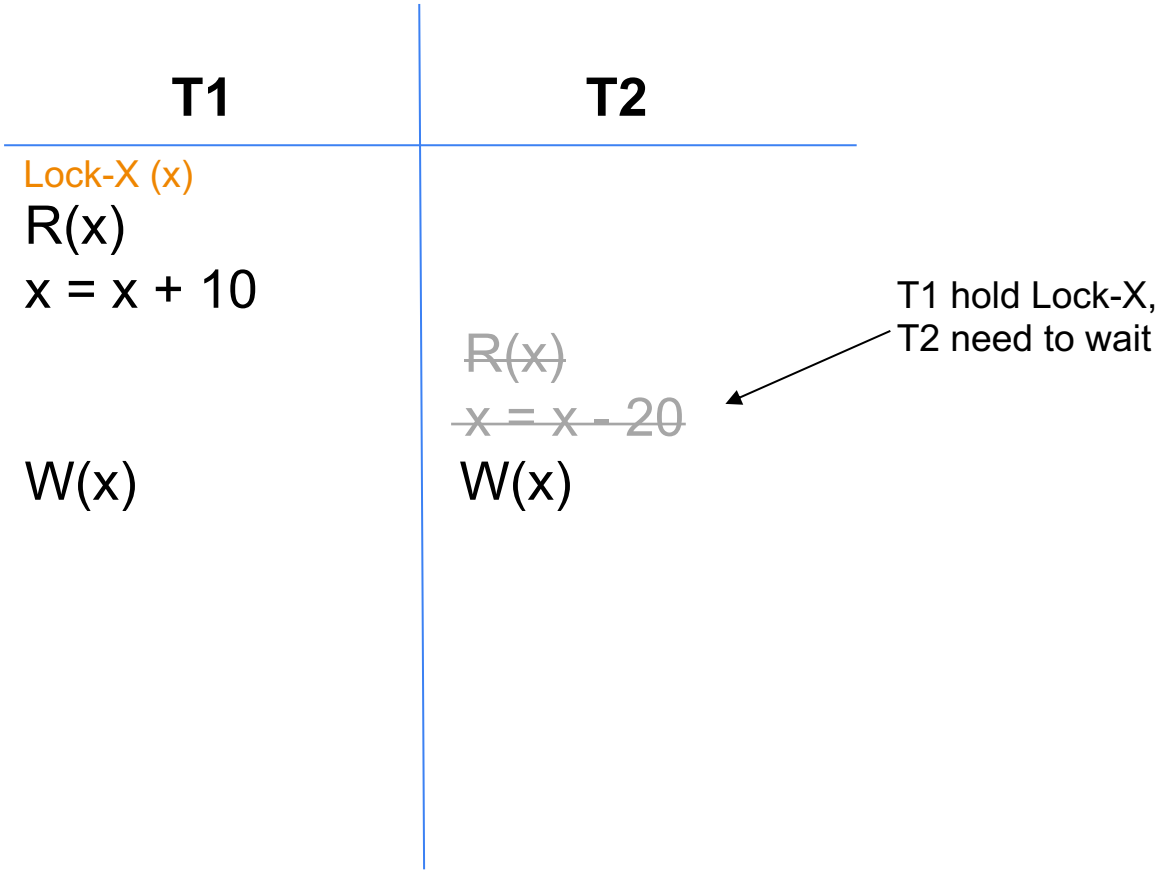
Write-write conflict - Solution

Database x: 100
T1: x = x + 10
T2: x = x - 20

T1	T2
Lock-X (x)	
R(x)	
x = x + 10	
	R(x)
	x = x - 20
W(x)	
	W(x)

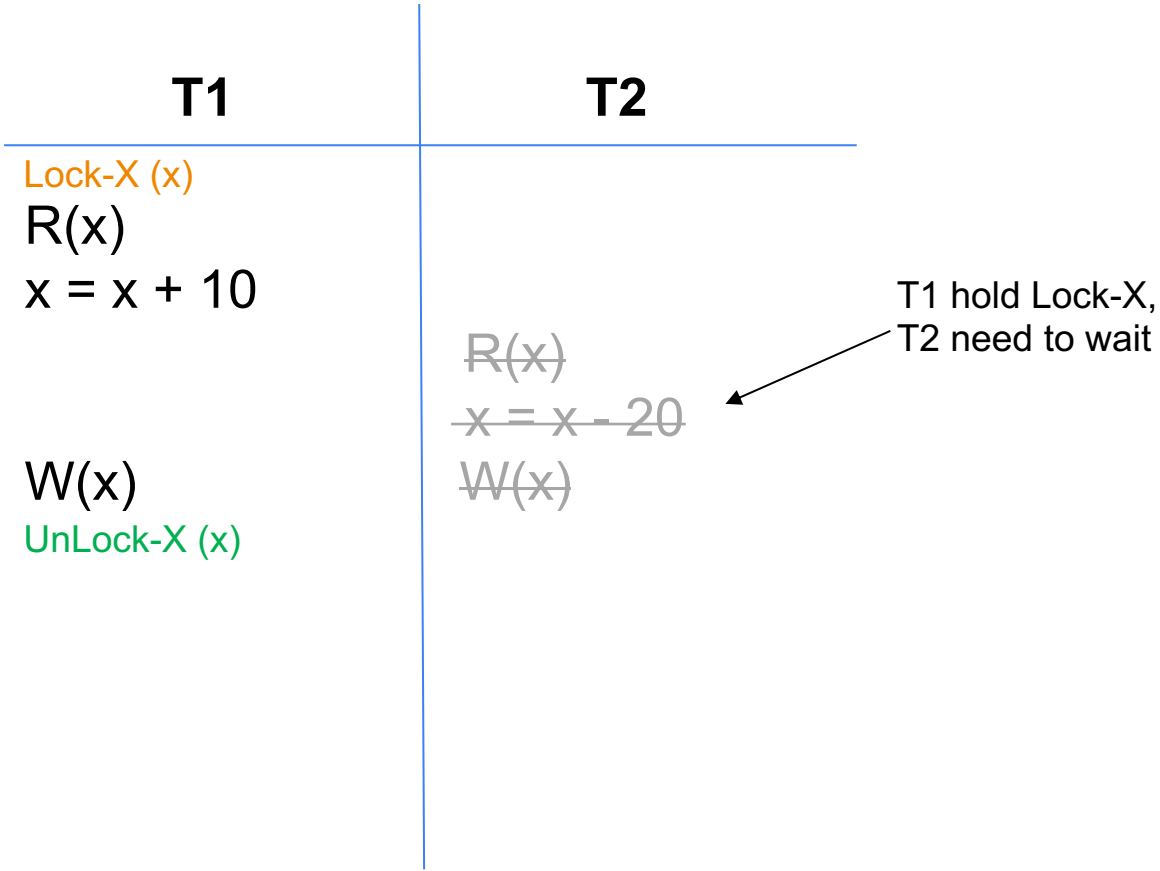
Write-write conflict

Database x: 100
T1: x = x + 10
T2: x = x - 20



Write-write conflict

Database x: **110**
T1: x = x + 10
T2: x = x - 20



Write-write conflict

Database x: ~~110~~ **90**
T1: x = x + 10
T2: x = x - 20

T1	T2
Lock-X (x)	
R(x)	
x = x + 10	
	~~R(x)~~
	~~x = x - 20~~
W(x)	~~W(x)~~
UnLock-X (x)	
	Lock-X (x)
	R(x)
	x = x - 20
	W(x)
	UnLock-X (x)

T1 hold Lock-X,
T2 need to wait

Deadlocks

Deadlocks

- Two Phase Lock might cause deadlocks

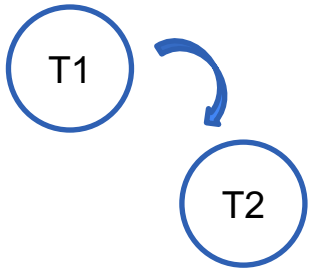
Deadlocks

T1	T2	T3
Lock-S(A)		
	Lock-X(B)	
		Lock-S(C)

Deadlocks

T1	T2	T3
Lock-S(A)		
	Lock-X(B)	
		Lock-S(C)
Lock-S(B)		

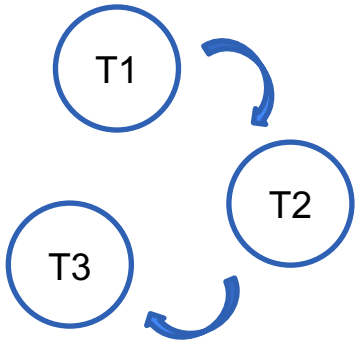
Wait-for-graph



Deadlocks

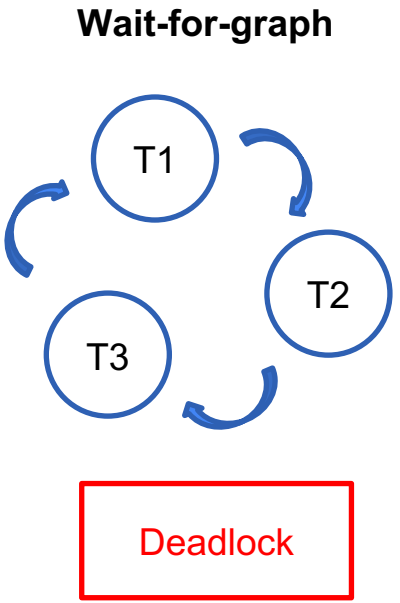
T1	T2	T3
Lock-S(A)		
	Lock-X(B)	
		Lock-S(C)
Lock-S(B)		
	Lock-X(C)	

Wait-for-graph



Deadlocks

T1	T2	T3
Lock-S(A)		
	Lock-X(B)	
		Lock-S(C)
Lock-S(B)		
	Lock-X(C)	
		Lock-X(A)



Deadlocks

- Two Phase Lock might cause deadlocks
- Deadlocks detection
  - Use Wait-for-graph
  - Abort one transaction to break cycle

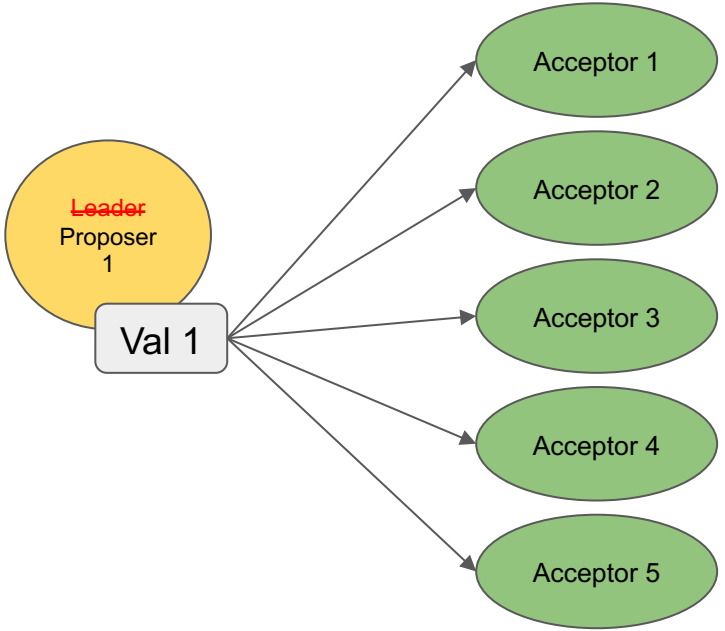
Protocol - Failure Scenarios

- Storage nodes failure
  - Solution: Be masked by the use of quorums
- Master failure
  - Solution
    - Be recovered by selecting new master
    - Trigger Phase 1 and Phase 2
- App-server failure
  - **Dangling transaction**
  - Solution
    - **Unique transaction-id**, **primary keys of write-set**, **log of learned options**
    - Reconstruct the node

Fast Paxos

Fast Paxos

- Does not require leadership
  - Any client / proposer can write values to the acceptors without becoming the leader
- Need larger quorum size
  - Classic : [N/2] responses
  - Fast : [2N/3] + 1 responses
  - For a value to be safely written
- Need more time to resolve conflicts



Fast Paxos – Resolve Conflicts

- Concurrent updates might cause collision
- Leader must step in to resolve conflicts
- 2 additional message rounds to resolve conflicts
- Need 3 message rounds

MDCC - Transactions Bypassing the Master

Protocol - MDCC use Fast Ballots Approach

1. All versions starts with a Fast Ballot number
2. This ballot number informs acceptors to accept next options from any Proposers
3. Clients can propose option **directly** to the acceptor.
4. Storage node accept the **first** proposed option

MDCC – Fast Policy

Fast Policy – Trade-offs

- Classic instances
  - Require 2 message rounds

- Fast instances
  - Require only 1 message round
  - Collision : will require another 2 message rounds

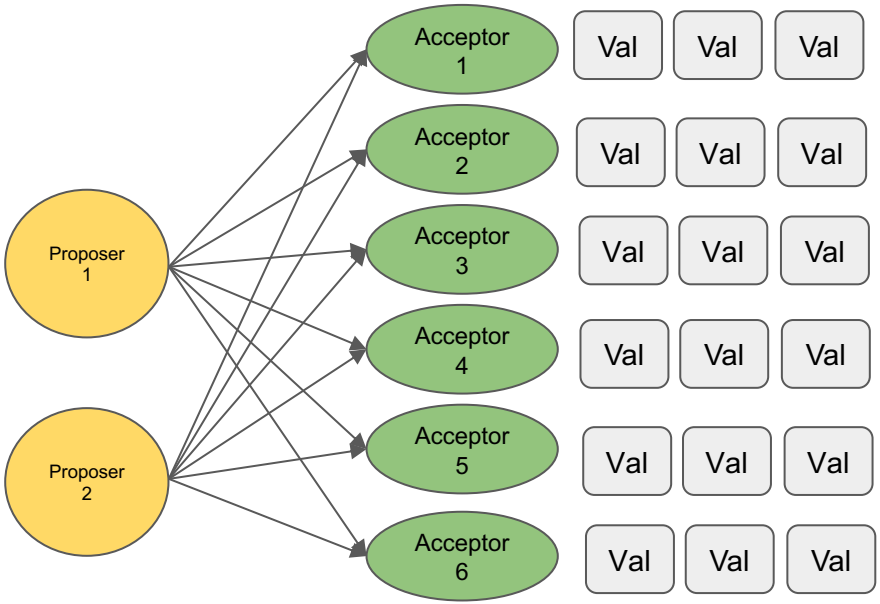
Fast Policy - Strategy

- Pre-set all instances to fast
- Detect a collision and set the next instances to classic.
- After resolving conflicts, go back to fast instance.

Generalized Paxos

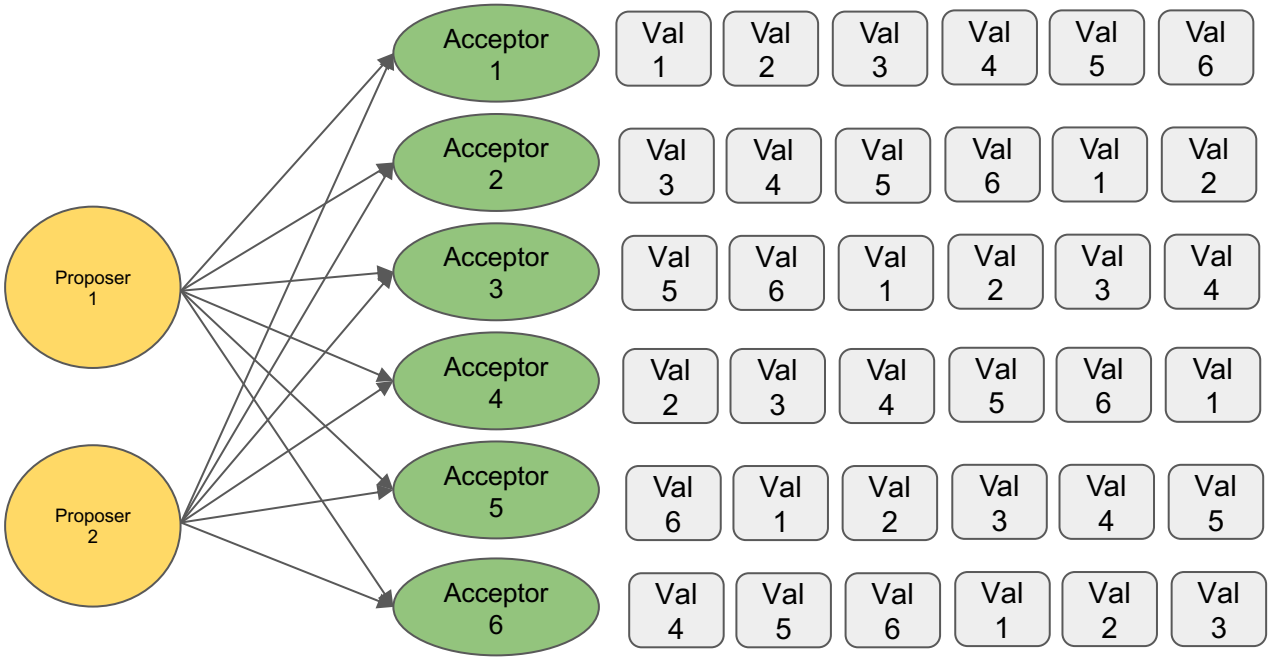
Generalized Paxos

- Combines Classic Paxos and Fast Paxos.
- Each round accepts a sequence of values.



Generalized Paxos

- **Compatibility of sequences** -> To detect conflicts.



Generalized Paxos - Example

- Proposed operations need to be commutative operations.

Commutativity Table

	Read(A)	Write(A)	Read(B)	Write(B)
Read(A)		✗		
Write(A)	✗	✗		
Read(B)				✗
Write(B)			✗	✗

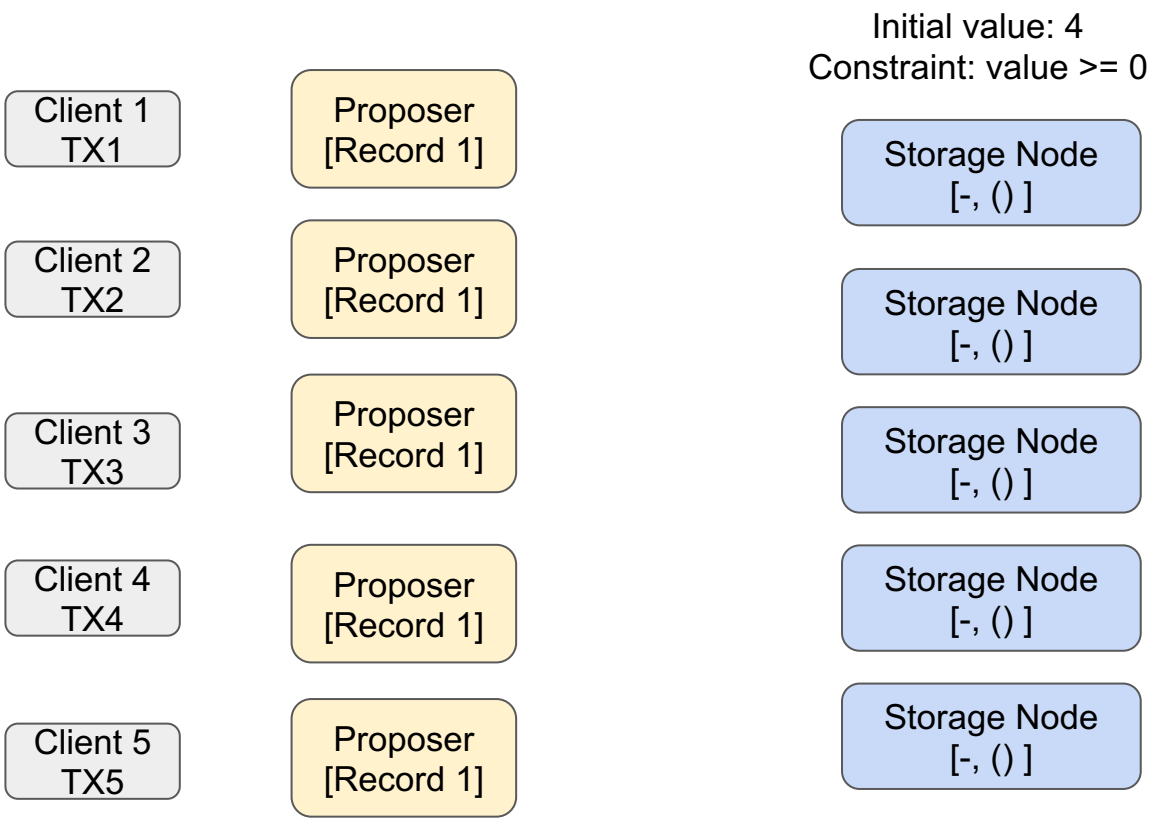
MDCC - Commutative Updates

MDCC - Commutative Updates

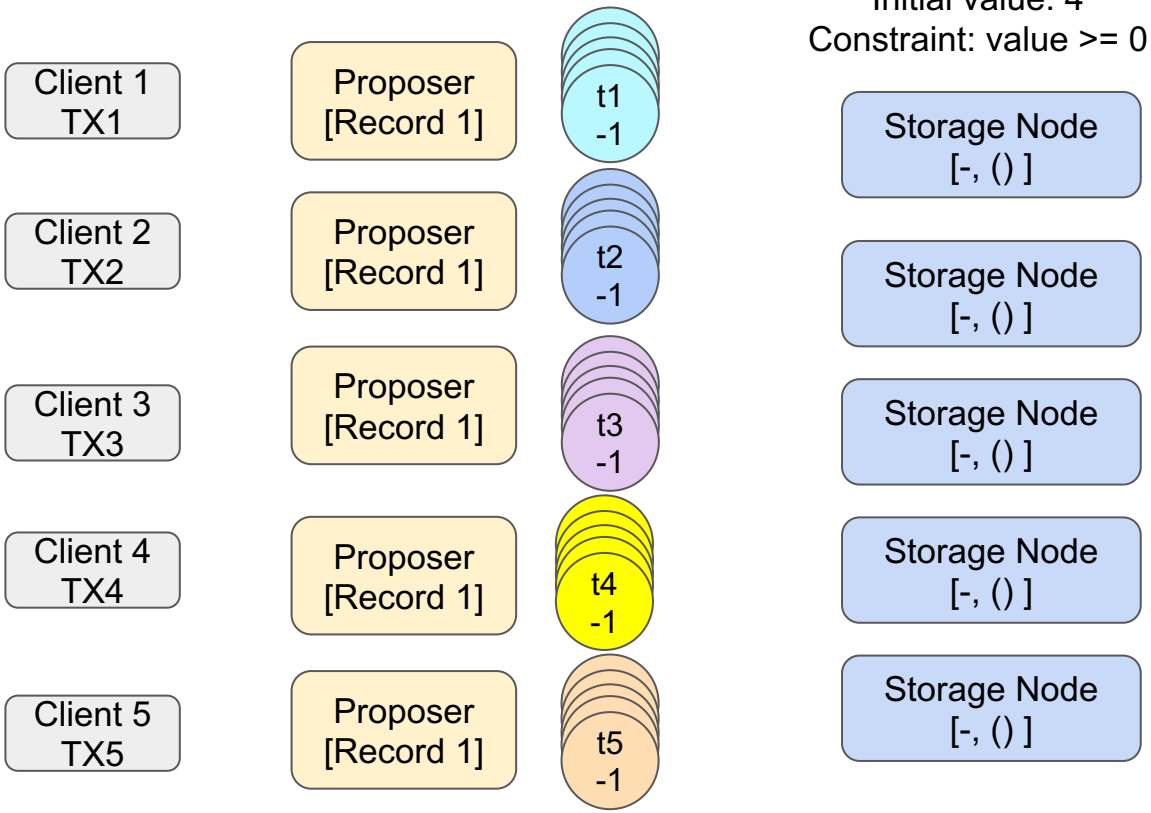
- MDCC usage of Generalized Paxos
  - Avoid conflicts for concurrent updates.
  - Sequence is only available for commutative operations.

- MDCC - New Demarcation Protocol
  - Use for commutative updates in MDCC
  - Deal with domain integrity constraints
    - EX: Constraint: value >= 0

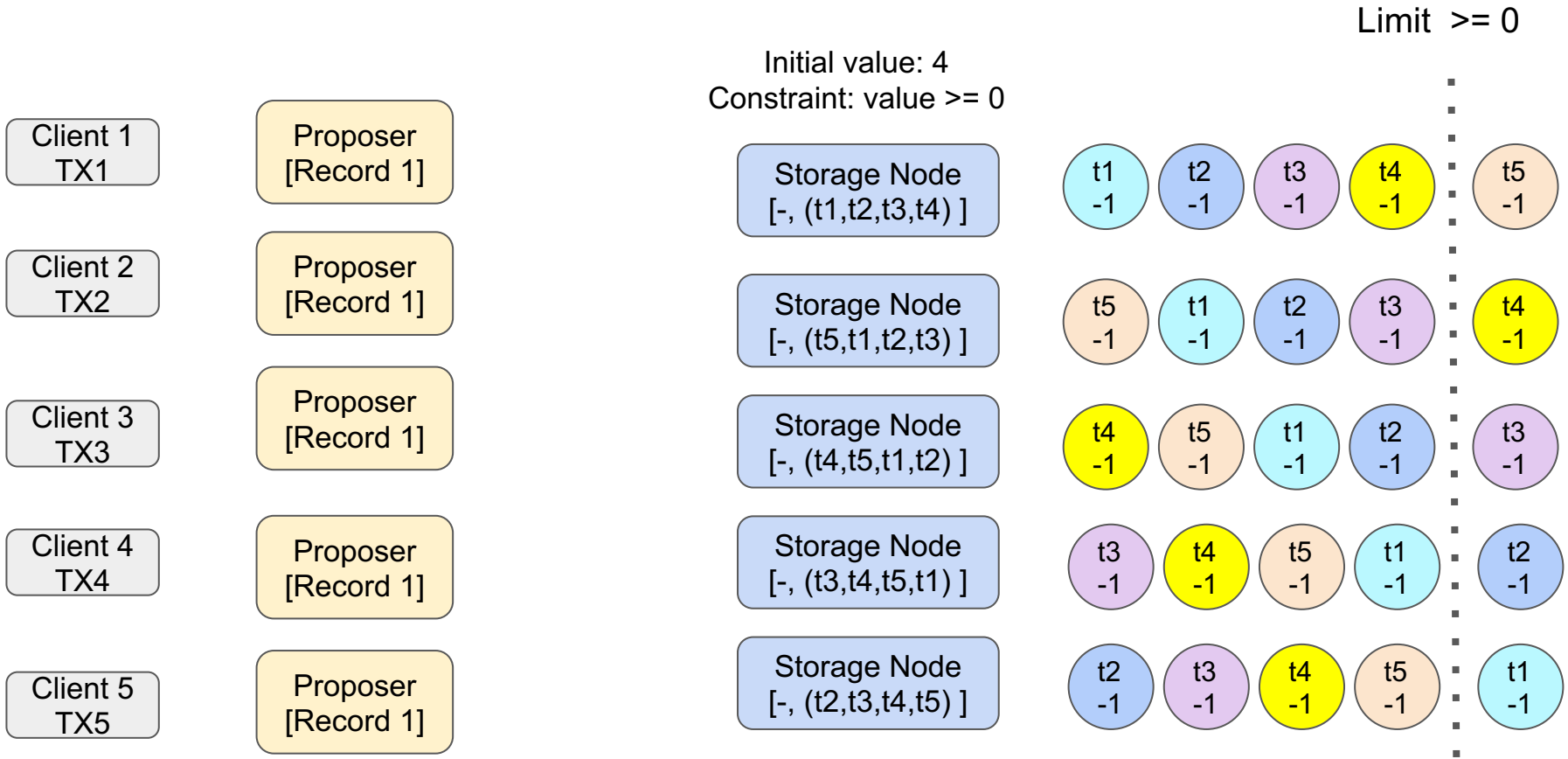
General Demarcation Protocol



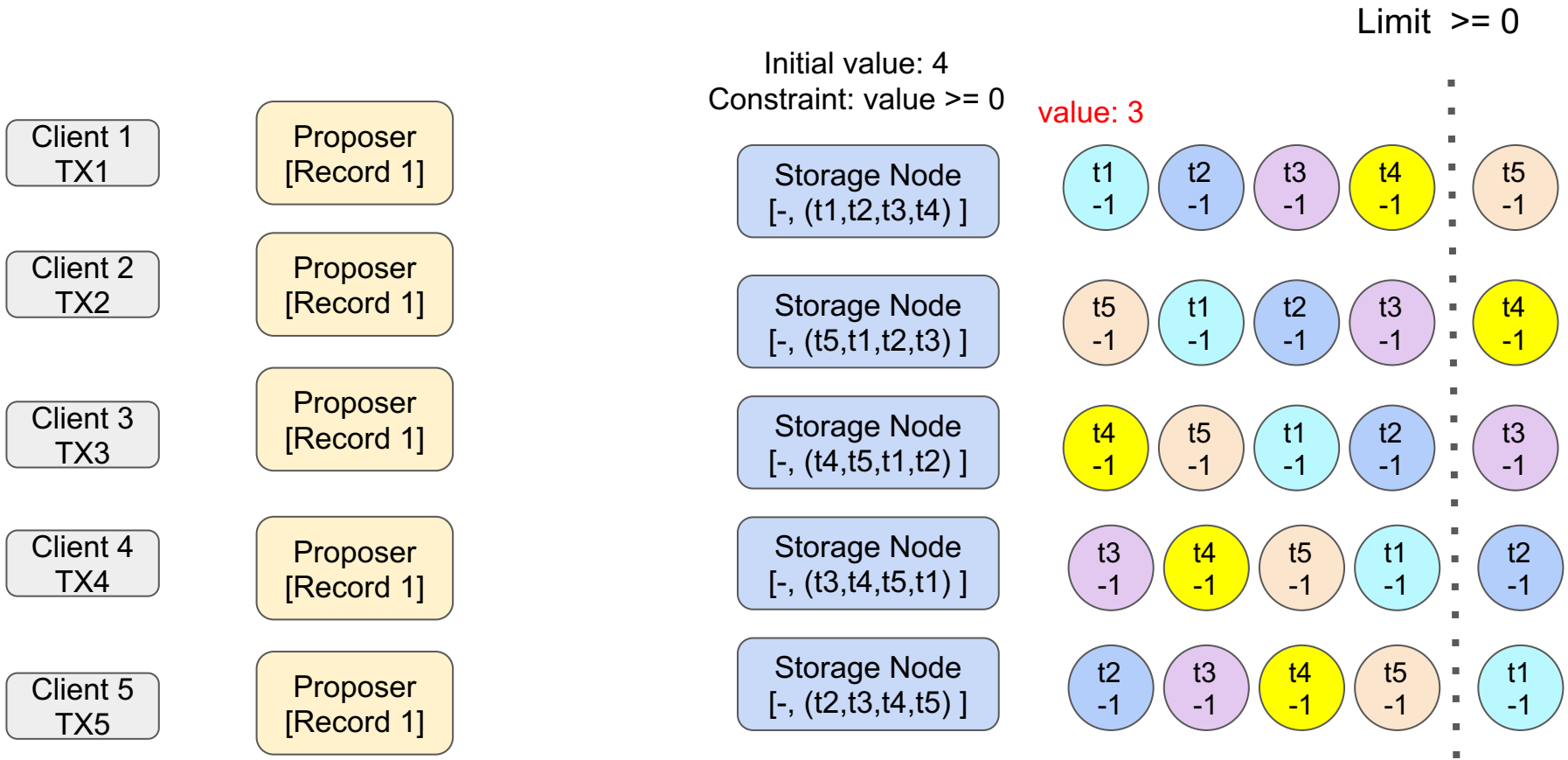
General Demarcation Protocol



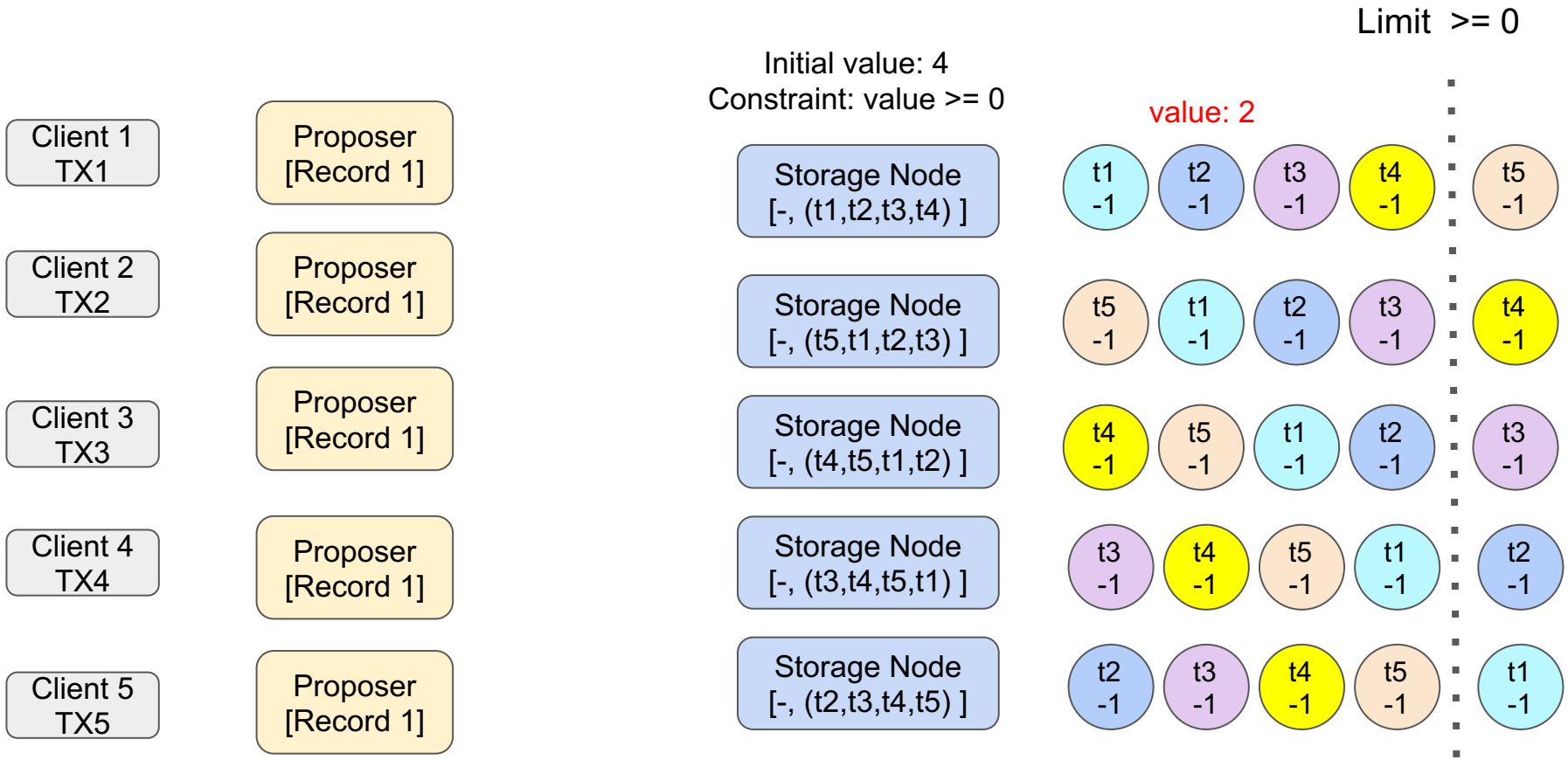
General Demarcation Protocol



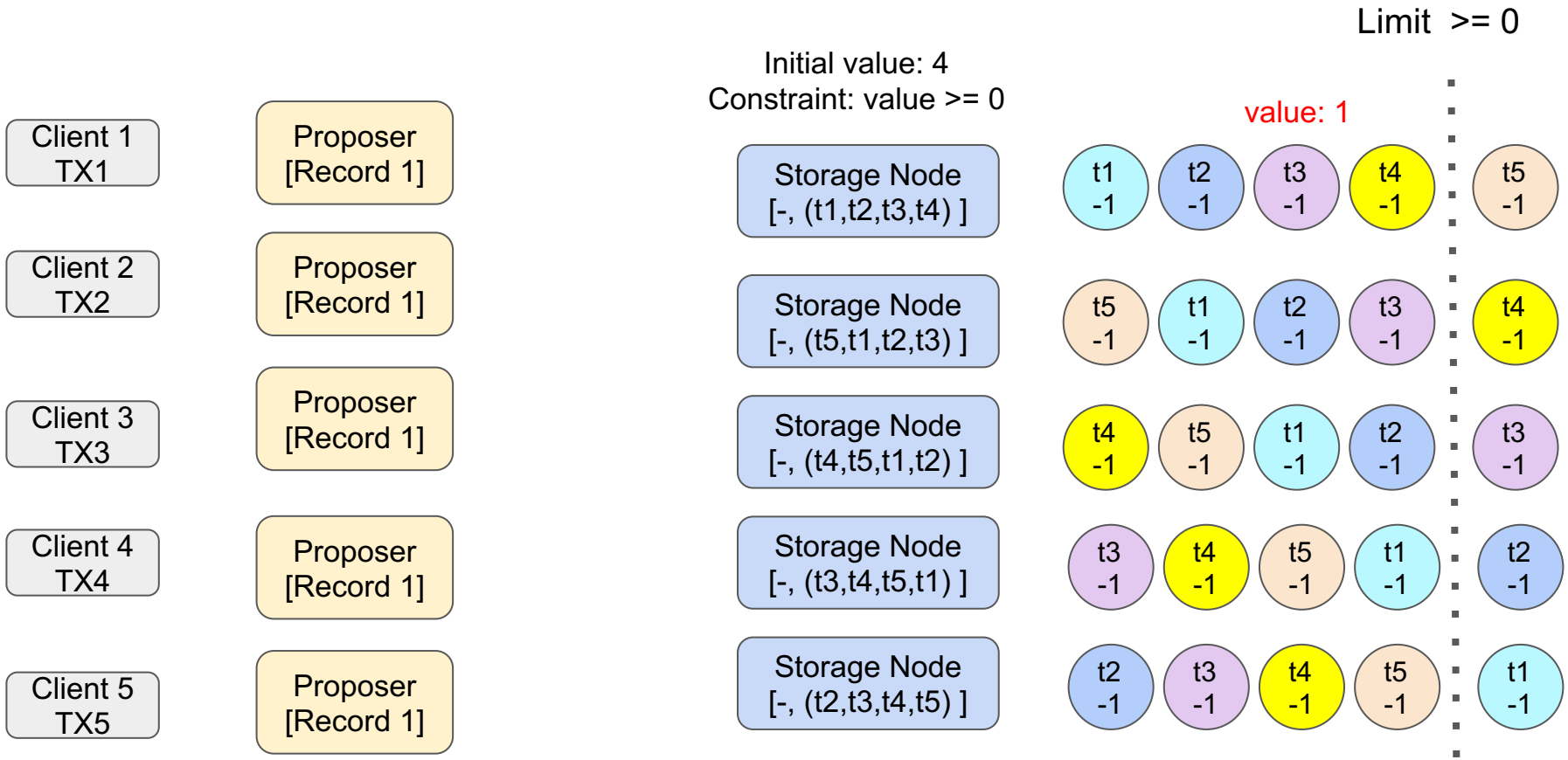
General Demarcation Protocol



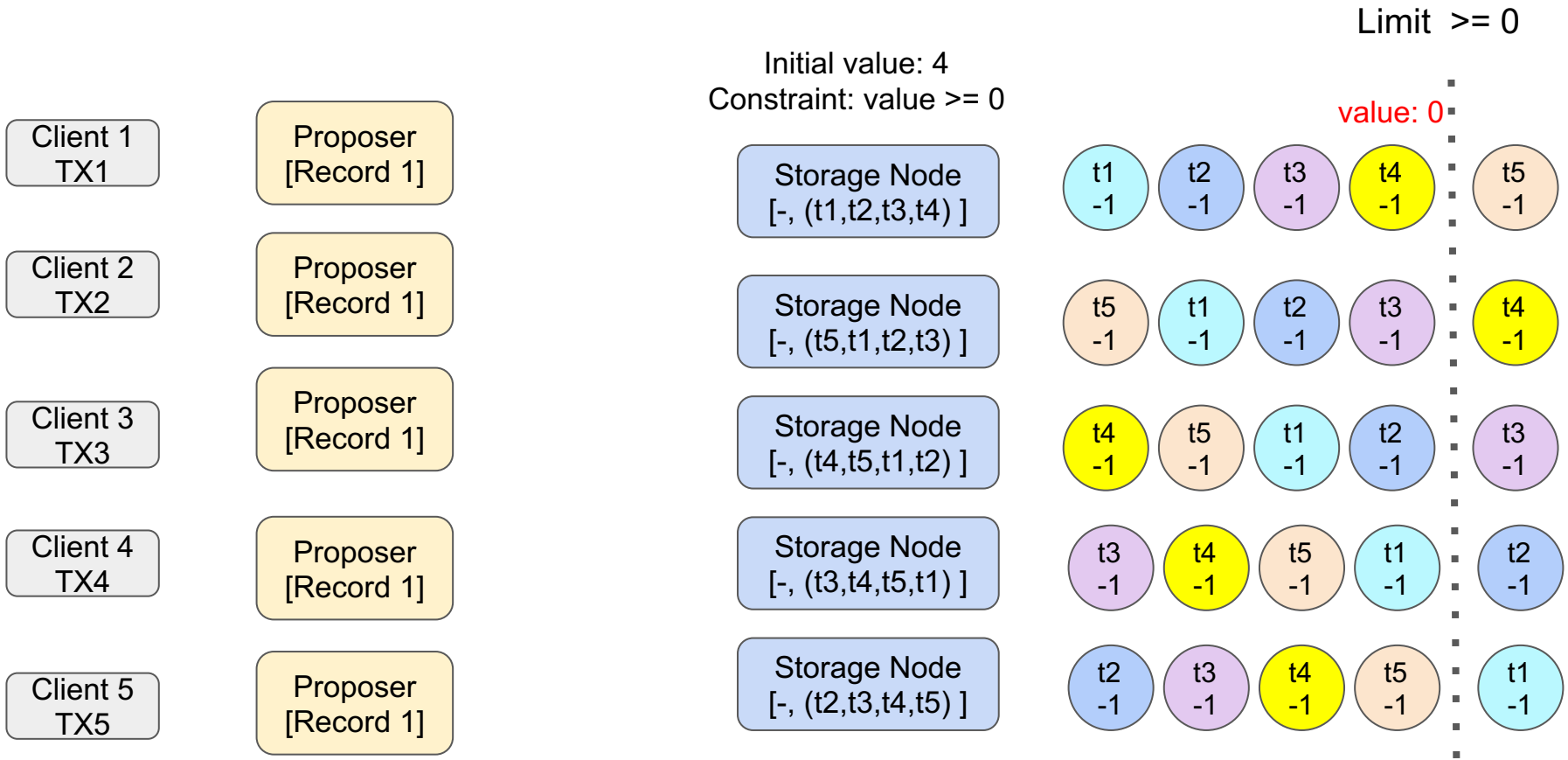
General Demarcation Protocol



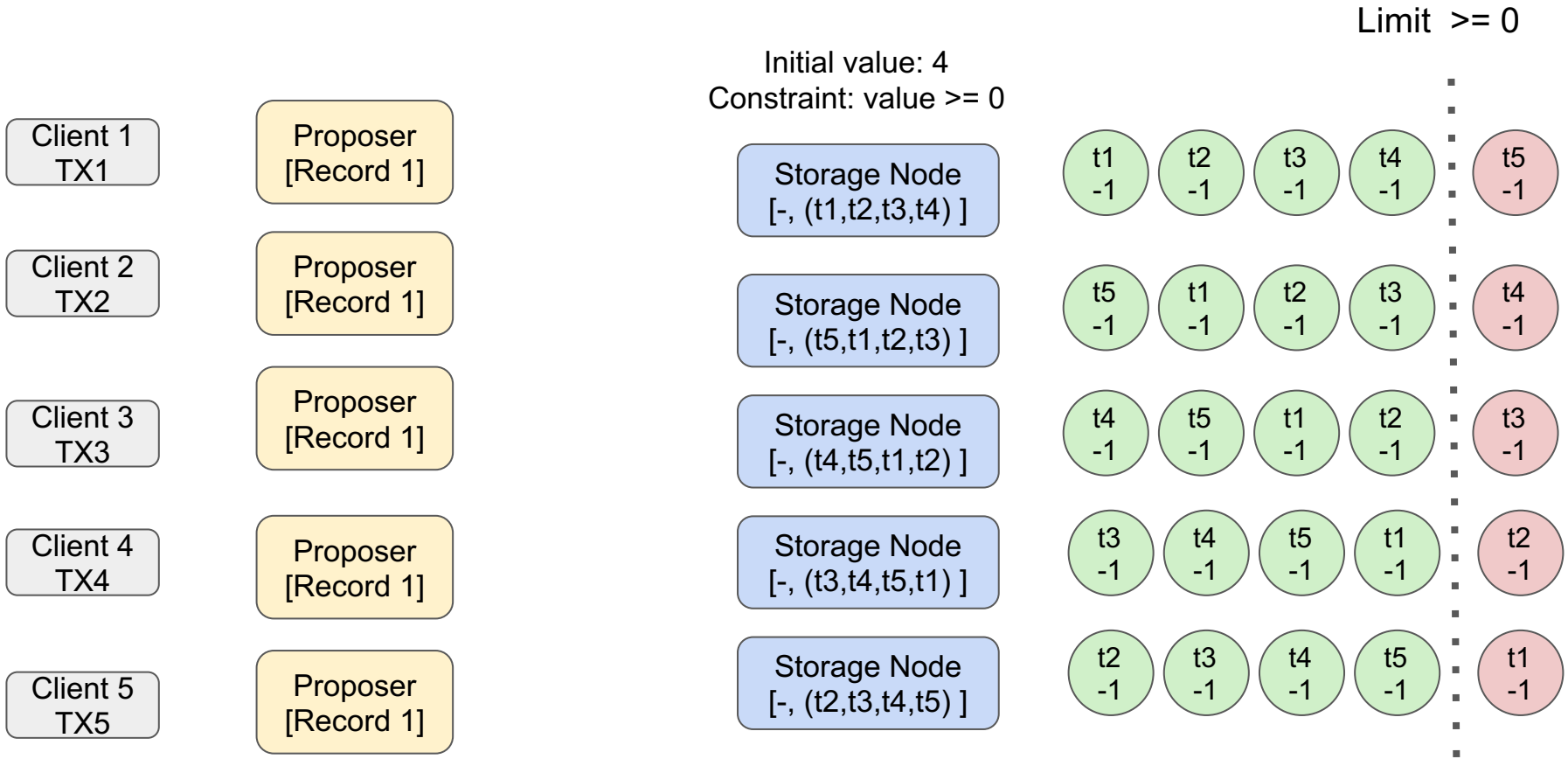
General Demarcation Protocol



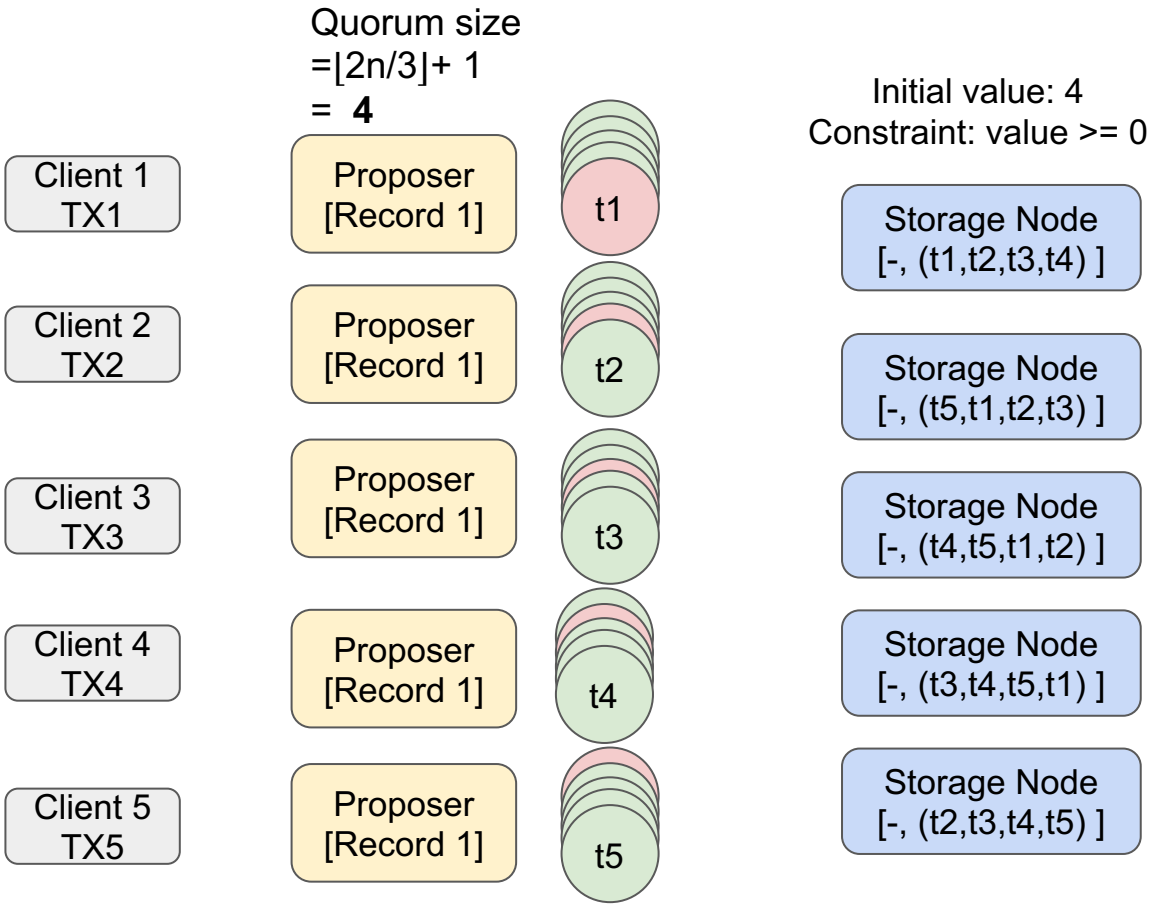
General Demarcation Protocol



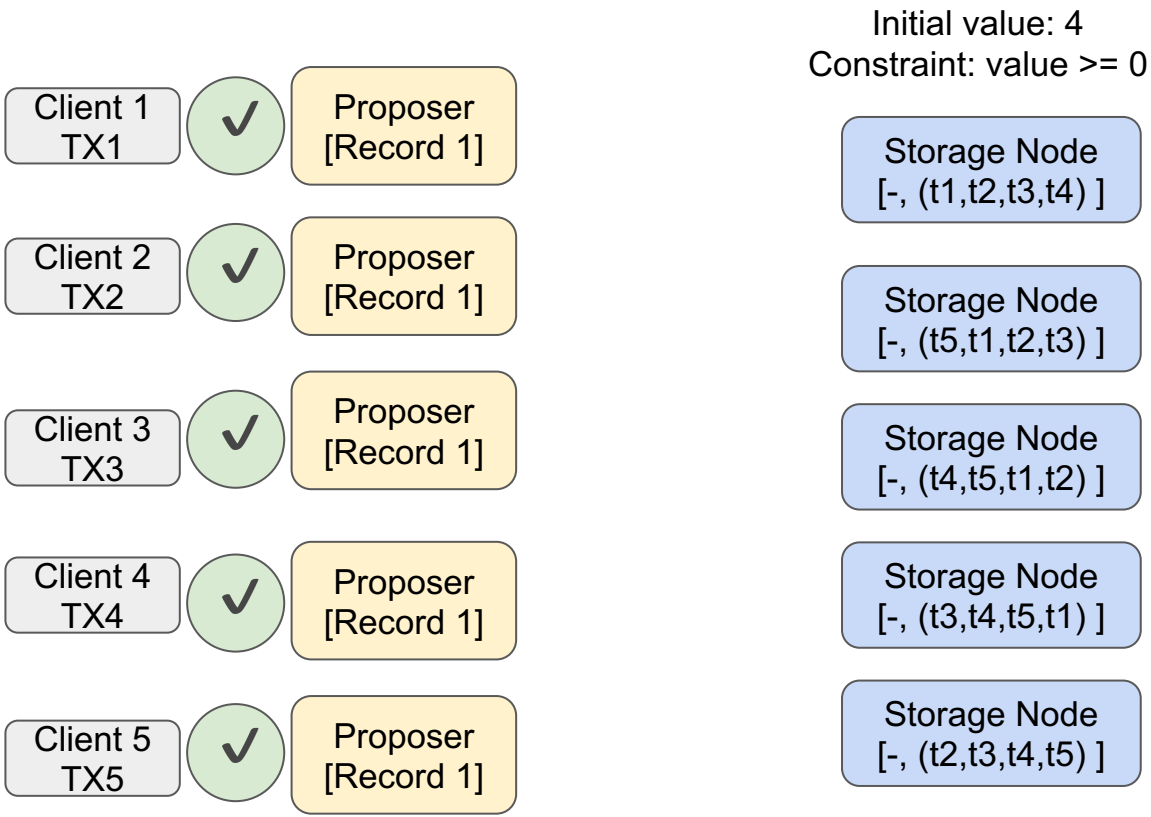
General Demarcation Protocol



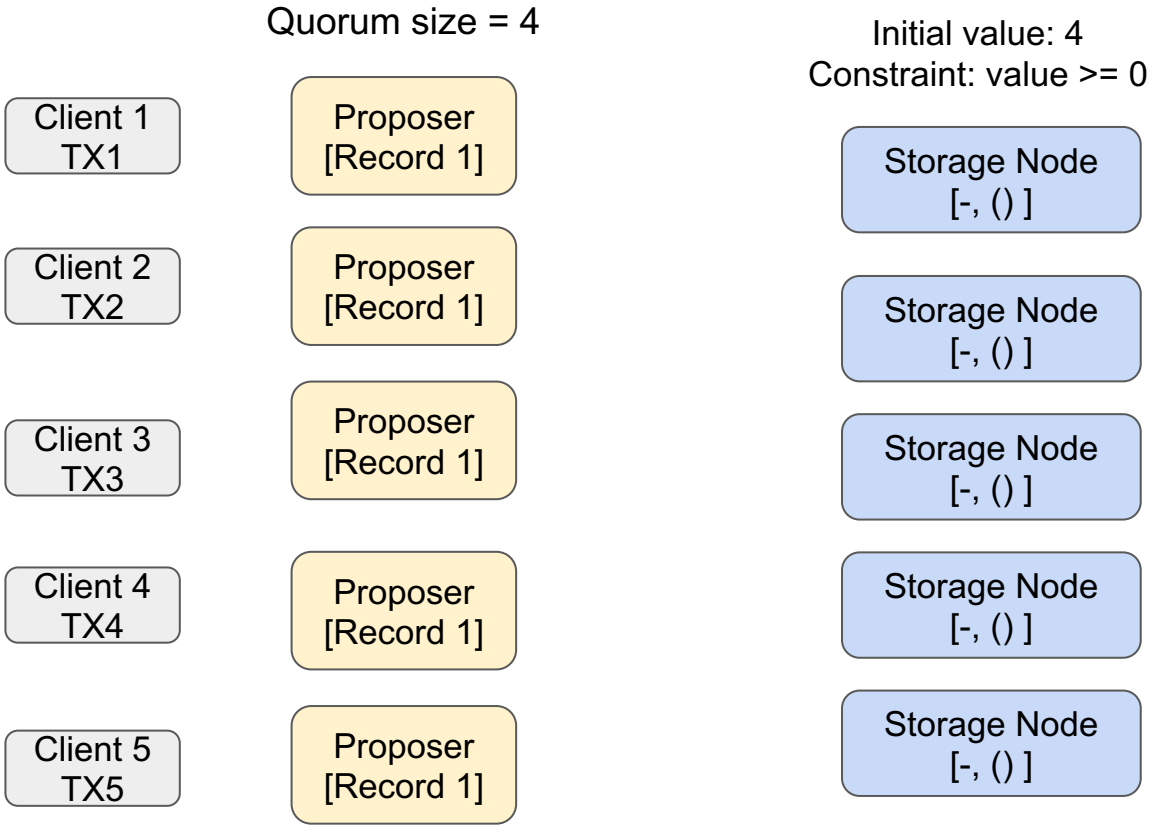
General Demarcation Protocol



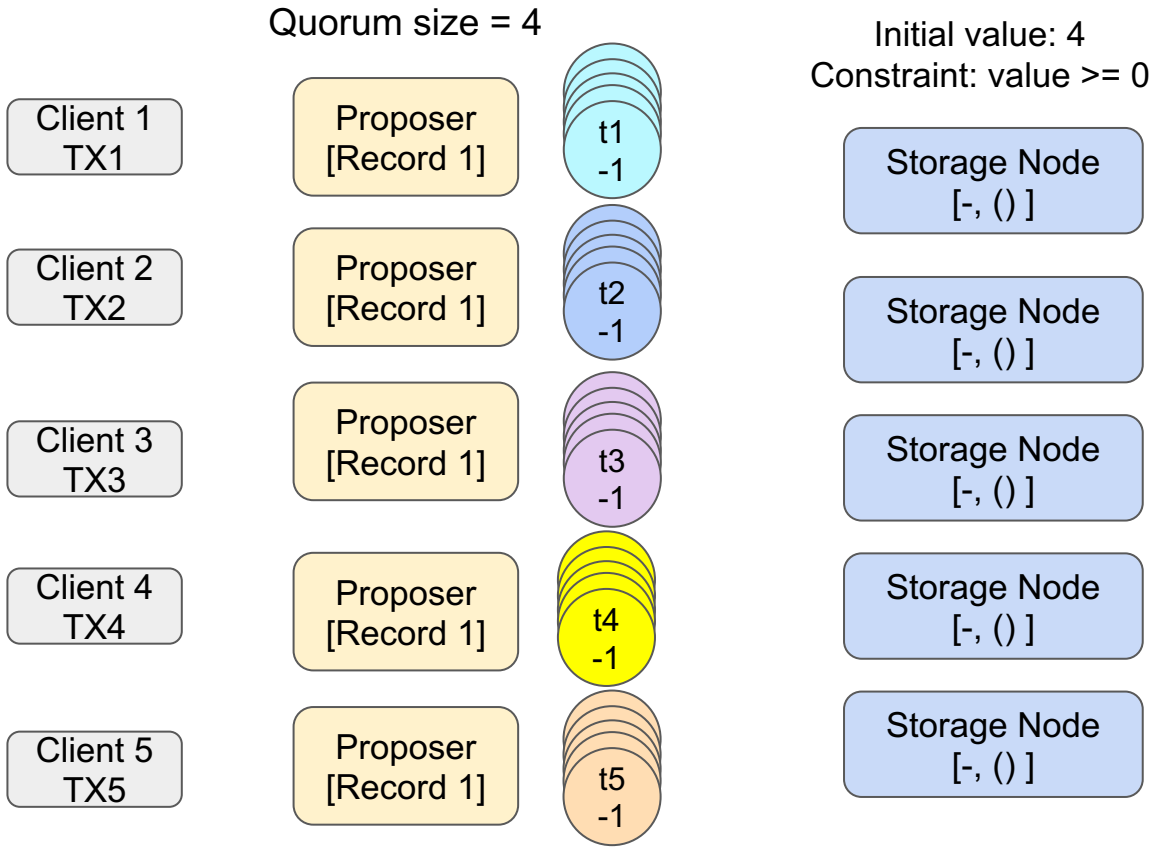
General Demarcation Protocol



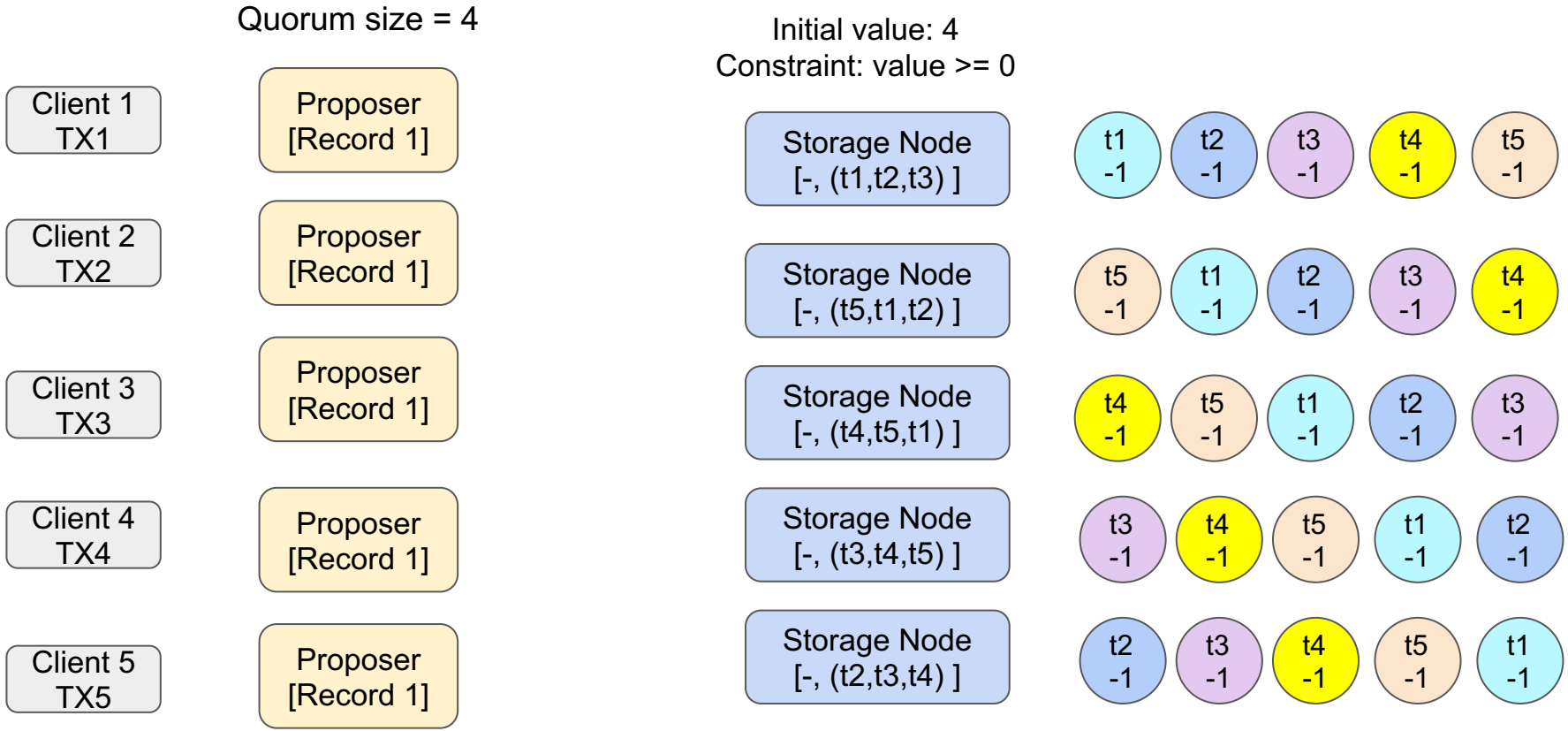
MDCC – New Demarcation Protocol



MDCC – New Demarcation Protocol



MDCC – New Demarcation Protocol

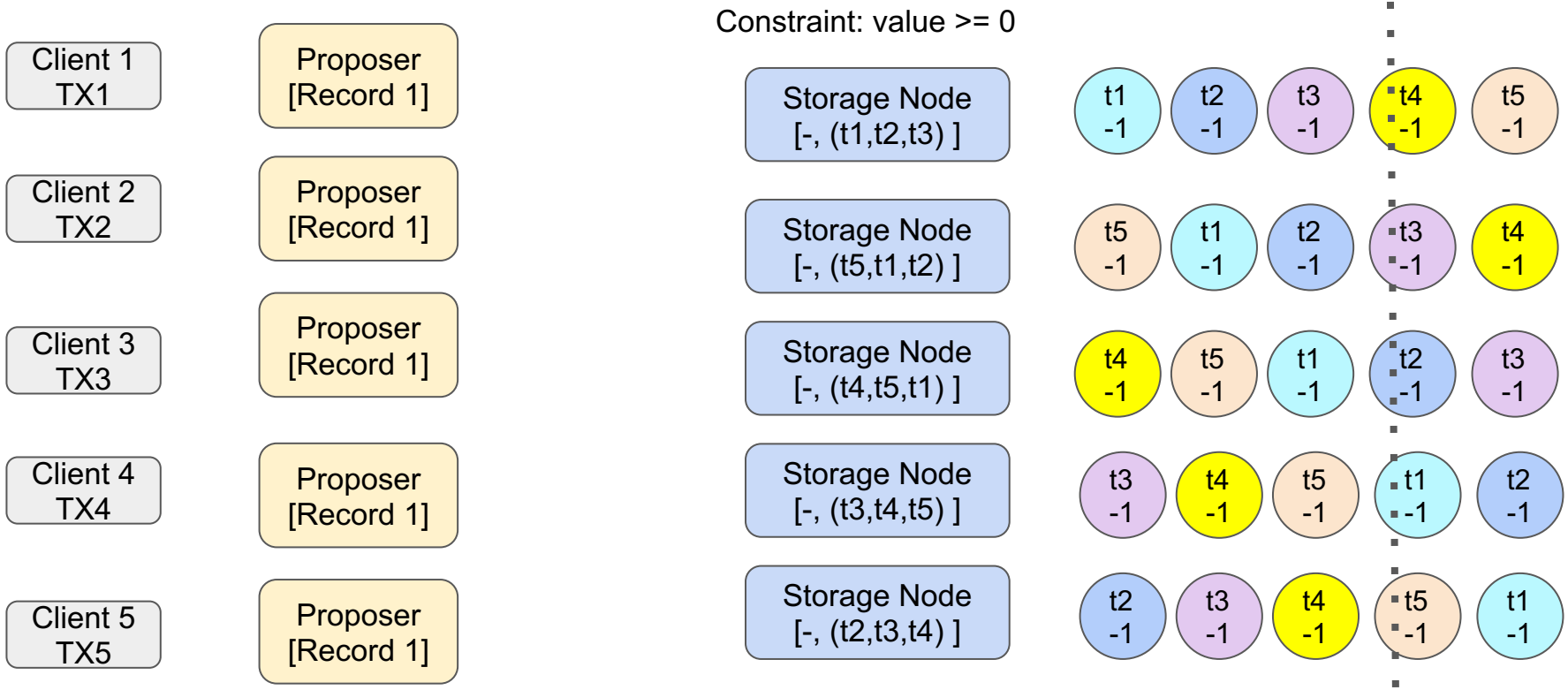


MDCC – New Demarcation Protocol

Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

Initial value: 4
Constraint: value >= 0



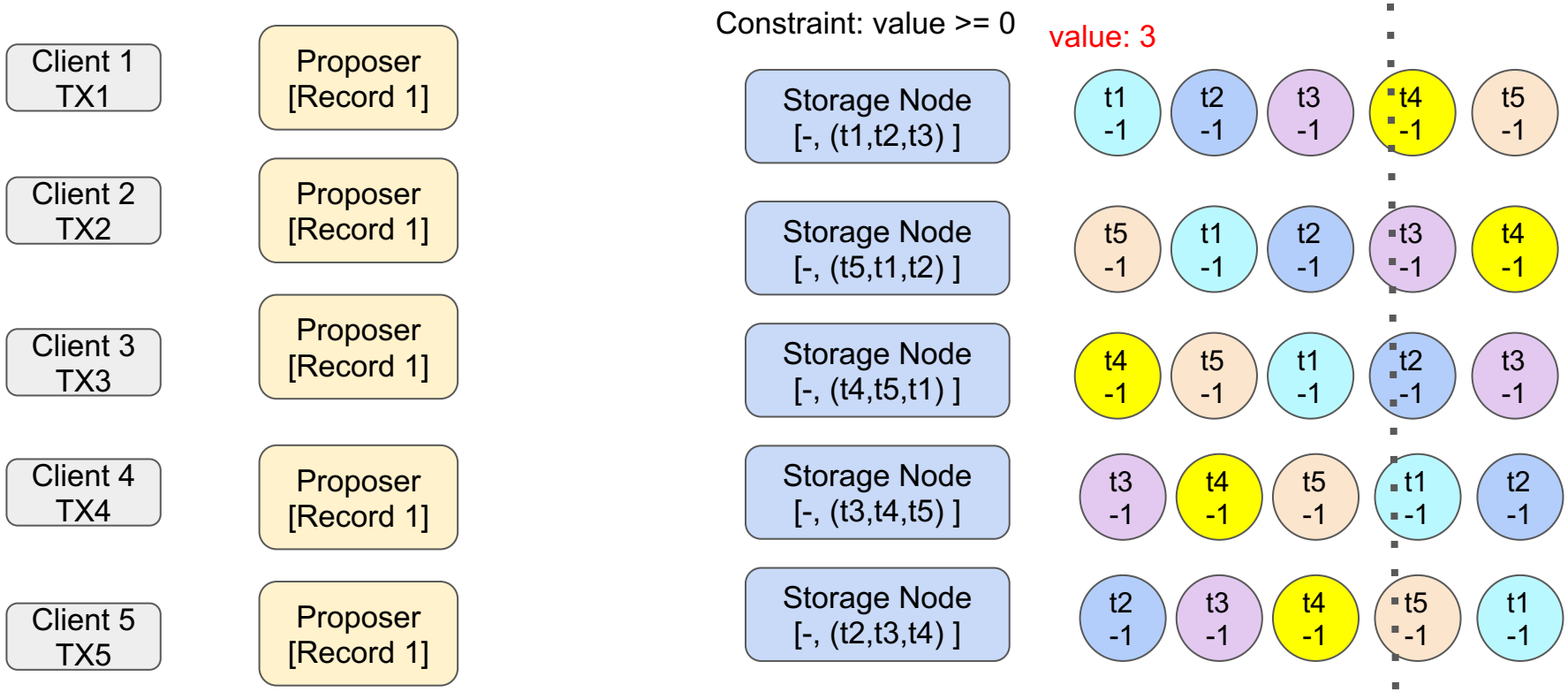
MDCC – New Demarcation Protocol

Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

Initial value: 4
Constraint: value >= 0

value: 3



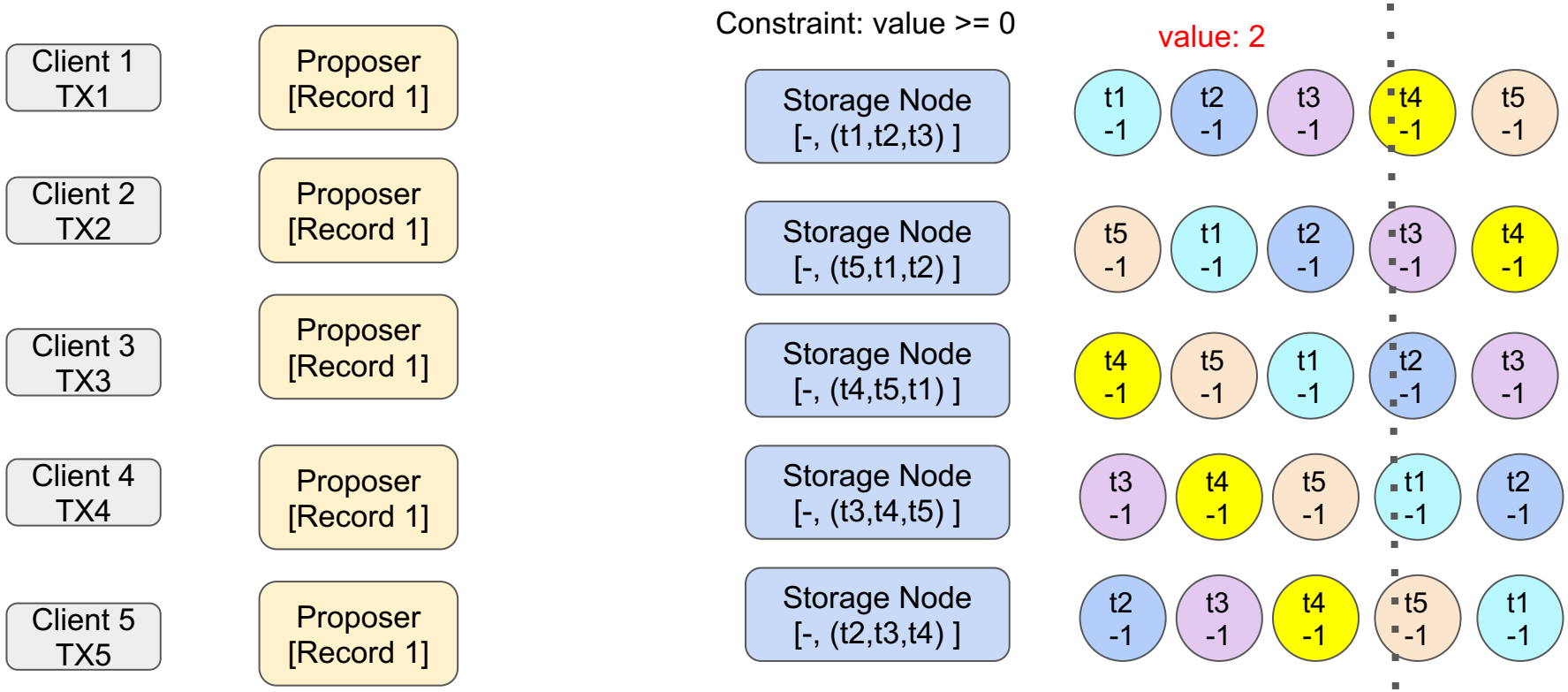
MDCC – New Demarcation Protocol

Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

Initial value: 4
Constraint: value >= 0

value: 2

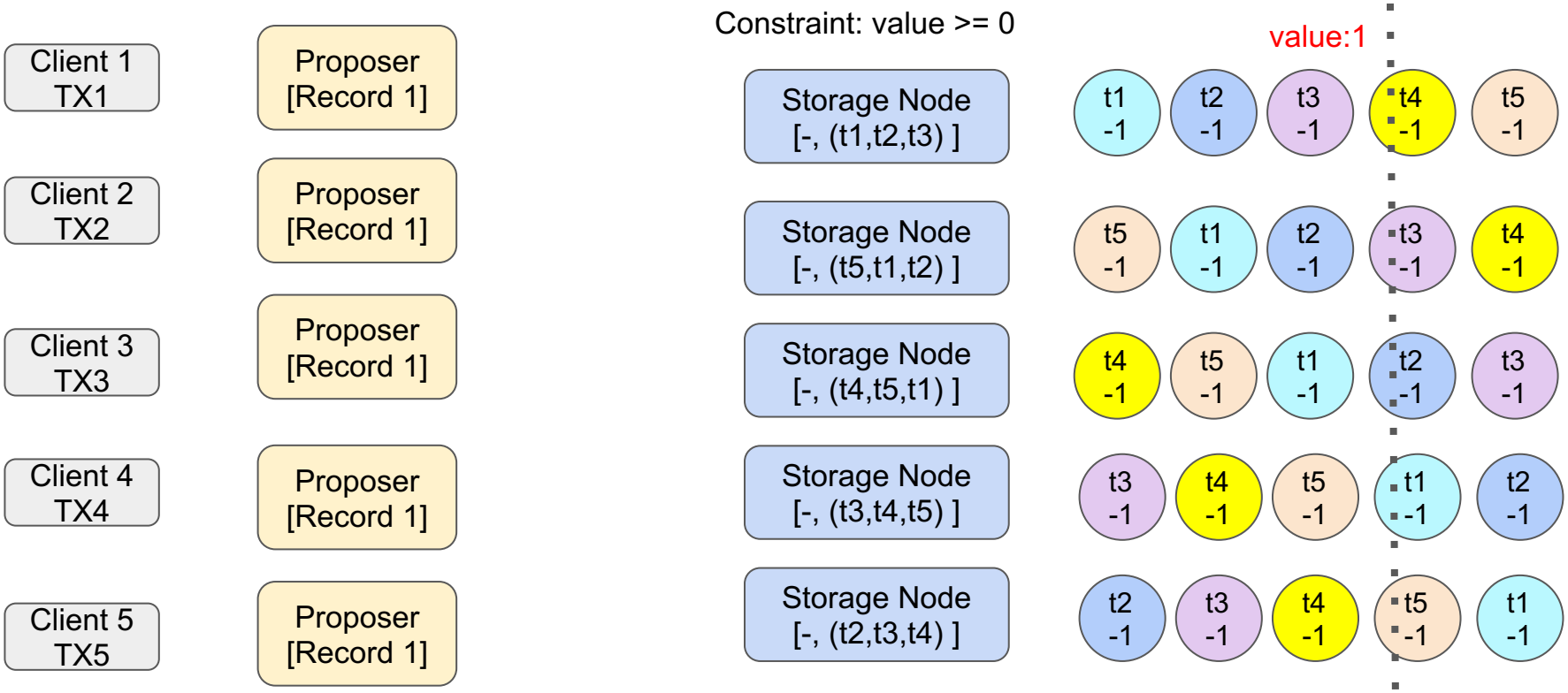


MDCC – New Demarcation Protocol

Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

Initial value: 4
Constraint: value >= 0

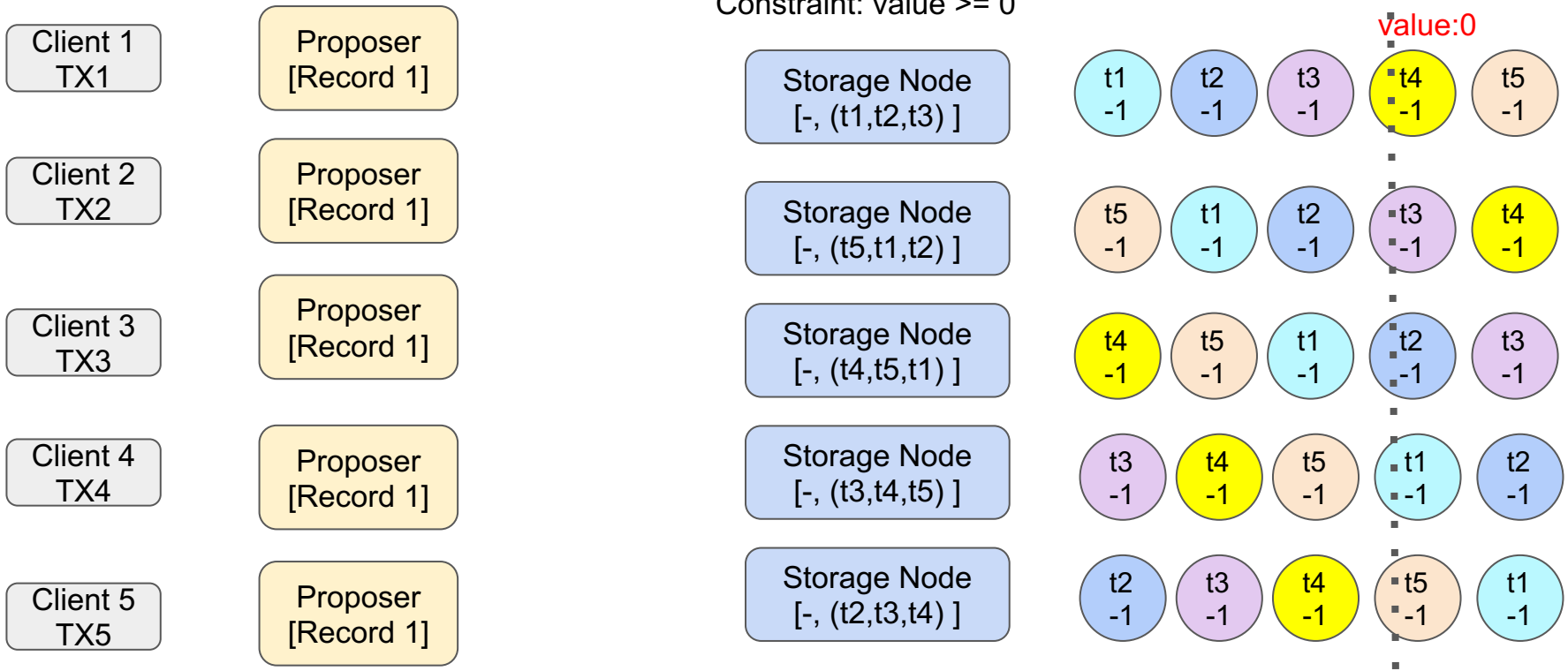


MDCC – New Demarcation Protocol

Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

Initial value: 4
Constraint: value >= 0

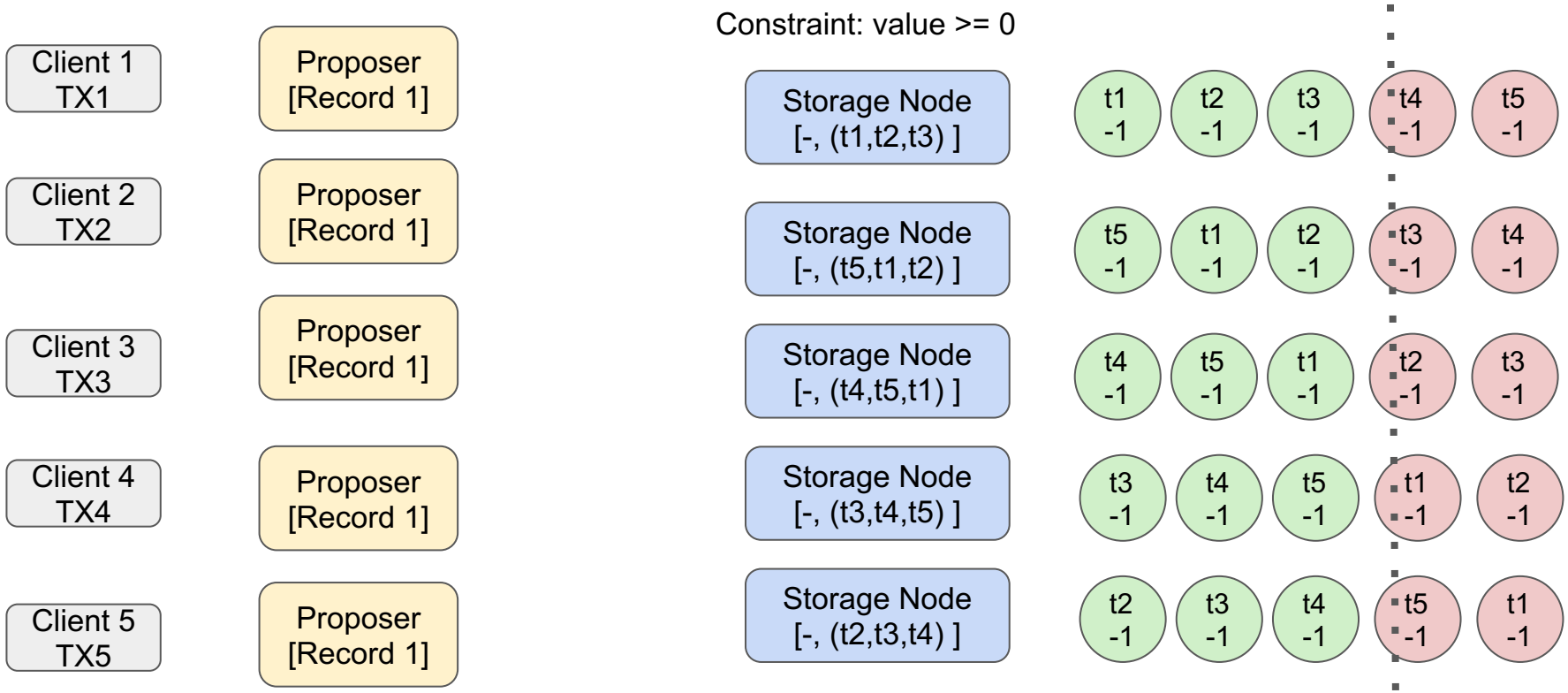


MDCC – New Demarcation Protocol

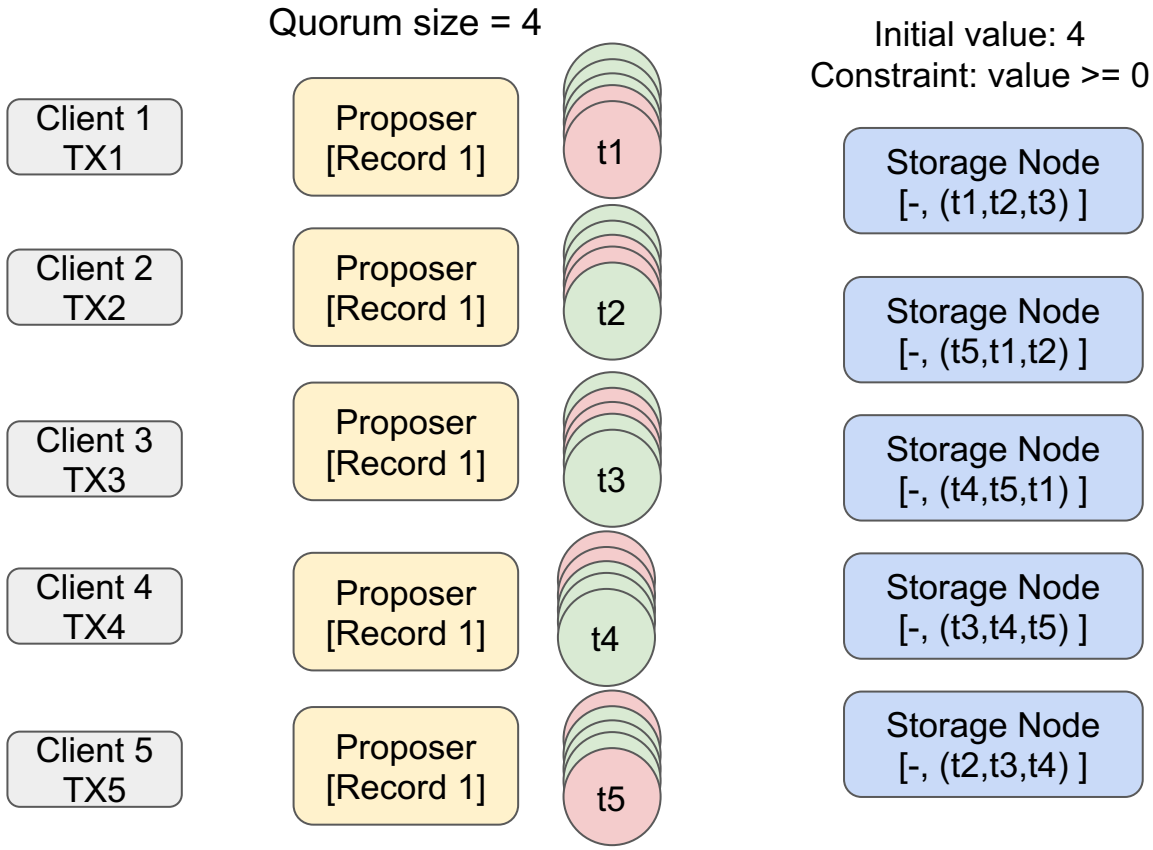
Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

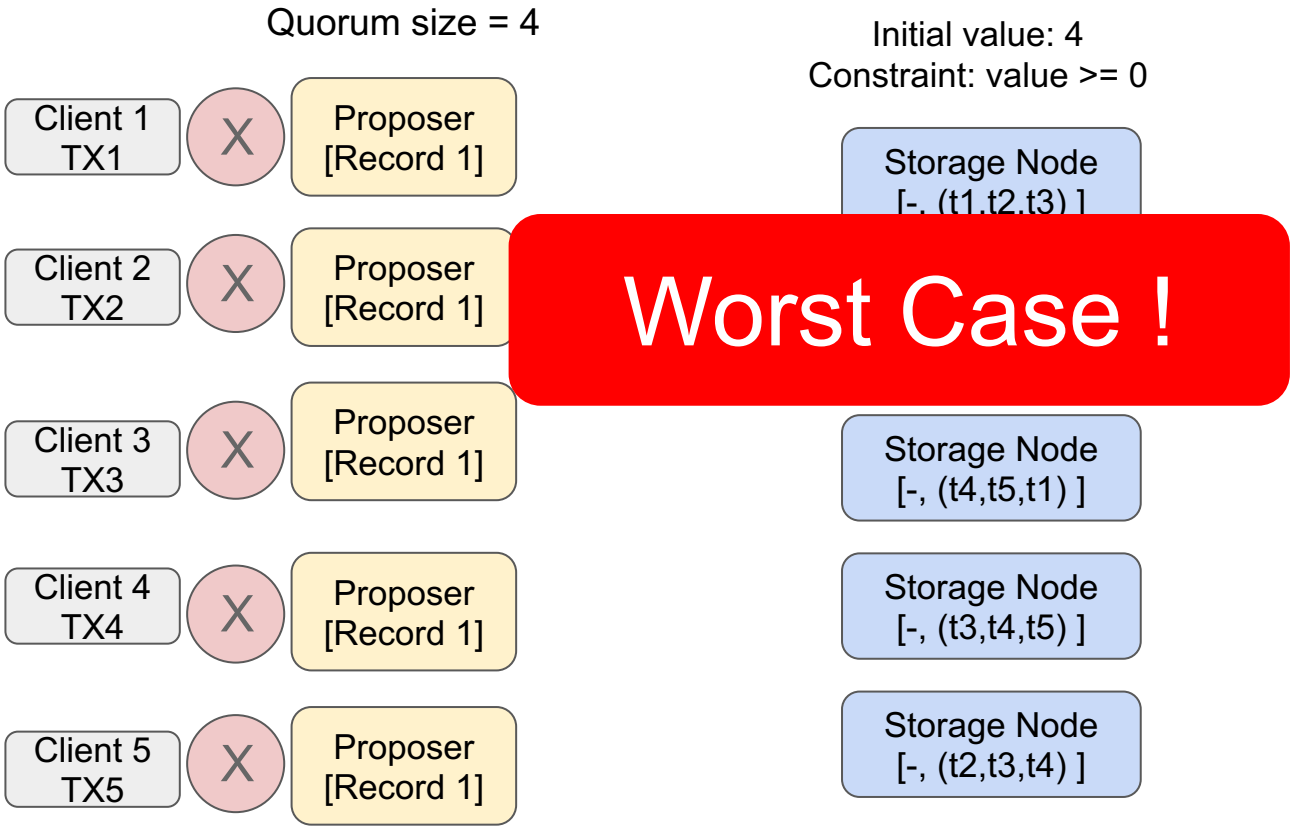
Initial value: 4
Constraint: value >= 0



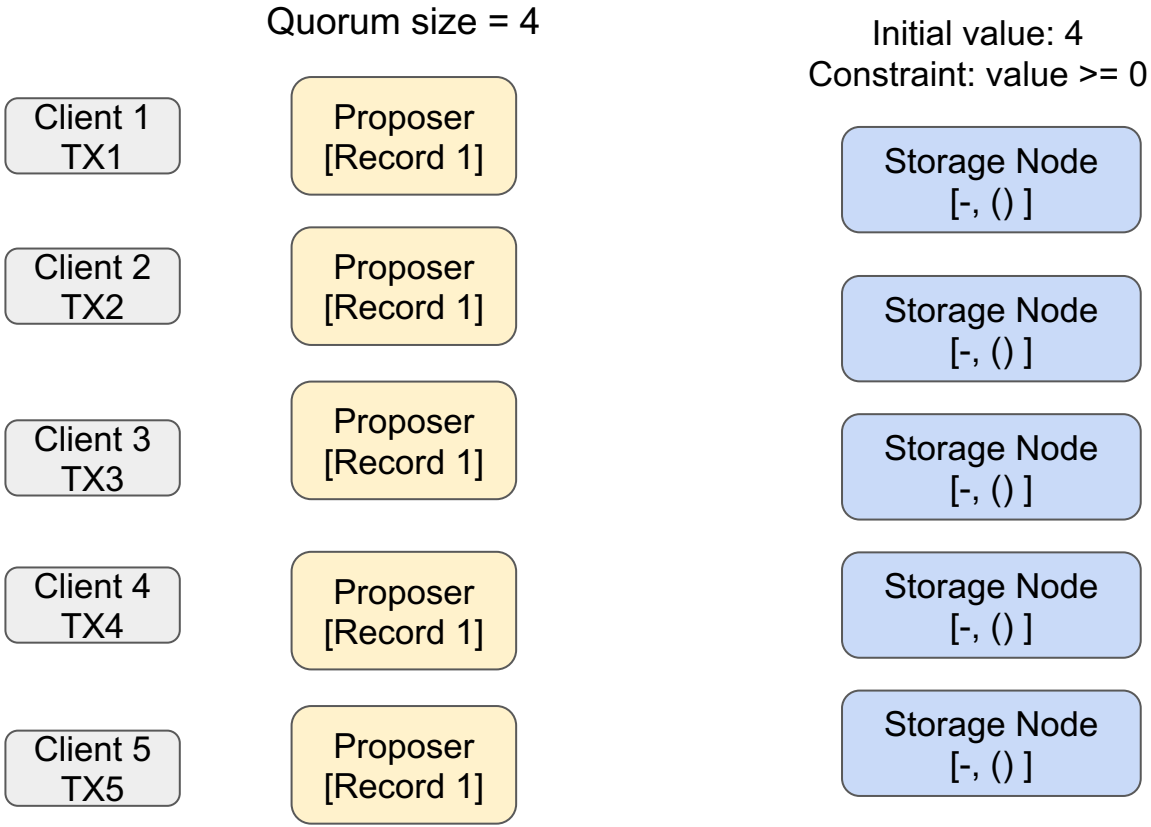
MDCC – New Demarcation Protocol



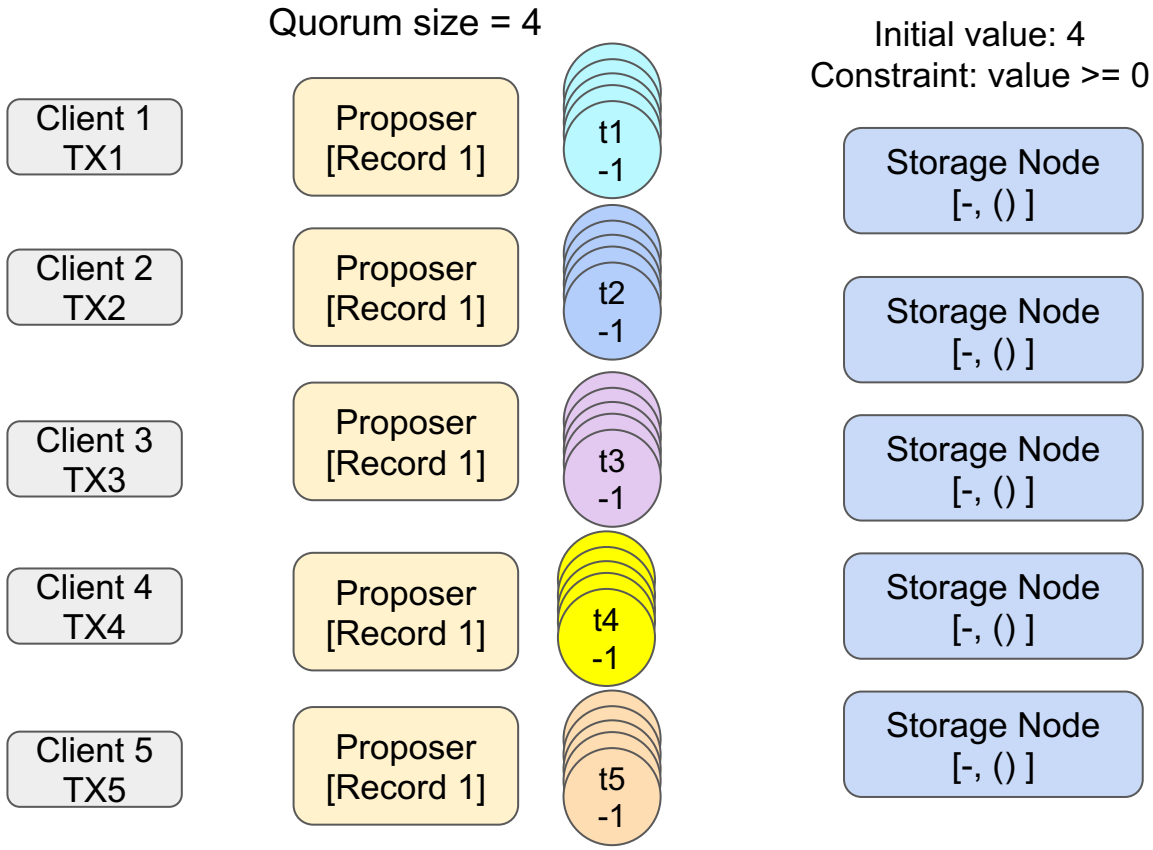
MDCC – New Demarcation Protocol



MDCC – New Demarcation Protocol



MDCC – New Demarcation Protocol

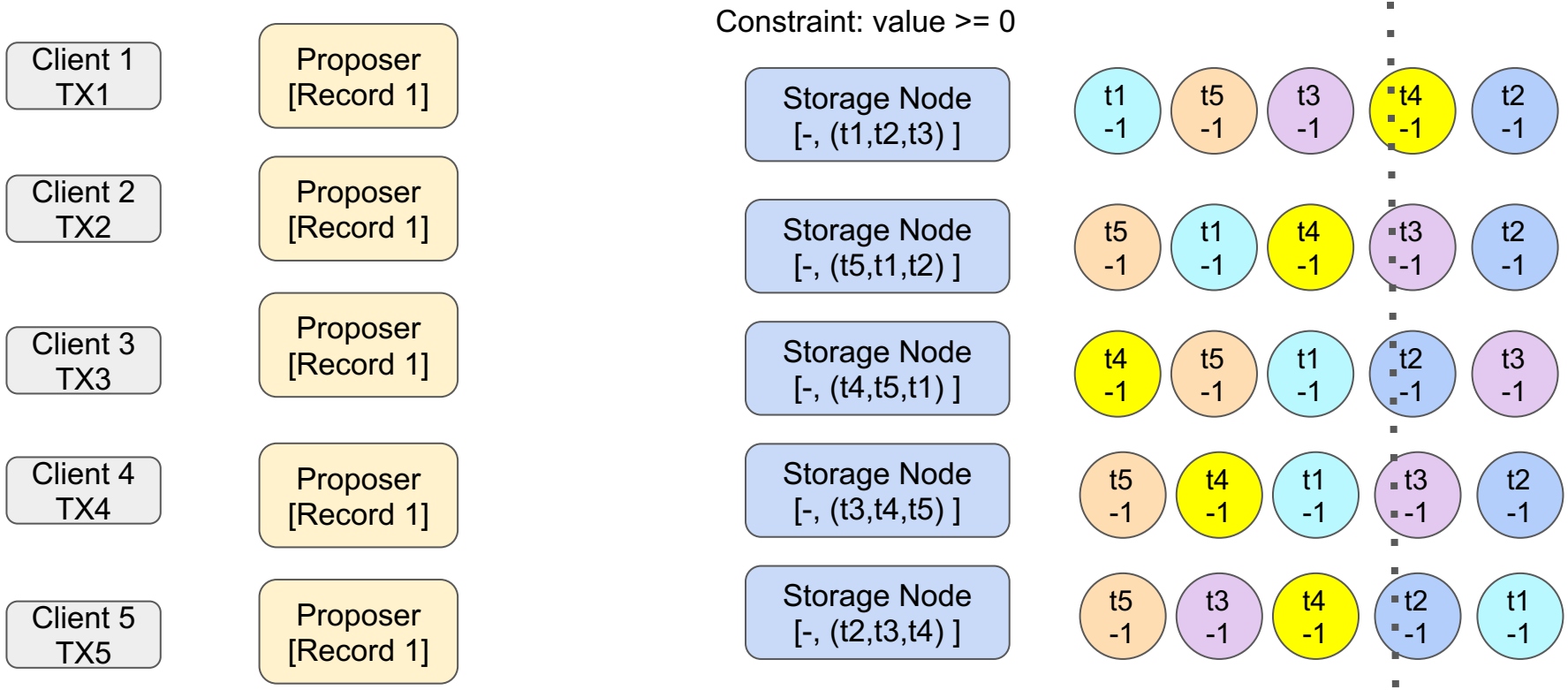


MDCC – New Demarcation Protocol

Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

Initial value: 4
Constraint: value >= 0

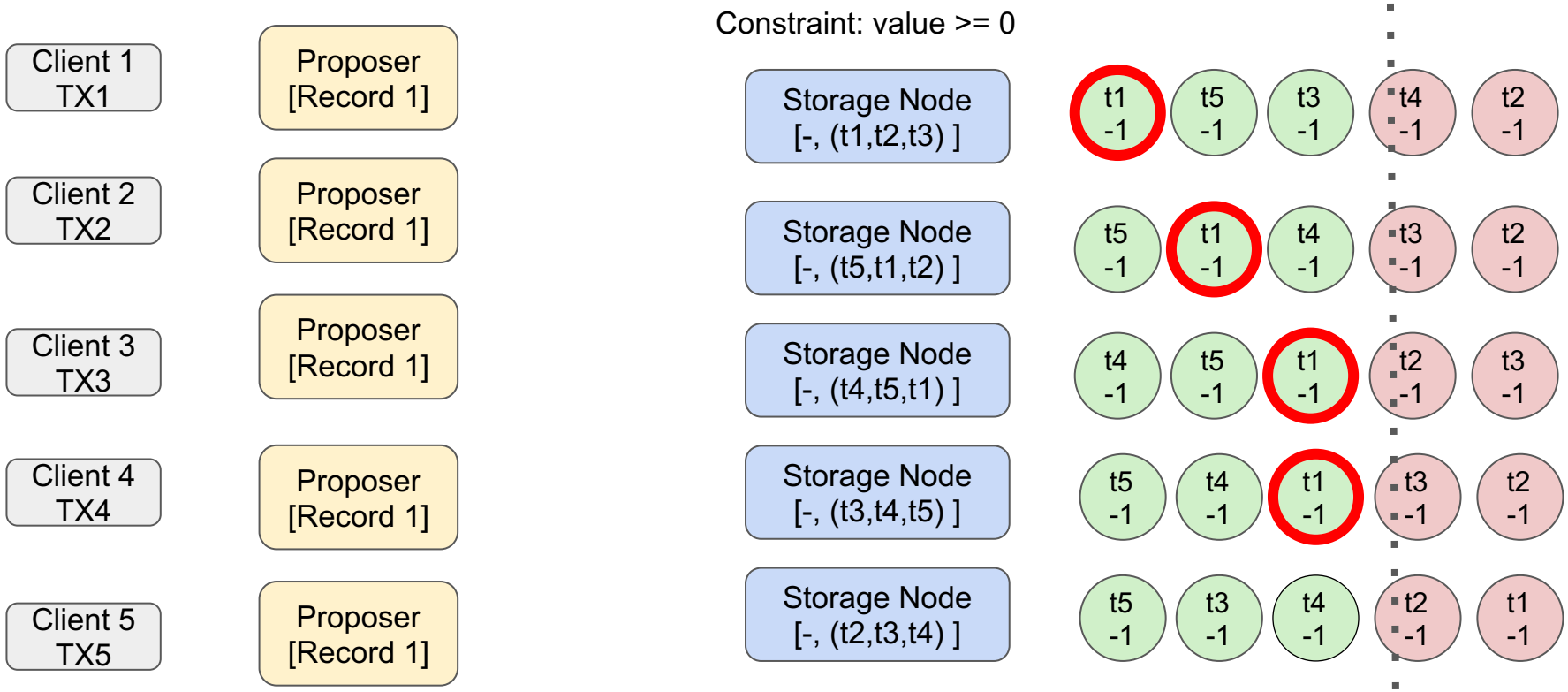


MDCC – New Demarcation Protocol

Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

Initial value: 4
Constraint: value >= 0

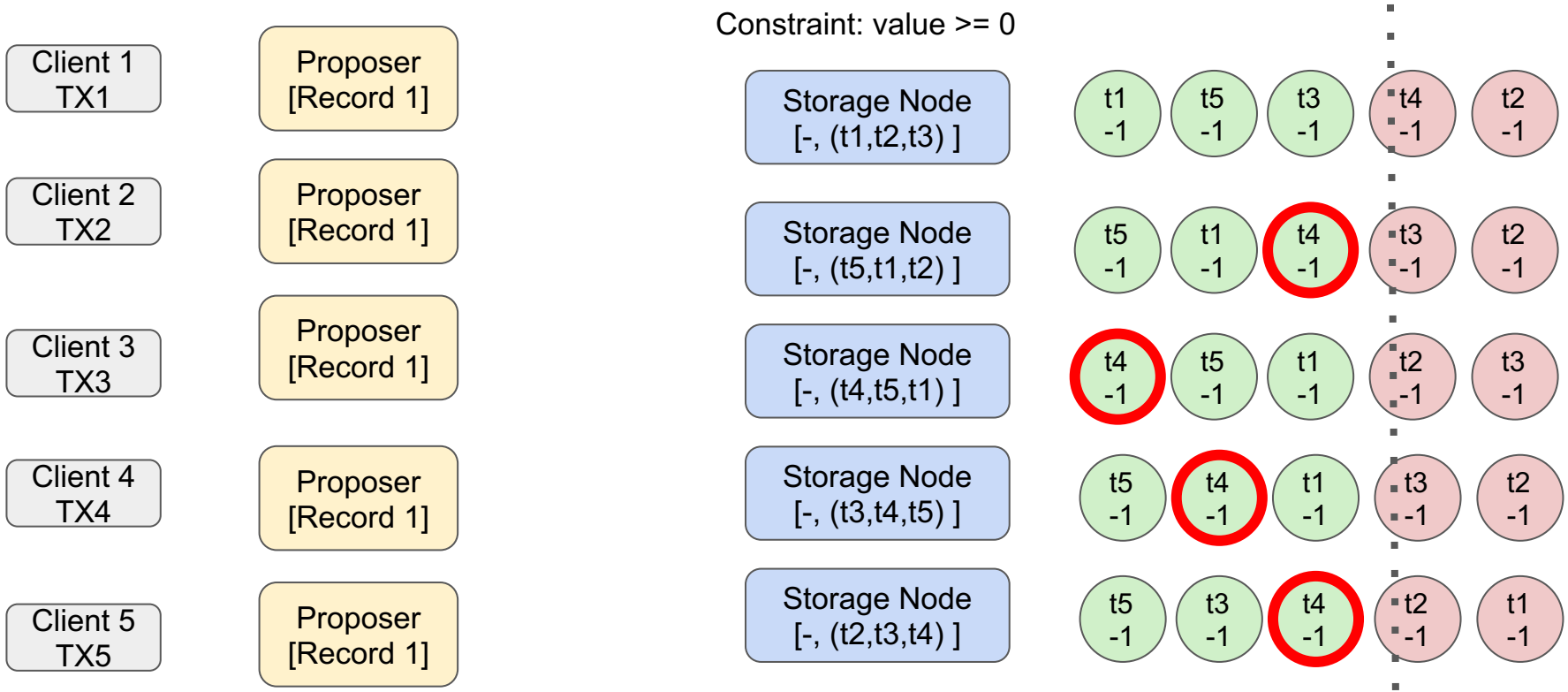


MDCC – New Demarcation Protocol

Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

Initial value: 4
Constraint: value >= 0

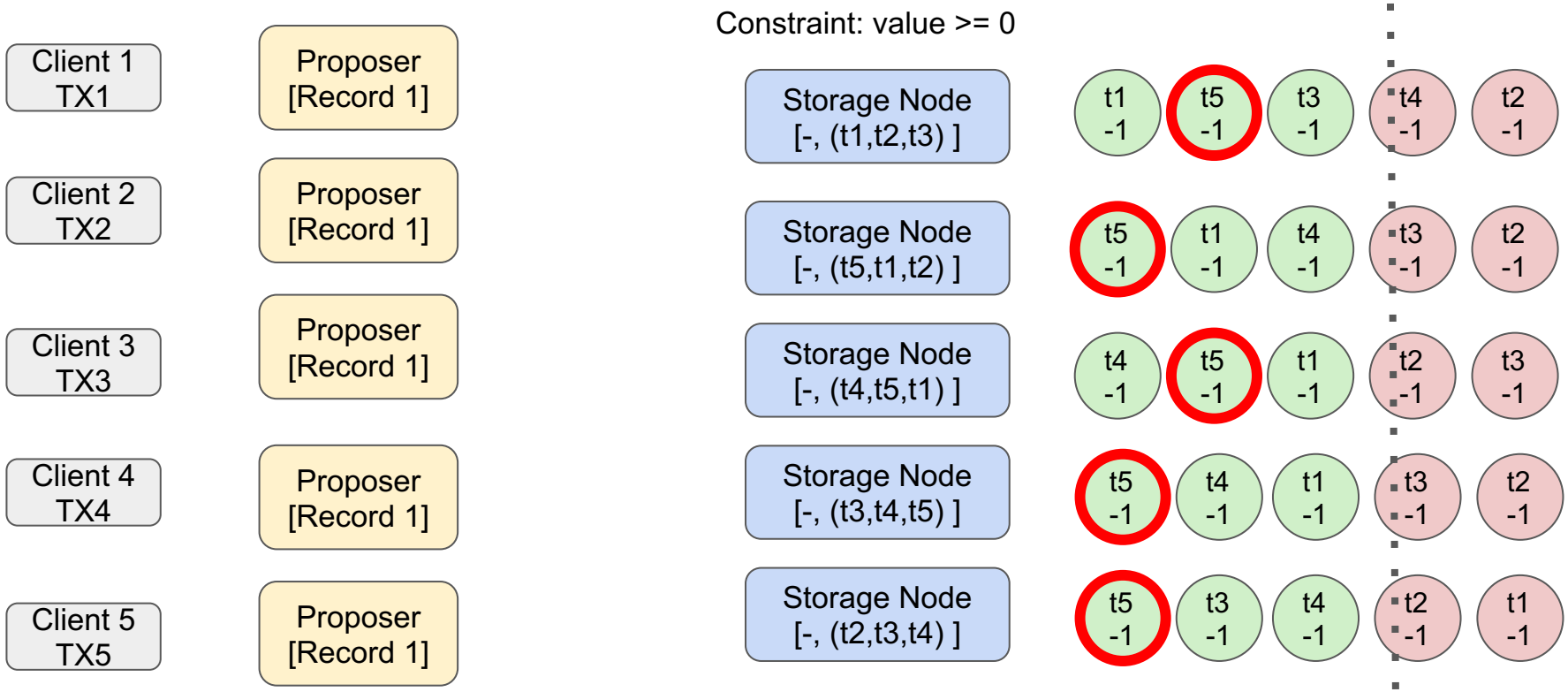


MDCC – New Demarcation Protocol

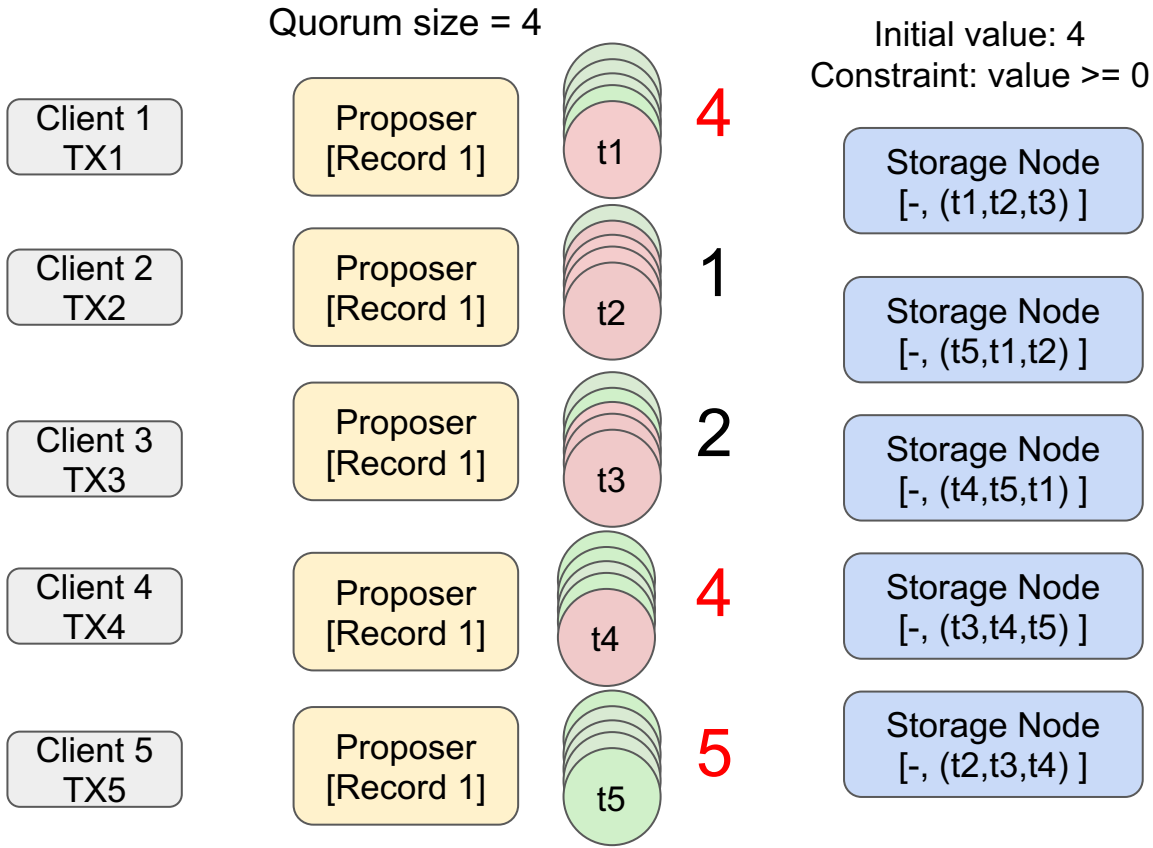
Limit >= (N-Q)/N
>= (5-4)/5
>= ⅕ = 0.8

Quorum size = 4

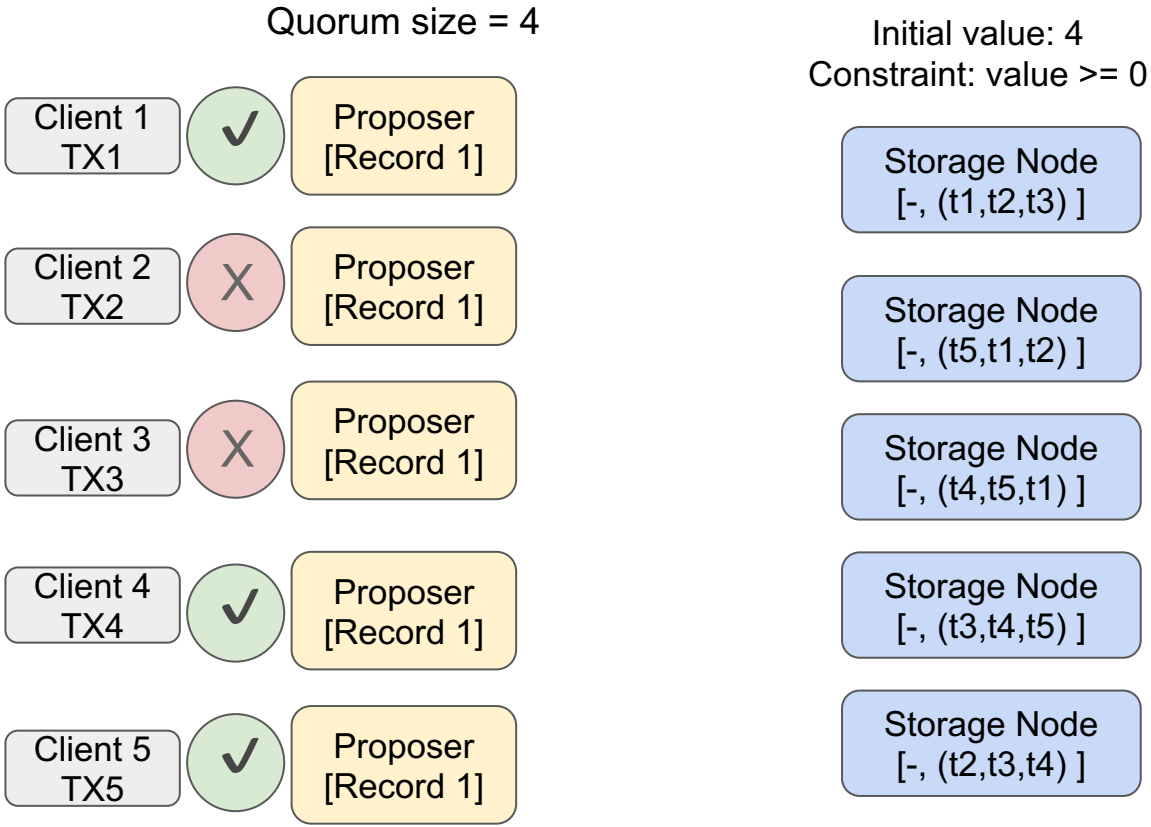
Initial value: 4
Constraint: value >= 0



MDCC – New Demarcation Protocol



MDCC – New Demarcation Protocol



Consistency Guarantees

Consistency Guarantees

- Read Committed without Lost Updates
  - Lost update
    - Ex: T1 read data X
      T2 write data X
      T1 write data X (overwrite data X)
  - Only allow to read committed changes
  - Solve lost updates by **detecting write-write conflicts**

Consistency Guarantees

- Staleness
  - Stale data
  - Require reading from majority of nodes
  - Techniques from Megastore
    - **Pseudo-master storage node**
    - They are **part of the quorum** of phase1 and 2

Consistency Guarantees

- Atomic Visibility
  - Only atomic durability
  - To support atomic visibility
    - Two-phase locking
    - Snapshot isolation

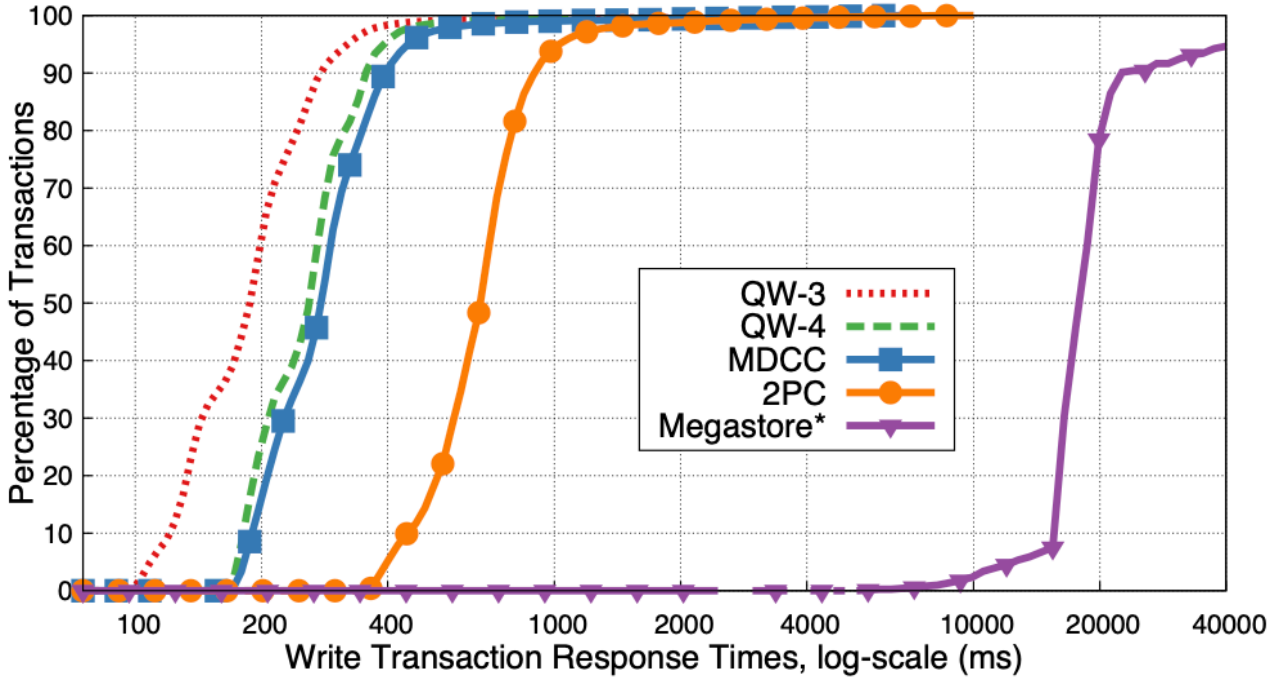
Evaluation

Evaluation

- TPC-W benchmark
  - for general performance
- Micro benchmark
  - for protocol investigation

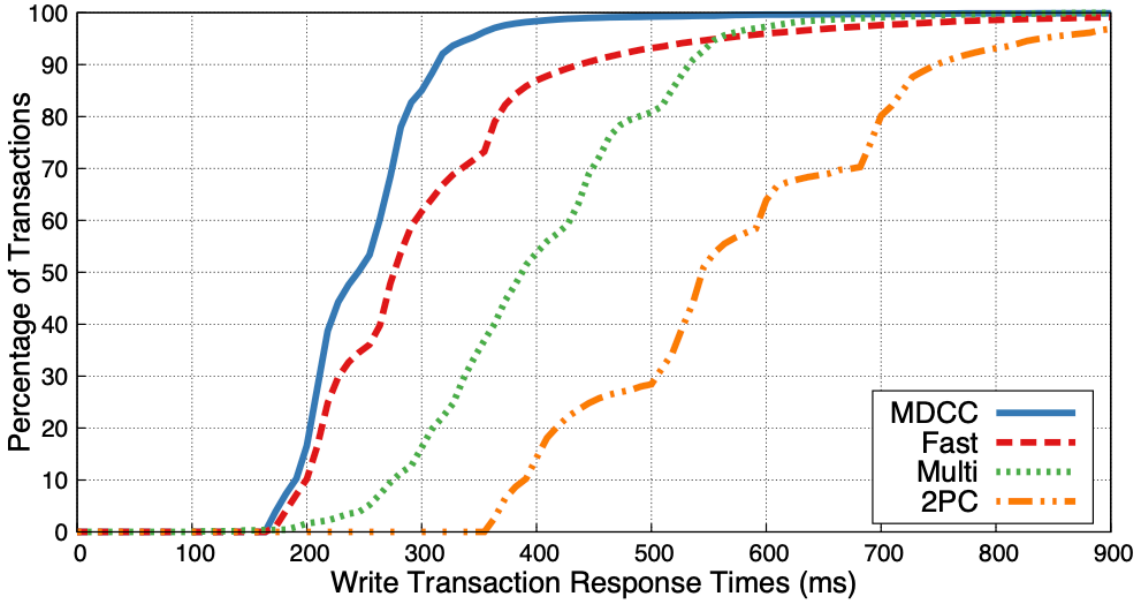
Evaluation

- TPC-W benchmark



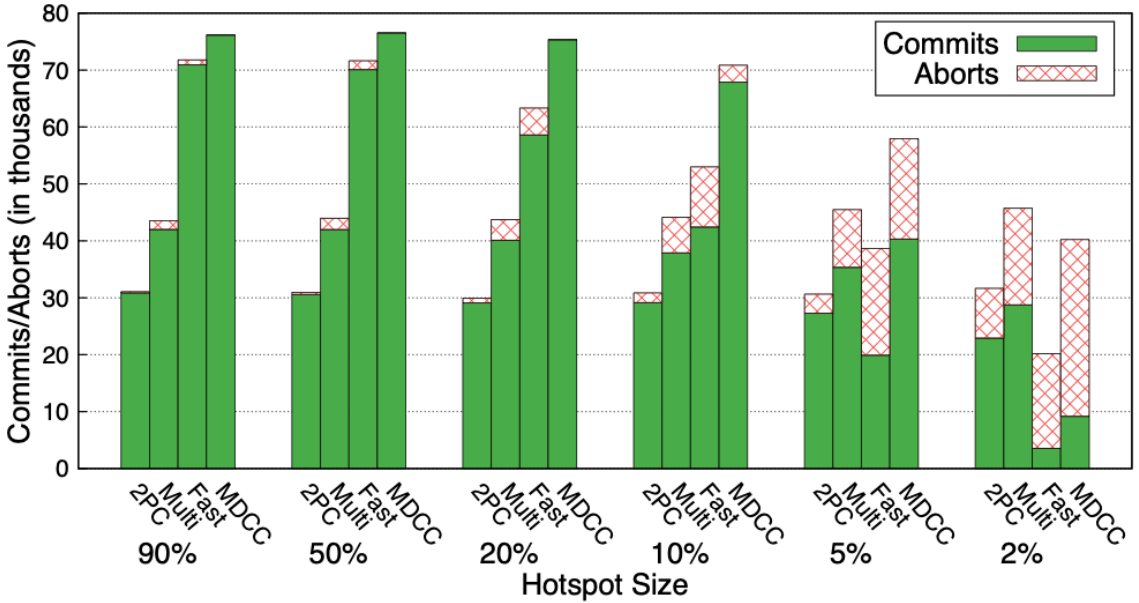
Evaluation

- Micro benchmark - write response times CDF



Evaluation

- Micro benchmark - varying conflict rate



Conclusion

- MDCC - Multi-Data Center Consistency
  - Optimization protocol
    - Conflicts are rare
    - Updates are commutative
  - Requires only 1 message round
  - Strong consistency
  - Low latency