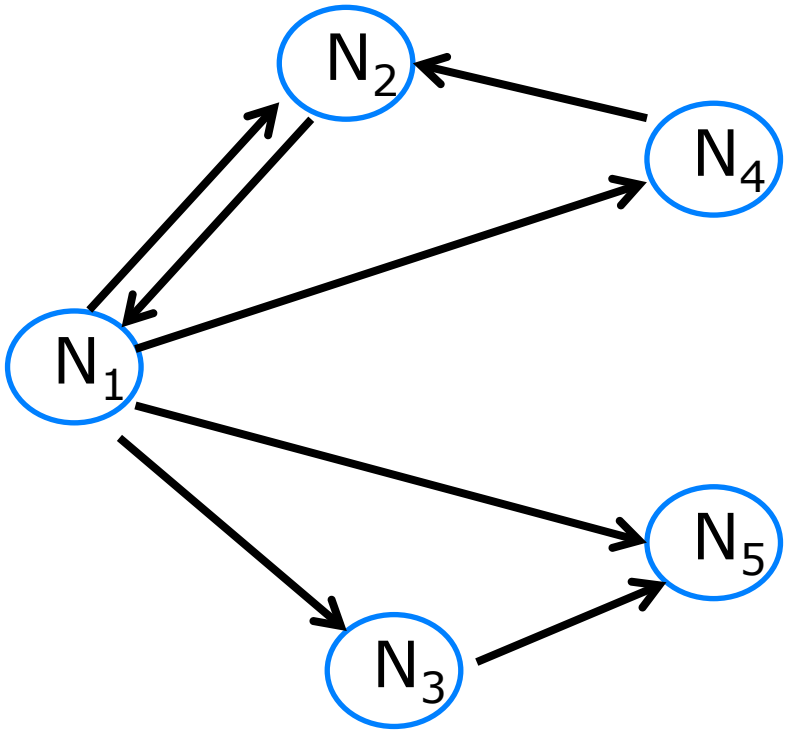
Graphs and more

Nuno Preguiça (CITI/UNL)

with Marc Shapiro (LIP6/INRIA), Carlos Baquero (UM), Marek Zawirski (LIP6), Valter Balegas (UNL)

Directed Graph



- Set of nodes

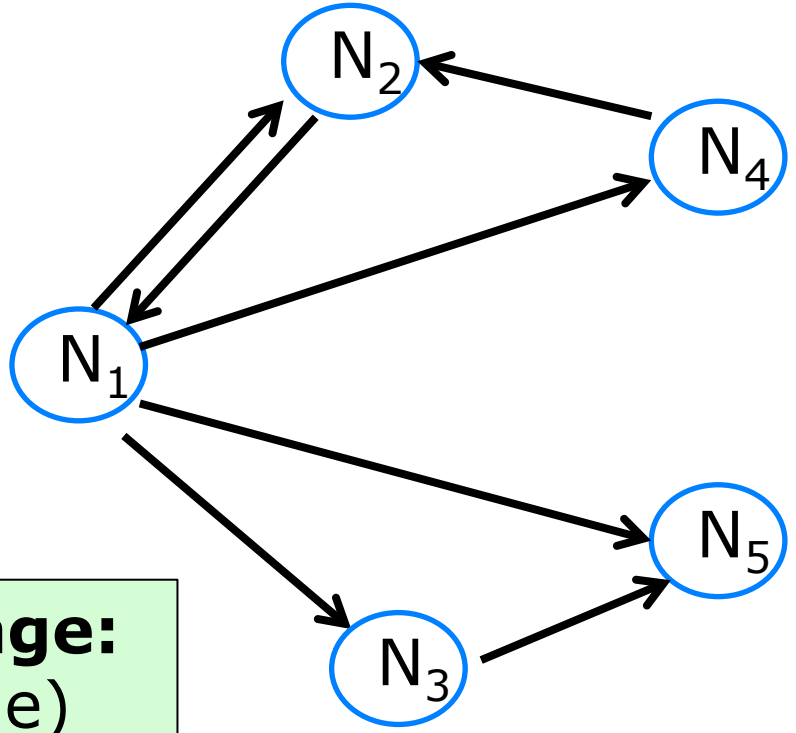
  {N1,N2,N3,N4,N5}

- Set of arcs

  {(N1,N2), (N1,N3),
  (N1,N4), (N1,N5),
  (N2,N1), (N3,N5),
  (N4,N2)}

Application scenario: web search engine

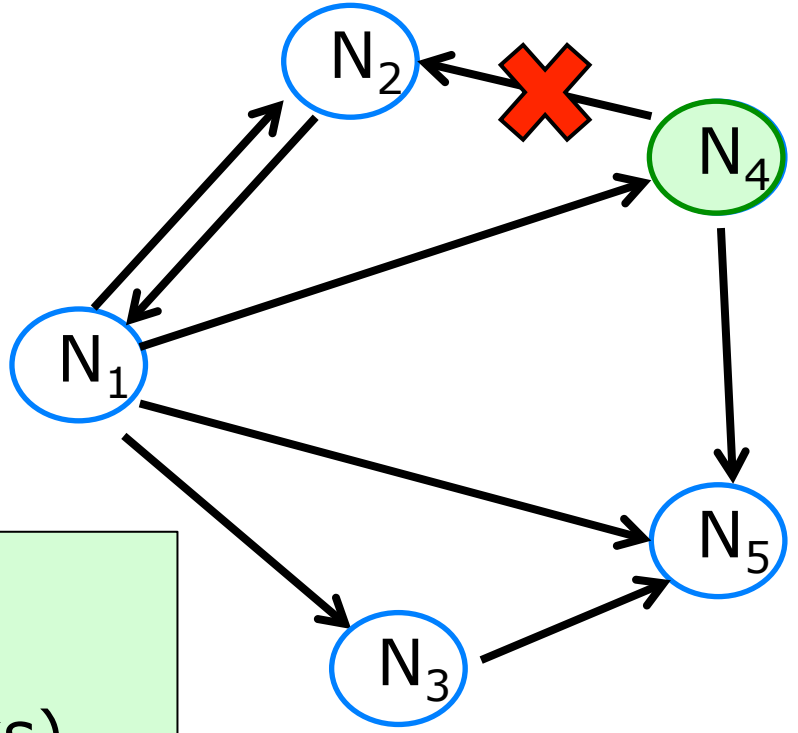
- Represent the web structure using a directed graph
- Web pages processed concurrently by multiple servers
- Incremental processing



When processing a web page:
1. Add node (for the web page)
2. Add arc (for links)
   NOTE: do not add tail node

Application scenario: web search engine

- Web evolves over time
  - Links change
  - Pages are removed
  - … and recreated
- Need to update the graph structure

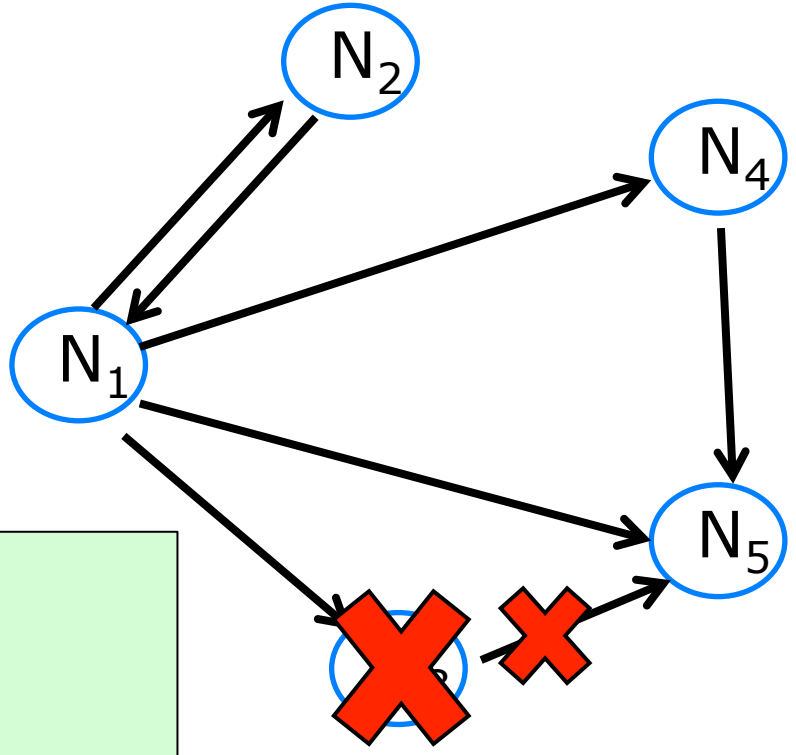


When re-processing an existing web page:

1. Add new arcs (for new links)
2. Remove arcs (for removed links)

Application scenario: web search engine

- Web evolves over time
  - Links change
  - Pages are removed
  - … and recreated
- Need to update the graph structure

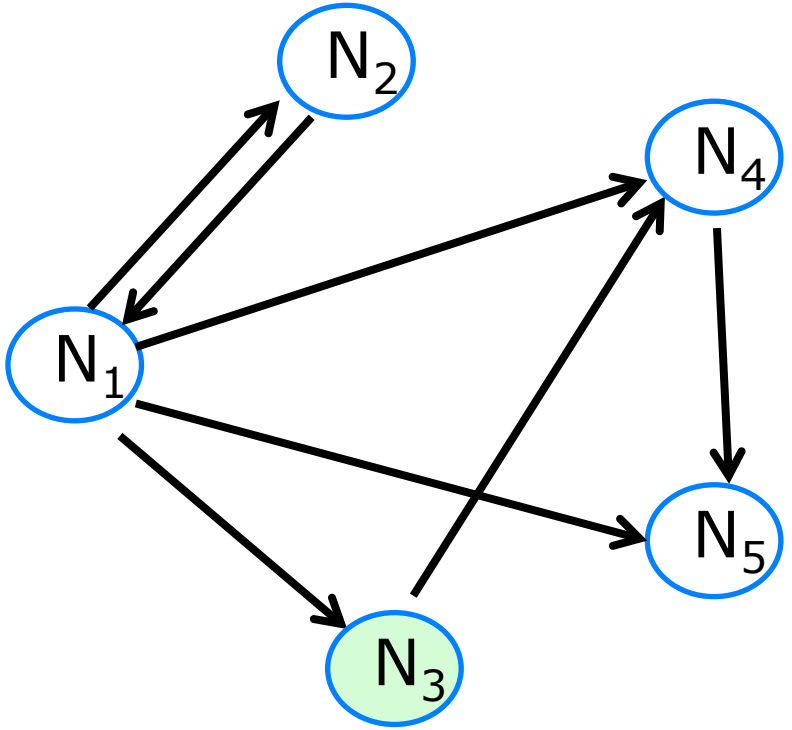


When re-processing a removed web page:

1. Remove arcs
2. Remove node

Application scenario: web search engine

- Web evolves over time
  - Links change
  - Pages are removed
  - … and recreated
- Need to update the graph structure



Graph data type

- Two sets: nodes + arcs
- Update operations:
  - addNode(node)
  - removeNode(node)
  - addArc(head, tail)
  - removeArc(head, tail)
- Query operations:
  - lookupNode(node): boolean
  - lookupArc(head, tail): boolean
  - listNode()
  - listArc()

Semantics on concurrent operations

- addNode(n) || addNode(n)
  - n is added to the set of nodes
- removeNode(n) || removeNode(n)
  - n is removed to the set of nodes
- addNode(n) || removeNode(n)
  - Alternatives:

1. n added to the set of nodes	OR-set
2. n removed from the set of nodes	OR-remove-set
3. result depends on (real-time or logical) timestamps	LWW-set
4. result depends on the number of times operation has been executed	counter-set (Weiss et. al.)

Semantics on concurrent operations (cont.)

- addArc(n1,n2) || addArc(n1,n2)
  - (n1,n2) is added to the set of arcs
- removeArc(n1,n2) || removeArc(n1,n2)
  - (n1,n2) is removed to the set of nodes
- addArc(n1,n2) || removeArc(n1,n2)
  - Alternatives:

1. (n1,n2) added to the set of nodes	OR-set
2. (n1,n2) removed from the set of nodes	OR-remove-set
3. result depends on (real-time or logical) timestamps	LWW-set
4. result depends on the number of times operation has been executed	counter-set (Weiss et. al.)

Semantics on concurrent operations (cont.)

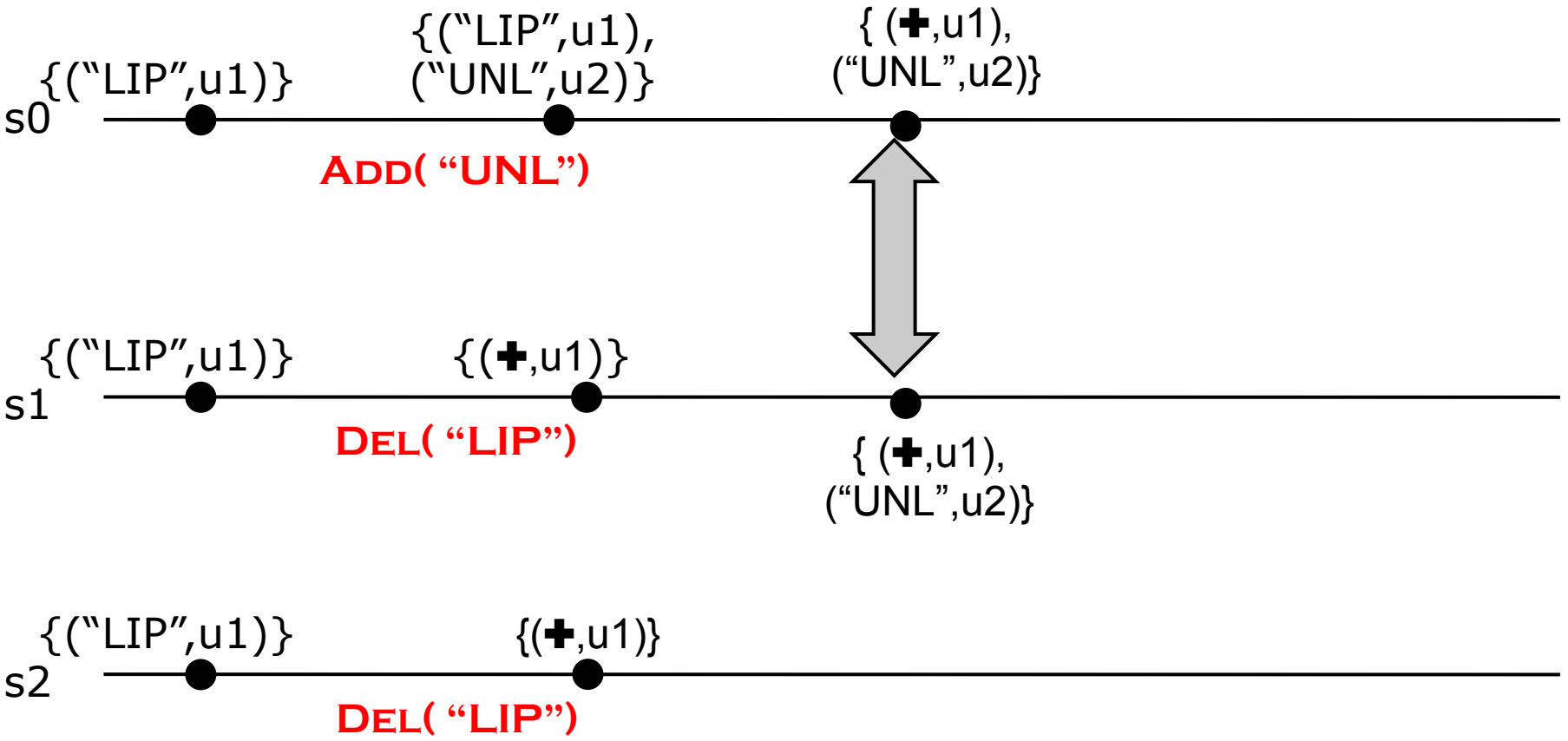
- addNode(n) || addArc(n1,n2), n=n1 V n=n2
  - n added, (n1,n2) added
- removeNode(n) || removeArc(n1,n2), n=n1 V n=n2
  - n remove, (n1,n2) removed
- addNode(n) || removeArc(n1,n2), n=n1 V n=n2
  - n added, (n1,n2) removed
- removeNode(n) || addArc(n1,n2), n=n1 V n=n2
  - n remove, (n1,n2) added but hidden

lookupArc((n1,n2)) : return (n1,n2) $\in$ A $\wedge$ lookup(n1) $\wedge$ lookup(n2)

Observed-remove Set CRDT

- Semantics
  - Maintains a set of atoms (that are not necessarily unique)
  - On concurrent add and delete of the same atom, adds wins
- Operations:
  - Add(a) -> adds the atom to the set
  - Delete(a) [Pre: a ∈ set] -> removes the atom from the set
  - Lookup() –> returns the atoms in the set

Observed-remove Set CvRDT



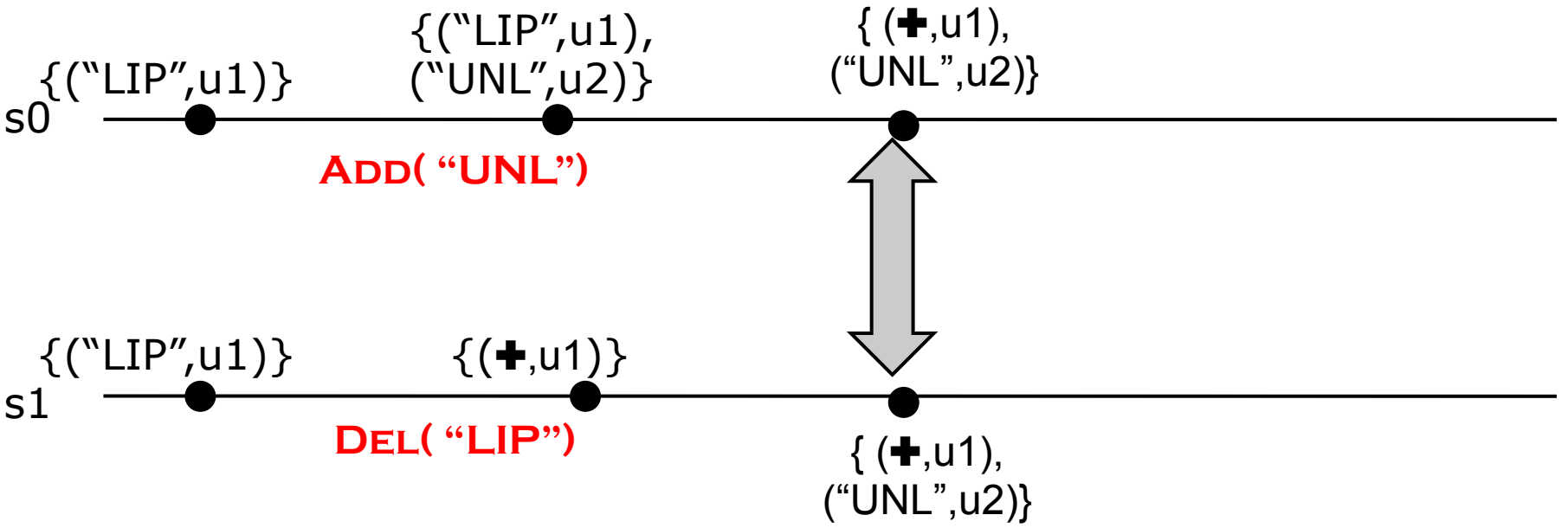
Observed-remove Set CvRDT

- State, S
  - S.A = set of pairs (a,uid_a)
  - S.R = set of tombstone uids

- Operations:
  - add (a) -> add((a,uid_a), old_uids), such that *old_uids={u:(a,u) ∈ S}*
    - *S.A := S U {(a,uid_a)} \ {(_,u): u ∈ old_uids} ; S.R = S.R U old_uids*
  - remove (a) -> remove(old_uids) , such that *old_uids={u:(a,u) ∈ S}*
    - *S.A := S \ old_uids ; S.R = S.R U old_uids*
  - lookup() : returns *{a:(a,_) ∈ S}*

- Merge (S1, S2) = S3:
  - S3.R = S1.R U S2.R
  - S3.A = (S1.A U S2.A) \ {(a,uid}: uid ∈ S3.R}

Observed-remove Set CvRDT

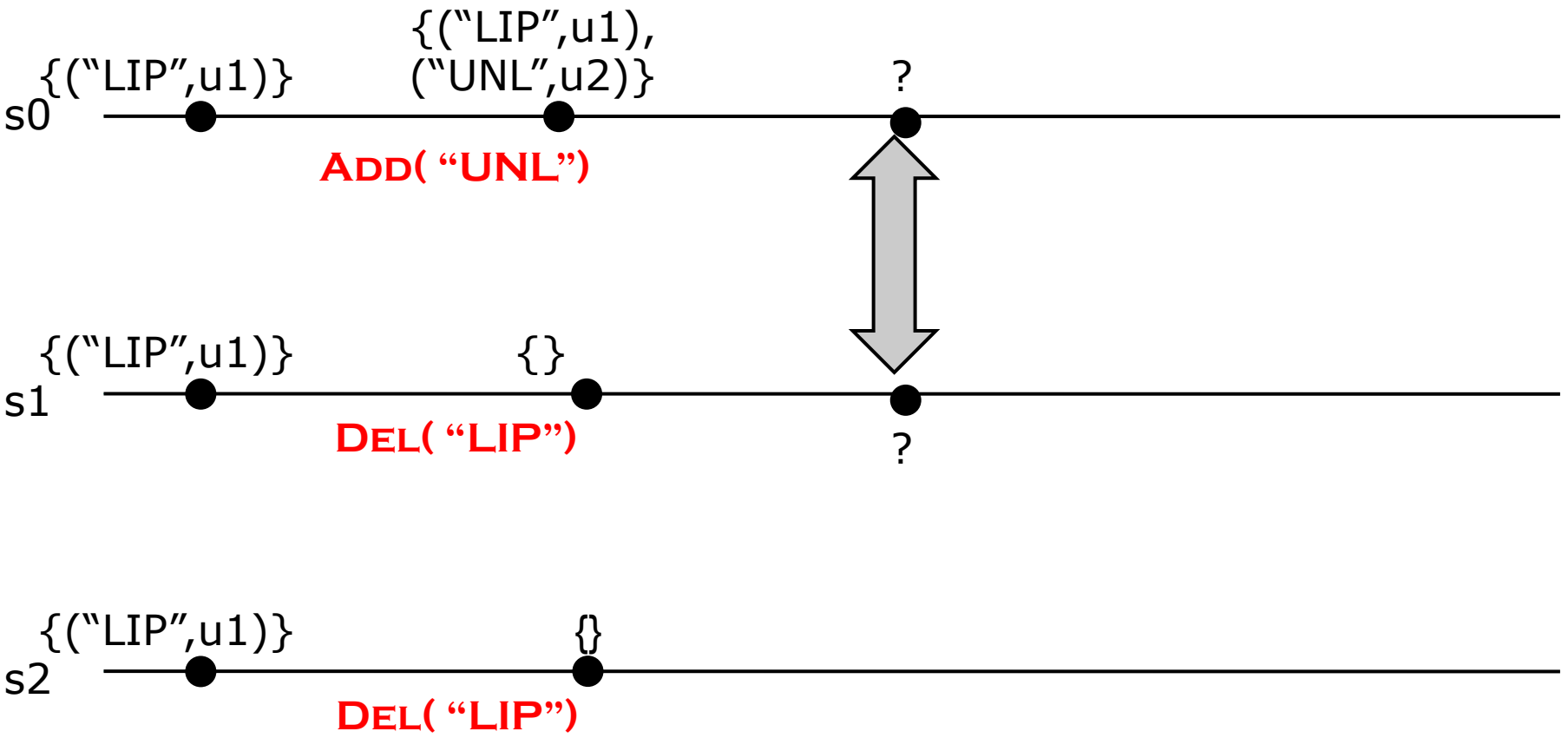


Can we garbage-collect deleted elements?

Garbage collection

- Context
  - CRDTs with operations add(uid), delete(uid)
    - E.g. Treedoc, Logoot, OR-set, etc.
- Problem
  - On executing *DELETE(UID)*, can we discard information on uid ?
  - What is the result of synchronization when uid ∈ Replica1 and NOT uid ∈ Replica2

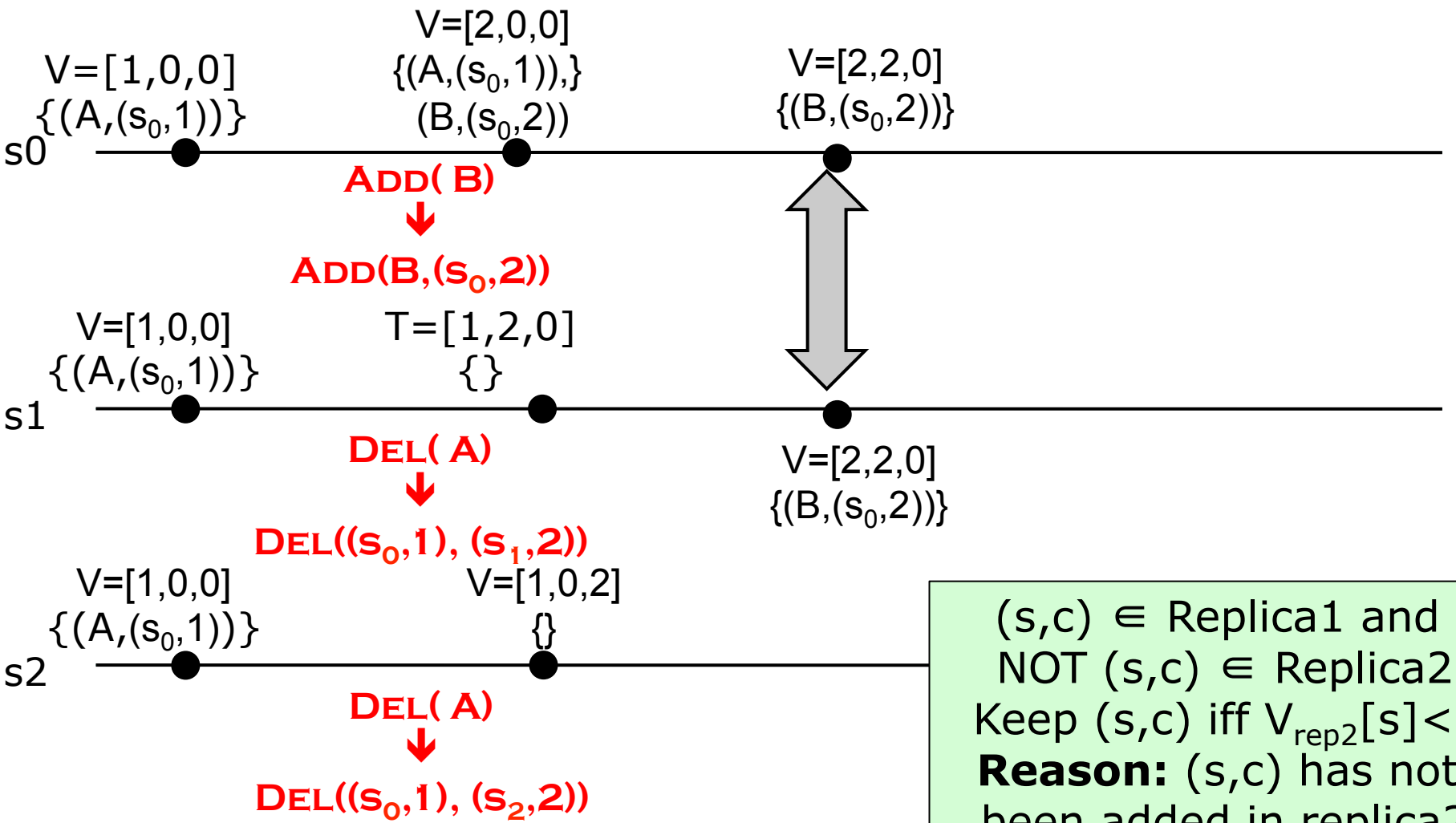
Observed-remove Set CvRDT



Garbage collection

- Context
  - CRDTs with operations add(uid), delete(uid)
    - E.g. Treedoc, Logoot, OR-set, etc.
- Problem
  - On executing *DELETE(UID)*, can we discard information on uid ?
  - What is the result of synchronization when uid ∈ Replica1 and NOT uid ∈ Replica2
- Idea
  - Generate UIDs in a way that they can be summarized in a simple way
  - Keep the summary of elements seen
- Solution
  - UID = (siteId,counter) Lamport timestamp
  - Summary of seen elements: version vector

Observed-remove Set CvRDT with GC



(s,c) ∈ Replica1 and NOT (s,c) ∈ Replica2
Keep (s,c) iff $V_{rep2}[s]<c$
Reason: (s,c) has not been added in replica2

GC basic algorithm

- State (at site i): version vector VV_i, set S_i, lamport clock c_i
- Operations:
  - add( ) -> add((s_i,c_i++))
    - *$S.S := S.S \cup \{(s_i,c_i)\}$; $VV_i[s_i] := c_i$*
  - remove((s,c)) -> remove((s,c)) , c_i++
    - *$S.S := S.S \setminus \{(s,c)\}$; $VV_i[s_i] := c$*

- Merge (S1, S2) = S3:
  - S3.VV = entry_max(S1.VV,S2.VV)
  - $S3.S = (S1.S \wedge S2.S) \cup \{(s,c) \in S1.S \setminus S2.S : S2.VV[s] < c\} \cup \{(s,c) \in S2.S \setminus S1.S : S1.VV[s] < c\}$

Garbage-collection

- Same approach can be used when tombstones are used
  - E.g. in any problem with operation pairs: add(uid), remove(uid)
- This includes Graphs, state-based versions of Treedoc, Logoot, etc.
- It is possible to use a single version-vector for a set of objects that are synchronized at the same time

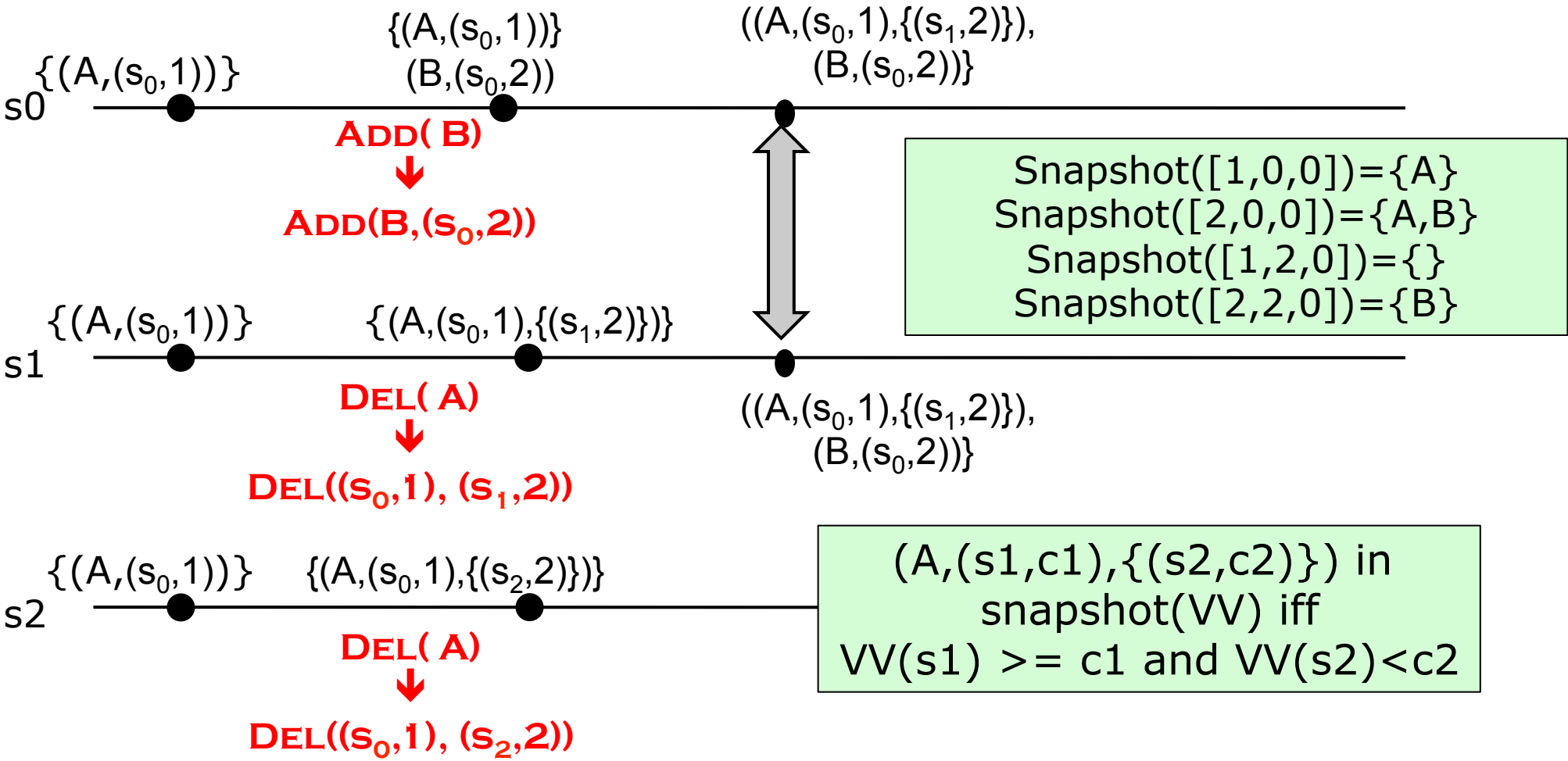
Need for snapshots

- Some computations require accessing a consistent state of the data
  - Transactions access a consistent view of the database (in most isolation levels, at least ☺)
- This applies to search engines
  - E.g. Google Percolator requires transactions with snapshot isolation

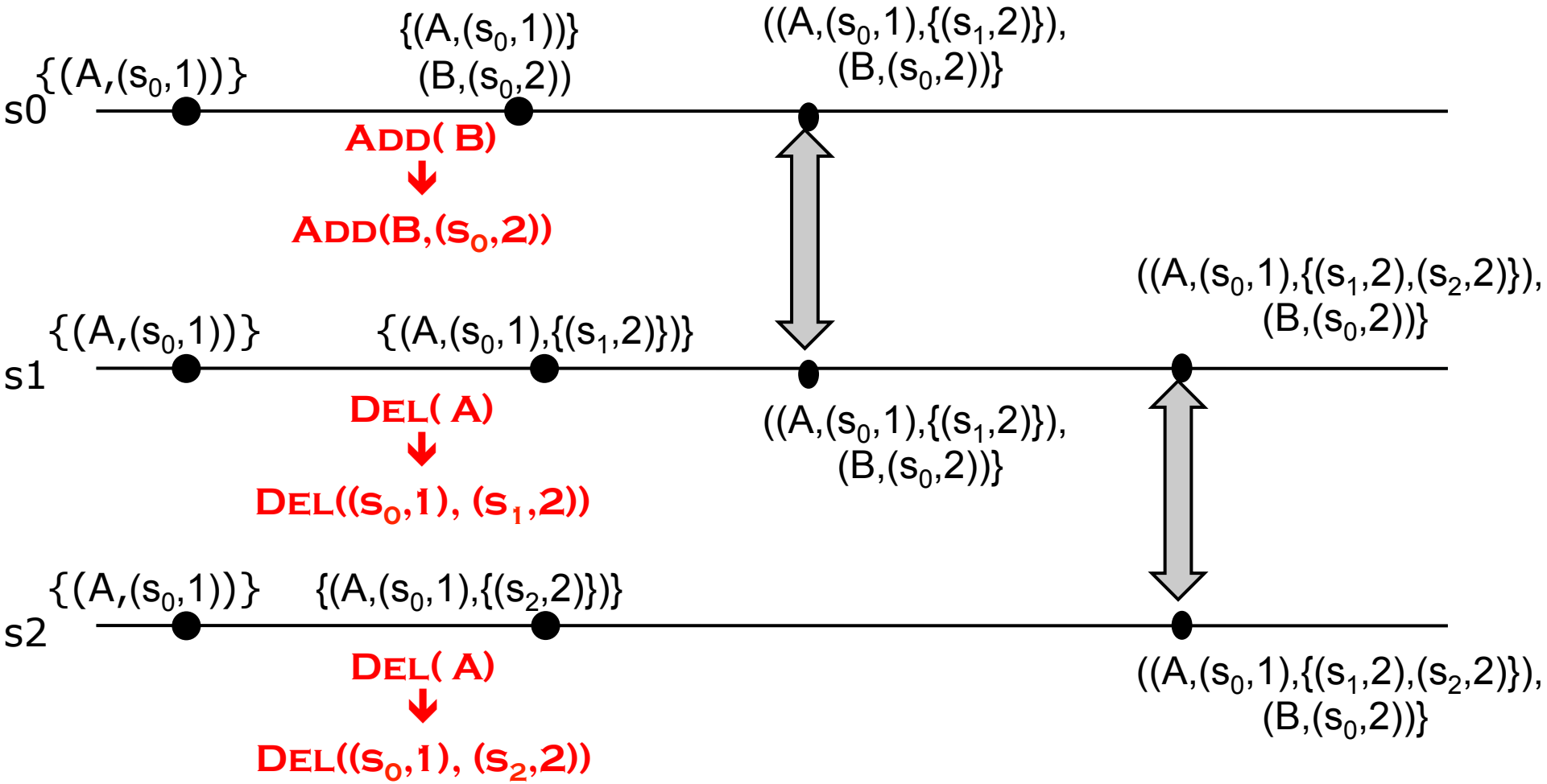
Snapshots in CRDTs

- Context
  - CRDTs we have built include global unique identifiers (UID)
- Idea
  - Generate these UIDs in a way that they can be used to define a snapshot easily
- Solution
  - UID = (siteId,counter) Lamport timestamp
  - Snapshot version defined by version vector
  - Need to keep tombstones (while a snapshot is being used)

Observed-remove Set CvRDT with snapshot



Observed-remove Set CvRDT with snapshot



Snapshot basic algorithm

- State (at site i): set S_i, lamport clock c_i
- Operations:
  - add(v) -> add(v,(s_i,c_i++))
    - *S.S := S.S U {(v,(s_i,c_i))}*
  - remove(v) -> remove((v,(s,c),(s_i,c_i++)))
    for each (s,c), such that , *(v,(s,c) ∈ S.S*
    - *S.S := S.S \ {(v,(s,c))} U {*(v,(s,c),(s_i,c_i))*}*
  - lookup(v,VV) -> ((v,(s,c)) ∈ S ∧ VV[s] >= c) ∨
    ((v,(s_a,c_a),(s_r,c_r)) ∈ S ∧ VV[s_a] >= c_a ∧ VV[s_r] < c_r)
- Merge (S1, S2):
  - S3.S = S1.S U S2.S \ {(v,a) ∈ S1.S: (v,a,r) ∈ S2.S)} \
    {(v,a) ∈ S2.S: (v,a,r) ∈ S1.S)}

Using snapshots in CRDTs

- Support access to consistent data snapshot in transactions
  - Information on deletions only necessary while processing transactions
  - Can be combined with GC support

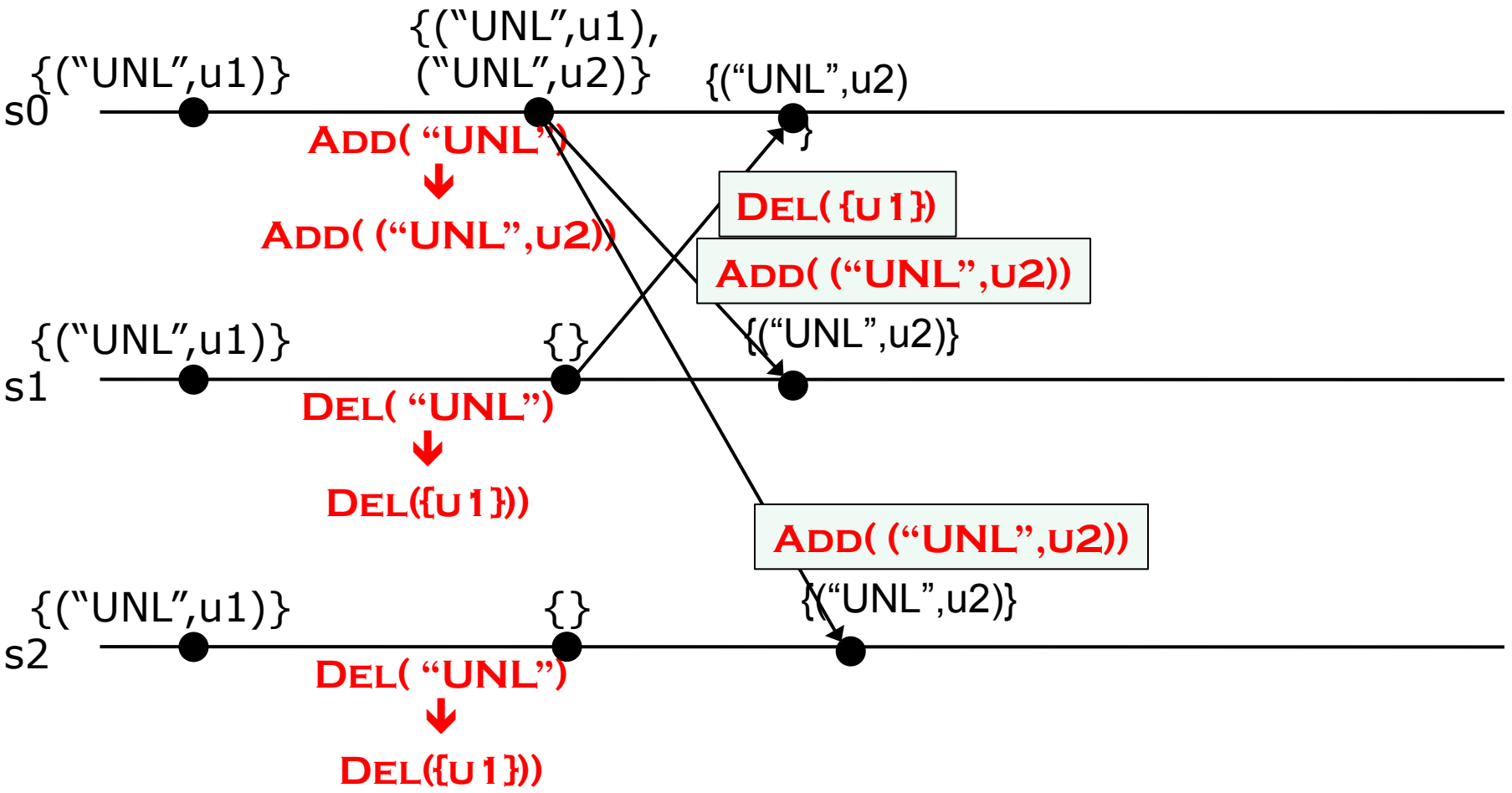
- Support access to data evolution history
  - Needs to keep all information – no garbage collection
  - Added to Treedoc

Final remarks

- Directed graph for web search application
  - Built using observed-remove set
- Garbage-collection optimization
  - Useful as replacement for tombstones
- Snapshot support
  - Support for transactions
  - Support for history browsing

Backup slides

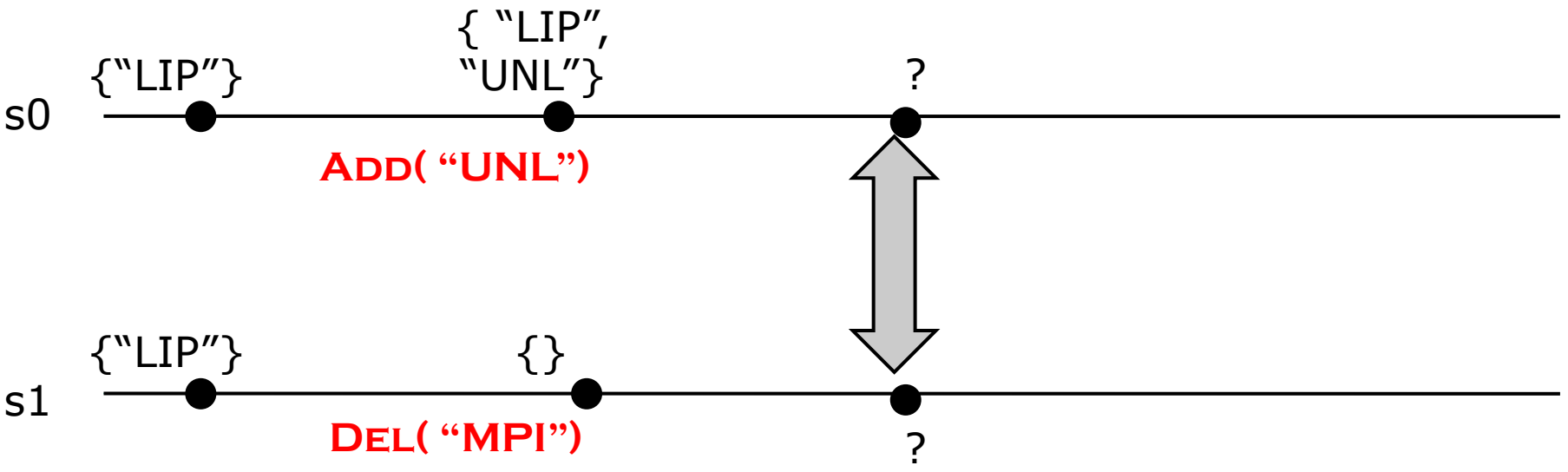
Observed-remove Set CmRDT



Observed-remove Set CmRDT

- State: set S of pairs (a,uid_a)
- Operations:
  - add (a) -> add((a,uid_a), old_uids), such that *old_uids={u:(a,u) ∈ S}*
    - *S := S U {(a,uid_a)}* \ {(_,u): u ∈ old_uids}
  - delete(a) -> delete(old_uids) , such that *old_uids={u:(a,u) ∈ S}*
    - *S := S \ old_uids*
    - *Idea: removes all observed uids for the given atom*
  - lookup() : returns *{a:(a,_) ∈ S}*

Observed-remove Set CvRDT



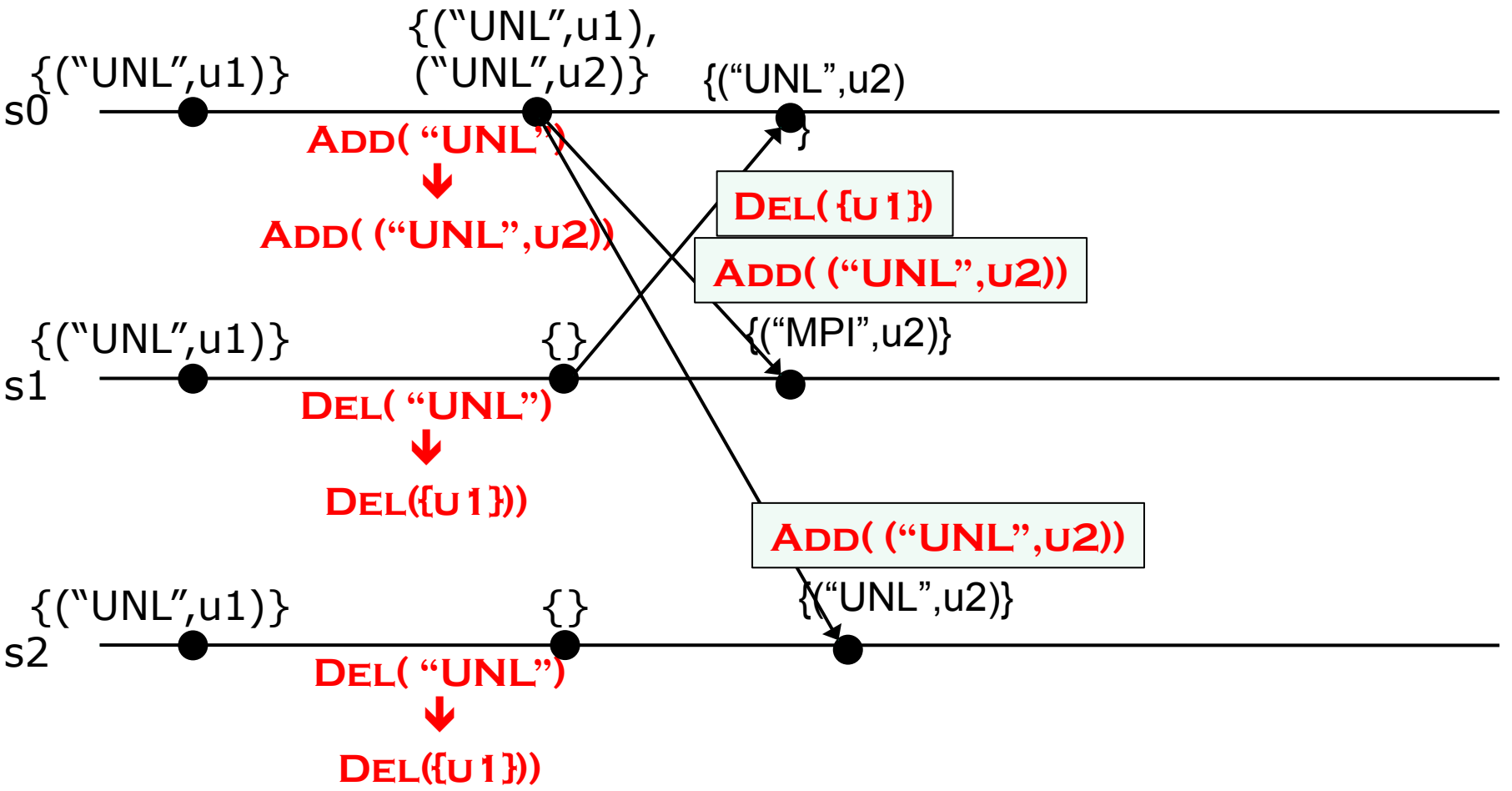
What should be the result?

Union leads to the Dynamo's shopping cart anomaly: {"UNL", "LIP"}

Observed-remove remove Set CRDT

- Semantics
  - Maintains a set of atoms (that are not necessarily unique)
  - On concurrent add and delete of the same atom, remove wins
- Operations:
  - Add(a) -> adds the atom to the set
  - Delete(a) [Pre: a ∈ set] -> removes the atom from the set
  - Lookup() –> returns the atoms in the set

Observed-remove remove Set CmRDT



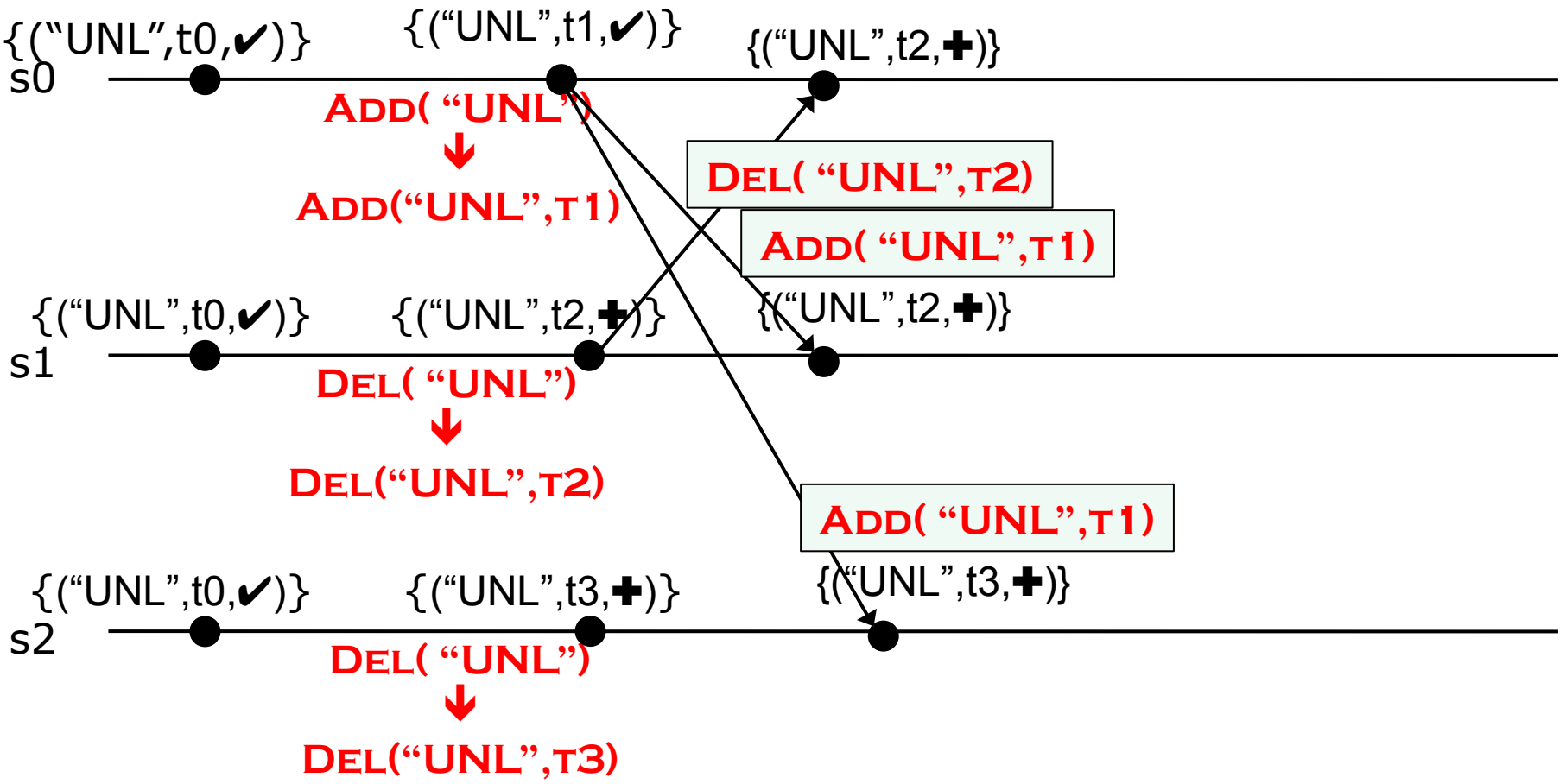
Observed-remove remove Set CmRDT

- State: set S of pairs (a,uid_a)
- Operations:
  - add (a) -> add((a,uid_a), old_uids), such that *old_uids={u:(a,u) ∈ S}*
    - *S := S U {(a,uid_a)}* \ {(_,u): u ∈ old_uids}
  - delete(a) -> delete(old_uids) , such that *old_uids={u:(a,u) ∈ S}*
    - *S := S \ old_uids*
    - *Idea: removes all observed uids for the given atom*
  - lookup() : returns *{a:(a,_) ∈ S}*

Observed-remove LWW Set CRDT

- Semantics
  - Maintains a set of atoms (that are not necessarily unique)
  - On concurrent add and delete of the same atom, last writer operation wins
- Operations:
  - Add(a) -> adds the atom to the set
  - Delete(a) [Pre: a ∈ set] -> removes the atom from the set
  - Lookup() –> returns the atoms in the set

Observed-remove LWW Set CmRDT



Observed-remove LWW Set CmRDT

- State: set S of tuples (a,ts, ✔|✚)
- Operations:
  - add (a) -> add(a, ts), such that *ts is a globally ordered clock – e.g. Lamport clock, real-time*

    if (a,ts_old,s) ∈ S then

    if ts_old < ts then S = S *U {(a,ts,✔)}* \ *{(a,ts_old,s)} endif*

    else S = S *U {(a,ts,✔)}*
  - delete(a) -> delete(a, ts), such that *ts is a globally ordered clock – e.g. Lamport clock, real-time*delete(old_uids) , such that *old_uids={u:(a,u) ∈ S}*

    if (a,ts_old,s) ∈ S then

    if ts_old < ts then S = S *U {(a,ts,✚)}* \ *{(a,ts_old,s)} endif*

    else S = S *U {(a,ts,✚)}*
  - lookup() : returns *{a:(a,_,✔) ∈ S}*