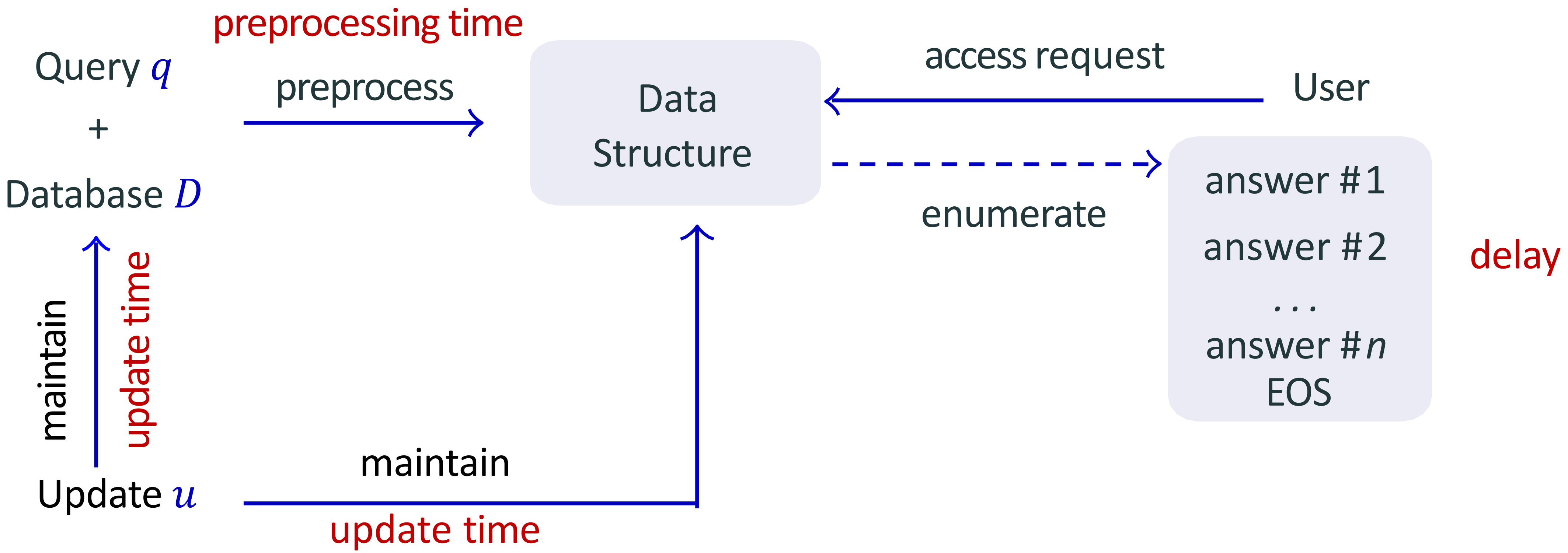
Incremental View Maintenance for Conjunctive Queries (Beyond Worst-Case Analysis)

Xiao Hu

Simons Institute

Problem Definition

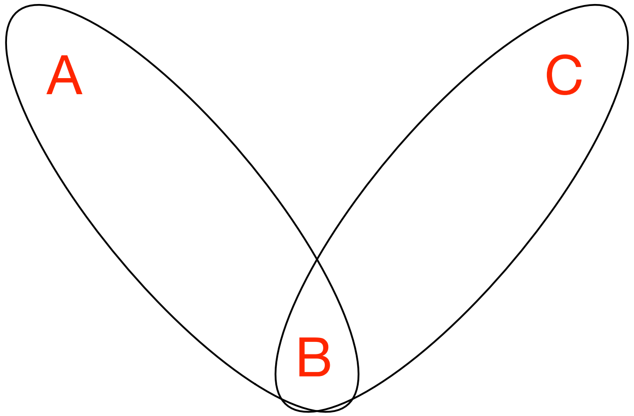


A data structure that can be preprocessed and updated efficiently while supporting constant-delay enumeration

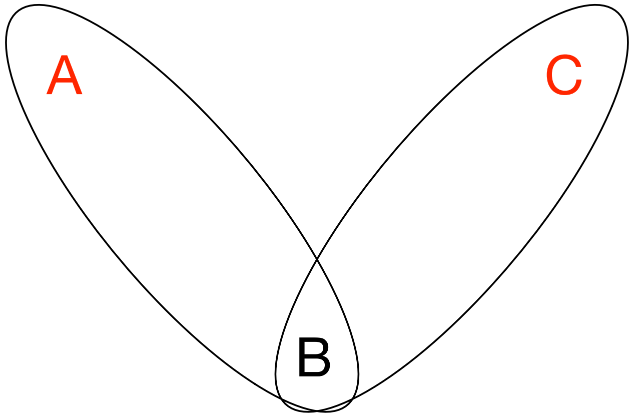
Hardness

- The problem is hard [BKS17]:
  - The update time is at least $\Omega(\sqrt{N})$
  - $O(1)$ update time is impossible unless the query is q-hierarchical

q-hierarchical query:
$R_1(A,B) \bowtie R_2(B,C)$

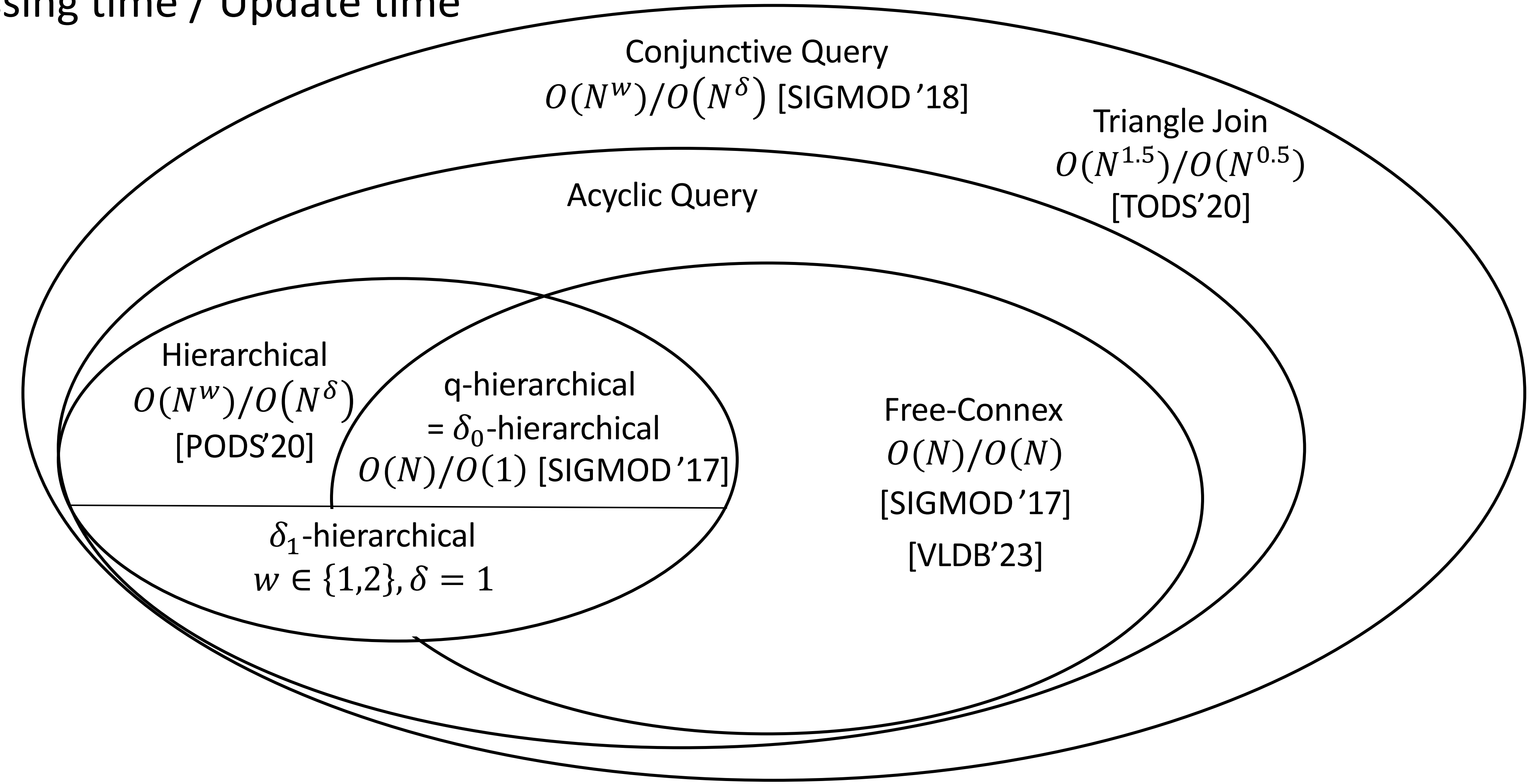


non-q-hierarchical query:
$\pi_{AC}R_1(A,B) \bowtie R_2(B,C)$



A Partial Landscape (from Bootcamp)

- Preprocessing time / Update time



Observations

- The class of q-hierarchical query is still small

- The upper bound is not very satisfactory
  - q-hierarchical: $O(1)$ update time
  - Non-q-hierarchical: $O(N)$ update time

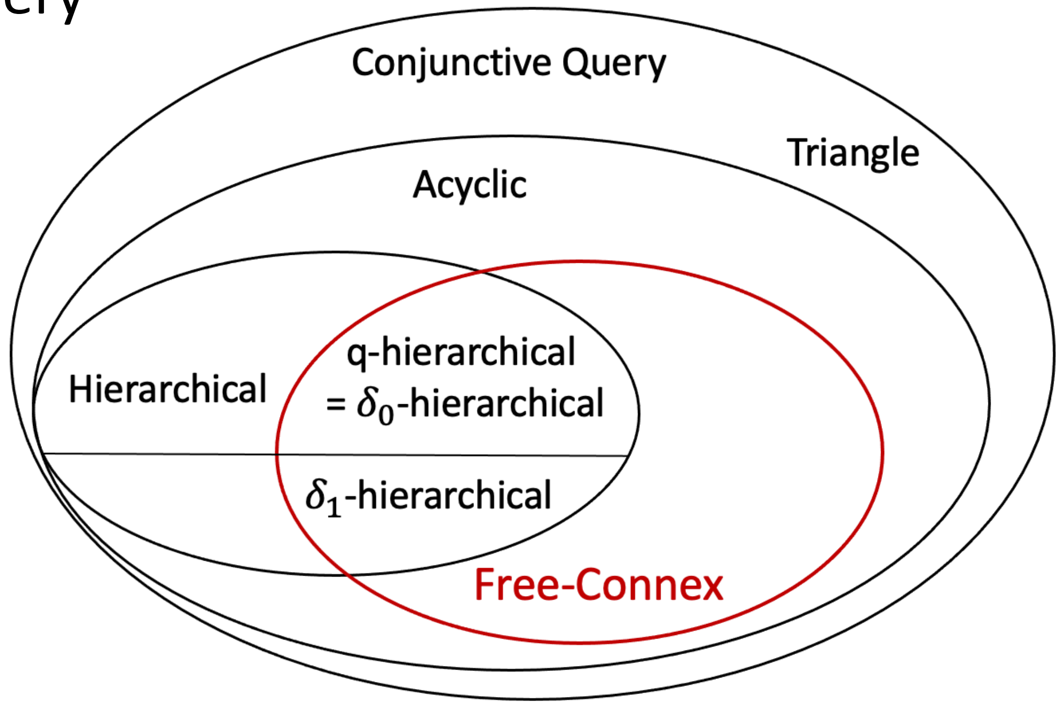
- The lower bound only holds for the worst-case update sequence

Outline

- Part I: Full Enumeration for Free-Connex Query

- Part II: Full Enumeration for Free-Connex Query with Aggregations

- Part III: Delta Enumeration for Free-Connex Query



- Answering Conjunctive Queries under Updates [BKS17]
- The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing under Updates [IUV17]
- General dynamic Yannakakis: conjunctive queries with theta joins under updates [IUVVW20]
- Change Propagation Without Joins [WHDY23]

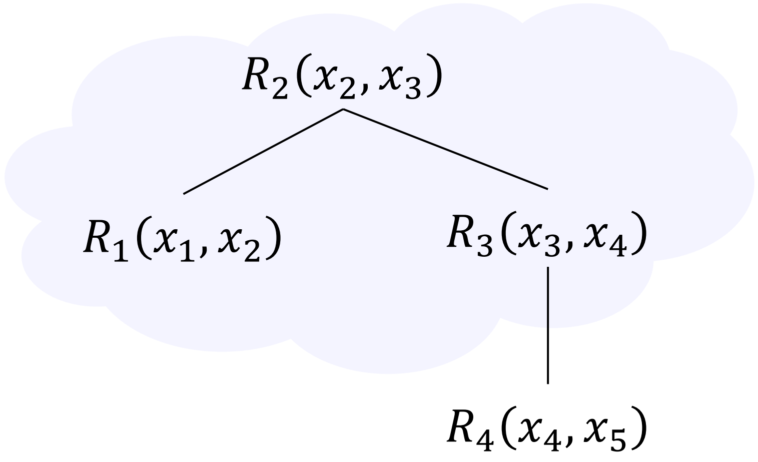
Conjunctive Queries (CQ)

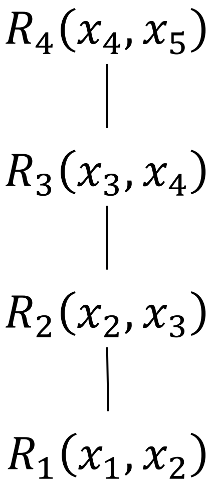
$$q = \pi_{\text{out}} R_1(e_1) \bowtie R_2(e_2) \bowtie \cdots \bowtie R_n(e_n)$$

- Relations: $R_1, R_2, \ldots, R_n$
- Attributes: $e_1 \cup e_2 \cup \cdots \cup e_n$
- Output attributes: $\text{out} \subseteq e_1 \cup e_2 \cup \cdots \cup e_n$
- Full Join: $\text{out} = e_1 \cup e_2 \cup \cdots \cup e_n$ (the projection "π_{out}" can be omitted)
- Boolean query: $\text{out} = \emptyset$
- Example:
  - $R_1(x_2, x_3) \bowtie R_2(x_1, x_3) \bowtie R_3(x_1, x_2)$
  - $\pi_{x_1,x_2,x_3,x_4} R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_3, x_4) \bowtie R_4(x_4, x_5)$
  - $\pi_{\emptyset} R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_3, x_4) \bowtie R_4(x_4, x_5)$

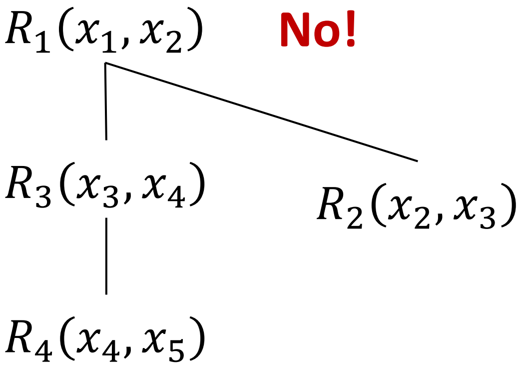
Join Tree for Free-Connex CQ

- A join tree T:
  - There is one-to-one correspondence between relations and nodes
  - For each attribute x, all nodes containing x forms a connected subtree
  - No non-output attributes appears above the topmost node of any output attribute





No!



No!

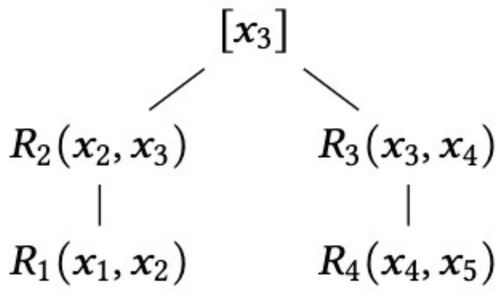
$$\pi_{x_1,x_2,x_3,x_4} R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_3, x_4) \bowtie R_4(x_4, x_5)$$

Generalized Join Tree for Free-Connex CQ

A generalized relation can be a proper subset of any original relation

- A generalized join tree T:
  - Each original relation corresponds to a node
  - For each attribute x, all nodes containing x forms a connected subtree
  - No non-output attributes appears above the topmost node of any output attribute
  - Generalized relations appear above of original relations
  - For every generalized relation e and its child e', $e \subseteq e'$.

- Height: max # original relations on any leaf-to-root path



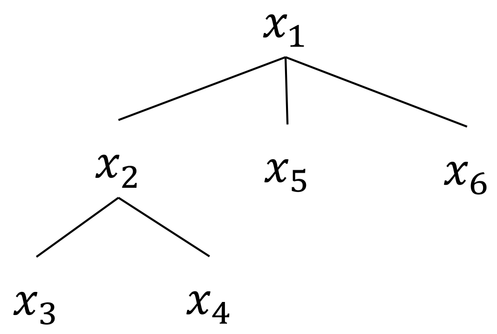
$$\pi_{x_1,x_2,x_3,x_4} R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_3, x_4) \bowtie R_4(x_4, x_5)$$

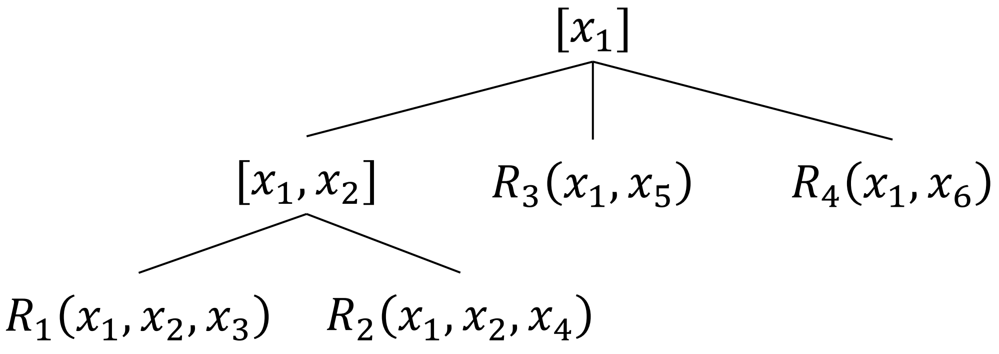
Q-hierarchical CQ

- For every pair of attributes x_1, x_2:
  - $E_{x_1} \subseteq E_{x_2}$, or $E_{x_2} \subseteq E_{x_1}$, or $E_{x_1} \cap E_{x_2} = \emptyset$
  - if $x_1 \in \mathrm{out}$ and $E_{x_1} \subsetneq E_{x_2}$, then $x_2 \in \mathrm{out}$

E_x is the set of relations containing attribute x

- A CQ q is q-hierarchical $\Longleftrightarrow$ it has a height-1 generalized join tree

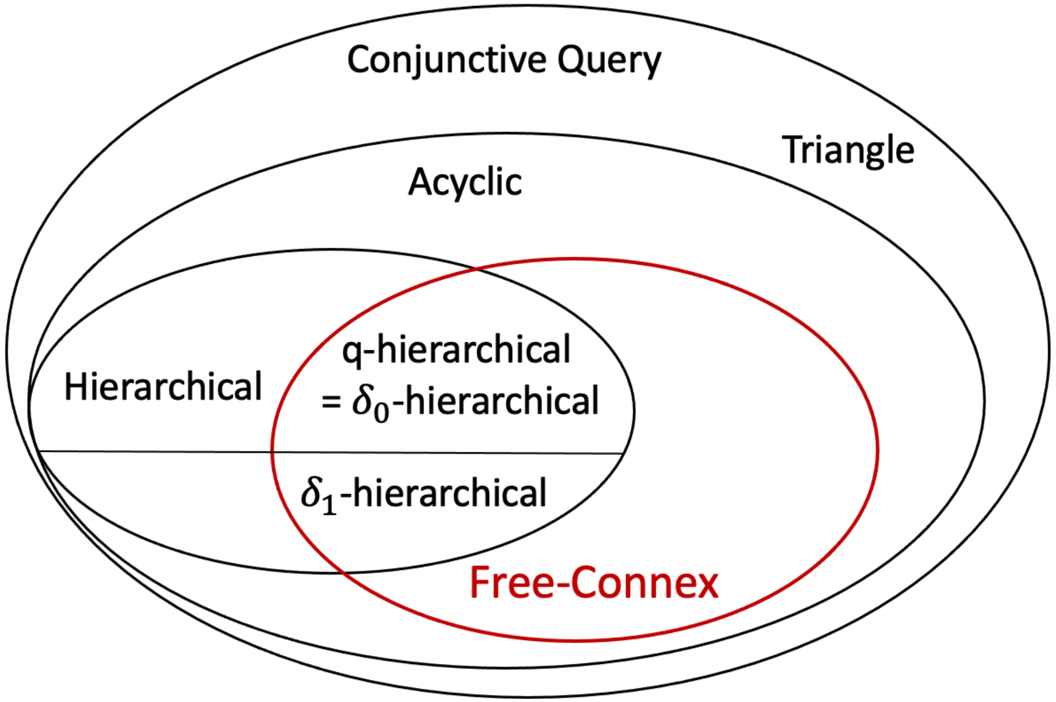




$$\pi_{x_1,x_2,x_3,x_4} R_1(x_1, x_2, x_3) \bowtie R_2(x_1, x_2, x_4) \bowtie R_3(x_1, x_5) \bowtie R_4(x_1, x_6)$$

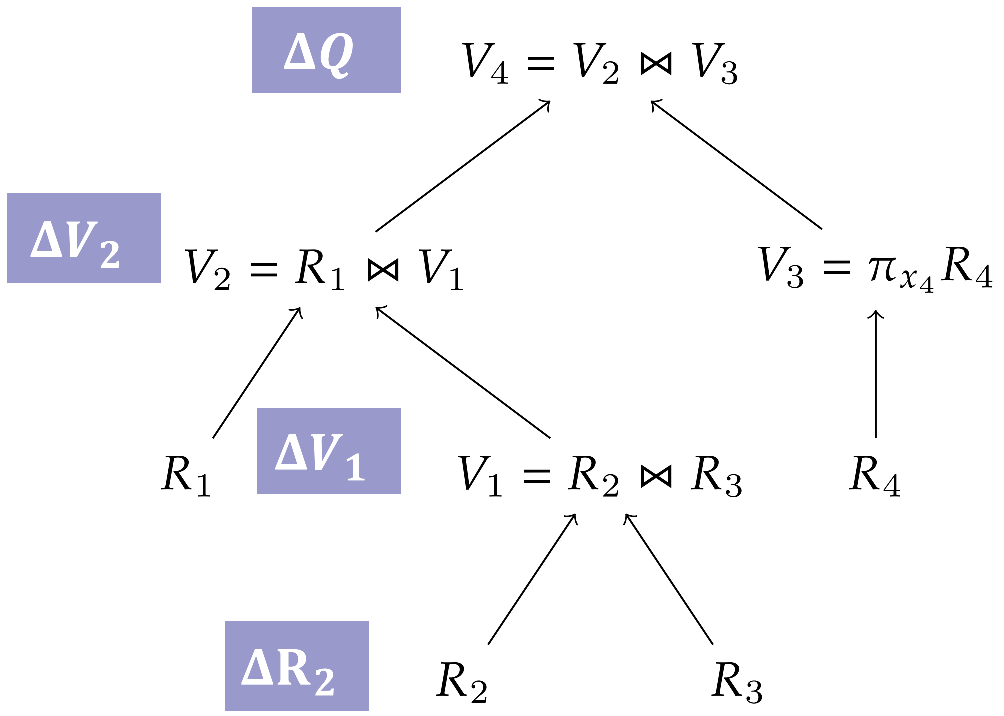
Outline

- Part I: Full Enumeration for Free-Connex CQ
- Part II: Full Enumeration for Free-Connex CQ with Aggregations
- Part III: Delta Enumeration for Free-Connex CQ



Change Propagation [RSS96][LSK01][CY12]

- Leaf Nodes: Original Relation
  Full and delta enumeration are trivial
- Internal Nodes: Operator
- Materialize the auxiliary views require super-linear space.
  - V_1 can be as large as $|R_2|\times|R_3|$
- The update cost can be super-linear.
  - ΔV_2 can be as large as $|R_2|\times|R_3|$
- All caused by the join operator.

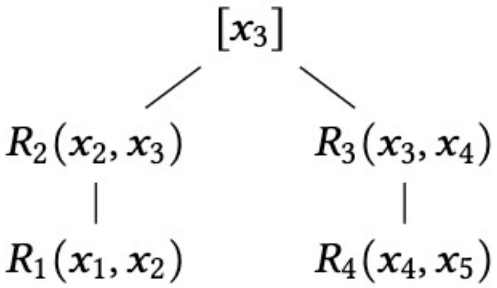


$$\pi_{x_1,x_2,x_3,x_4} R_1(x_1,x_2) \bowtie R_2(x_2,x_3) \bowtie R_3(x_3,x_4) \bowtie R_4(x_4,x_5)$$

Change Propagation Without Joins

- Basic idea: Replace each join operator with a semi-join followed by a projection.

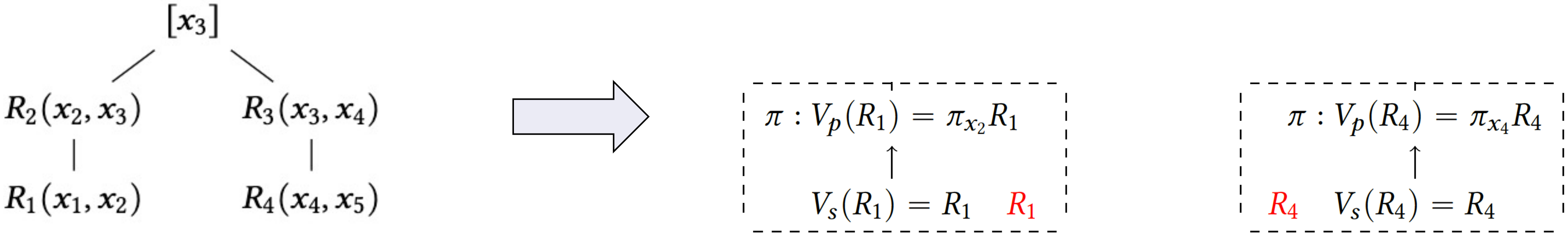
- Projection View $V_p(R_e)$
- Semi-join View $V_s(R_e)$



Change Propagation Without Joins

- Basic idea: Replace each join operator with a semi-join followed by a projection.

- Projection View $V_p(R_e)$
- Semi-join View $V_s(R_e)$

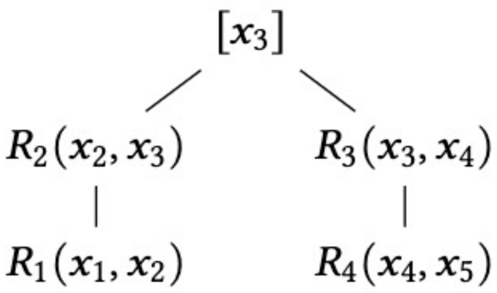


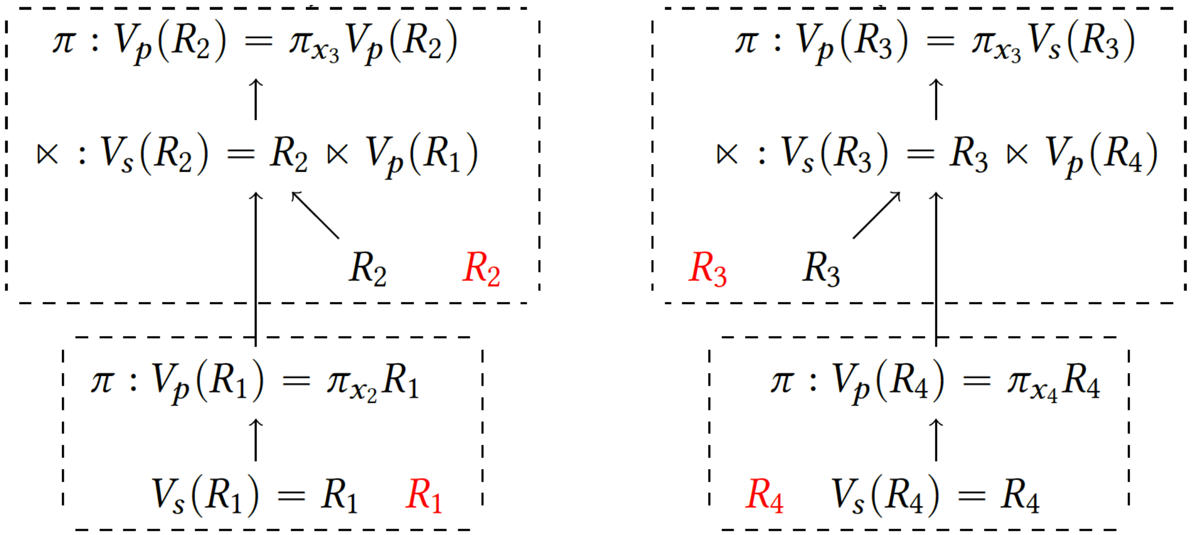
$$\pi_{x_1,x_2,x_3,x_4} R_1(x_1,x_2) \bowtie R_2(x_2,x_3) \bowtie R_3(x_3,x_4) \bowtie R_4(x_4,x_5)$$

Change Propagation Without Joins

- Basic idea: Replace each join operator with a semi-join followed by a projection.

- Projection View $V_p(R_e)$
- Semi-join View $V_s(R_e)$



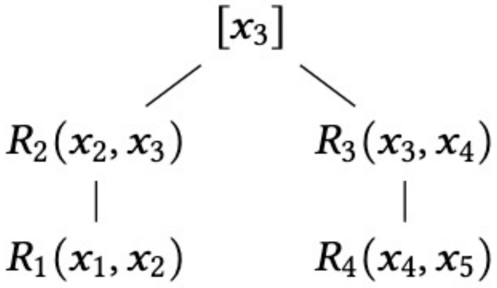


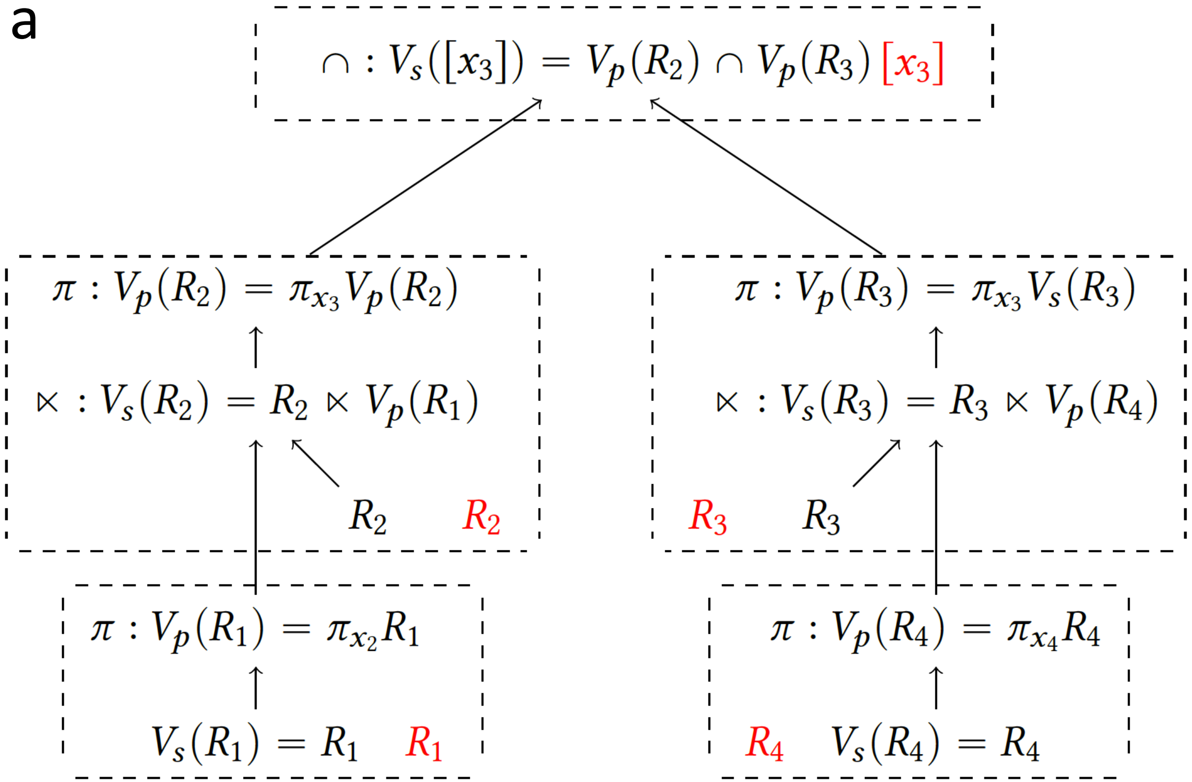
$$\pi_{x_1,x_2,x_3,x_4} R_1(x_1,x_2) \bowtie R_2(x_2,x_3) \bowtie R_3(x_3,x_4) \bowtie R_4(x_4,x_5)$$

Change Propagation Without Joins

- Basic idea: Replace each join operator with a semi-join followed by a projection.

- Projection View $V_p(R_e)$
- Semi-join View $V_s(R_e)$





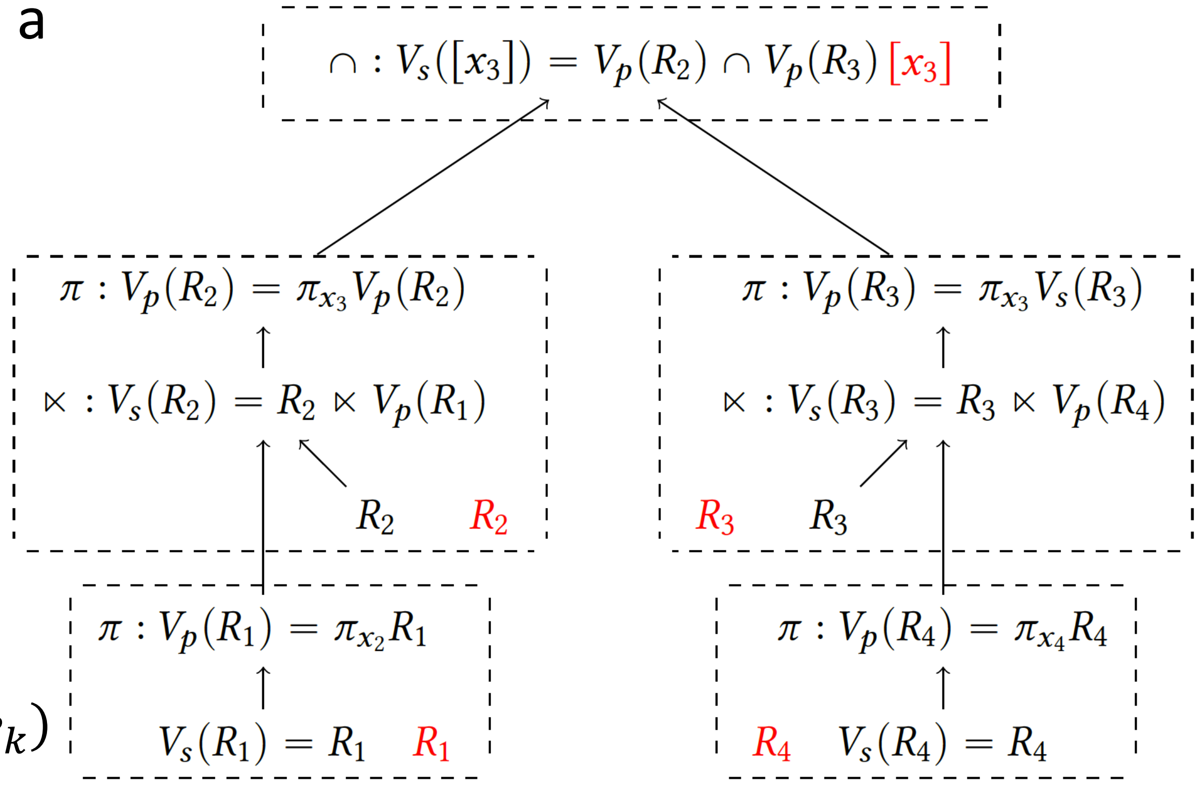
$$\pi_{x_1,x_2,x_3,x_4} R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_3, x_4) \bowtie R_4(x_4, x_5)$$

Change Propagation Without Joins

- Basic idea: Replace each join operator with a semi-join followed by a projection.

- Projection View $V_p(R_e) = \pi_{e \cap \text{par}(e)} V_s(R_e)$

- Semi-join View $V_s(R_e)$
  - Leaf node: $V_s(R_e) = R_e$
  - Internal node with children $e_1, e_2, \cdots, e_k$
    - $V_s(R_e) = R_e \ltimes V_p(e_1) \ltimes V_p(e_2) \ltimes \cdots \ltimes V_p(e_k)$
    - $V_s(R_e) = V_p(e_1) \cap V_p(e_2) \cap \cdots \cap V_p(e_k)$



$$\pi_{x_1,x_2,x_3,x_4} R_1(x_1,x_2) \bowtie R_2(x_2,x_3) \bowtie R_3(x_3,x_4) \bowtie R_4(x_4,x_5)$$

Semi-join under Updates

- The size of results will be bounded by $|R_1|$
  - Bounded memory cost for materialization
- Each update in R_2 can cause at most $O(|R_1|)$ changes in the result.
  - Bounded maintenance cost

$R_1(x_1, x_2)$

x_1	x_2
1	$2n$
2	$2n$
$\dots$	
$n-1$	$2n$
n	$2n$

$\ltimes$

$R_2(x_2, x_3)$

x_2	x_3
$2n$	$2n+1$
$2n$	$2n+2$
$\dots$	
$2n$	$3n-1$
$2n$	$3n$

$=$

x_1	x_2
1	$2n$
2	$2n$
$\dots$	
$n-1$	$2n$
n	$2n$

Projection under Updates

- The size of $\pi_x(R_1)$ cannot exceed N
  - Bounded memory cost for materialization
- Each update can cause at most 1 change in the result.
  - Constant update time guarantee (with derivation counting)

$R_1(x_1, x_2)$

x_1	x_2
1	$2n$
2	$2n$
$\dots$	
$n-1$	$2n$
n	$2n$

π_{x_2} $=$

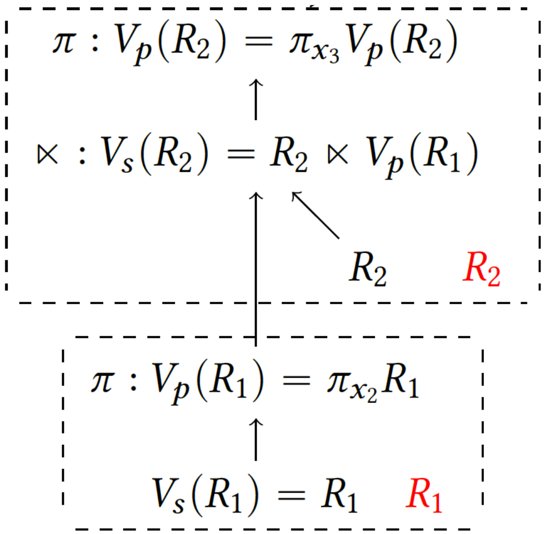
x_2
$2n$

View Maintenance

- Auxiliary counter for $t \in V_p(R_e)$:
  $c[t] = \left|\{t' \in V_s(R_e): \pi_{e \cap \mathrm{par}(e)} t' = t\}\right|$

- Auxiliary counter for $t \in V_s(R_e)$:
  - Internal node e with children $e_1, e_2, \cdots, e_k$
    $c[t] = \left|\{i \in [k]: c[\pi_{e \cap e_i} t] > 0\}\right|$

- From $R_e \to V_s(R_e)$: $O(1)$ time
- From $V_s(R_e) \to V_p(R_e)$: $O(1)$ time
- From $V_p(R_e) \to V_s(R_{\mathrm{par}(e)})$: $O(N)$ time



$V_s(R_2)$

x_2	x_3	$c[t]$
1	2	0
2	2	1
4	3	0
1	1	0
2	4	1
1	4	0

$V_p(R_1)$

x_2	$c[t]$
2	2
3	1

↑

$V_s(R_1)$

x_1	x_2
1	2
2	2
3	3

Running Example: initialization

R_1

x_1	x_2
1	2
2	2
3	3

R_2

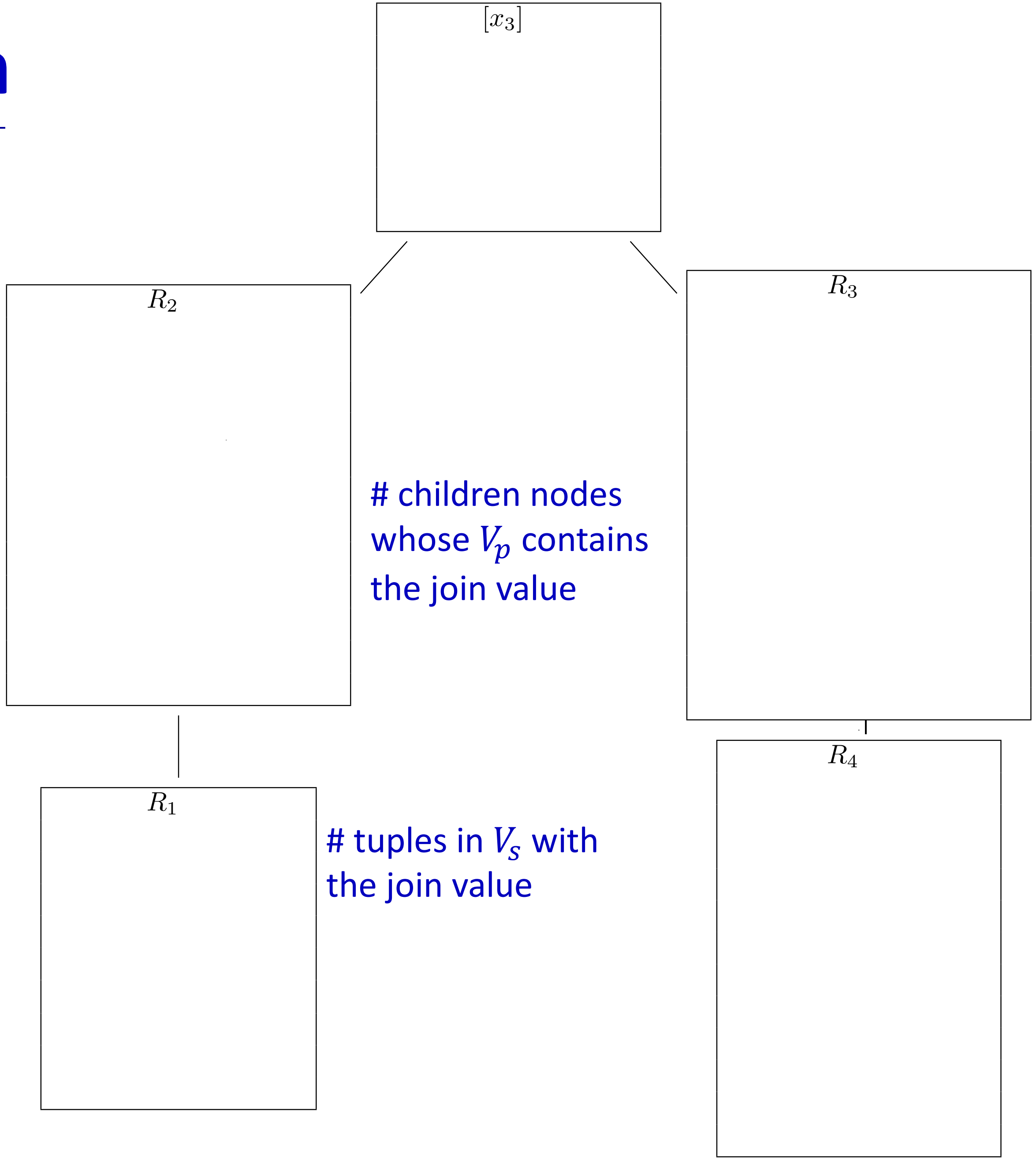
x_2	x_3
1	2
2	2
4	3
1	1
2	4
1	4

R_3

x_3	x_4
1	1
2	5
3	3
1	2
4	4

R_4

x_4	x_5
1	1
2	2
3	3
4	4



Tuples not “exist” in V_s:

Running Example: insert (1, 1) to R_1

1. insert (1,1) to $V_s(R_1)$
2. increase $c[1] \in V_p(R_1)$ by 1
3. As $1 \notin V_p(R_1)$, update every $c[(1,*)]$ in $V_s(R_2)$
4. update $V_p(R_2)$ correspondingly
5. As $1 \notin V_p(R_2)$, update $c[1]$ in $V_s([x_3])$

$[x_3]$

$V_s([x_3])$

x_3	$c[t]$
1*	2
2	1
3	1
4	2

R_2

$V_p(R_2)$

x_3	$c[t]$
4	2
2	2
1	1

$V_s(R_2)$

x_2	x_3	$c[t]$
1	2	1
2	2	1
4	3	0
1	1	1
2	4	1
1*	4*	1

R_3

$V_p(R_3)$

x_3	$c[t]$
1	2
3	1
4	1

$V_s(R_3)$

x_3	x_4	$c[t]$
1	1	1
2	5	0
3	3	1
1	2	1
4	4	1

R_1

$V_p(R_1)$

x_2	$c[t]$
2	2
3	1
1	1

$V_s(R_1)$

x_1	x_2
1	2
2	2
3	3
1	1

R_4

$V_p(R_4)$

x_4	$c[t]$
1	1
2	1
3	1
4	1

$V_s(R_4)$

x_4	x_5
1	1
2	2
3	3
4	4

Running Example: delete (1, 1) from R_4

No counter decreases to 0, hence stops!

$[x_3]$ — $V_s([x_3])$

x_3	$c[t]$
1	2
2	1
3	1
4	2

R_2

$V_p(R_2)$

x_3	$c[t]$
4	2
2	2
1	1

$V_s(R_2)$

x_2	x_3	$c[t]$
1	2	1
2	2	1
4	3	0
1	1	1
2	4	1
1	4	1

R_3

$V_p(R_3)$

x_3	$c[t]$
1	1
3	1
4	1

$V_s(R_3)$

x_3	x_4	$c[t]$
1*	1*	0
2	5	0
3	3	1
1	2	1
4	4	1

R_1

$V_p(R_1)$

x_2	$c[t]$
2	2
3	1
1	1

$V_s(R_1)$

x_1	x_2
1	2
2	2
3	3
1	1

R_4

$V_p(R_4)$

x_4	$c[t]$
~~1~~	~~1~~
2	1
3	1
4	1

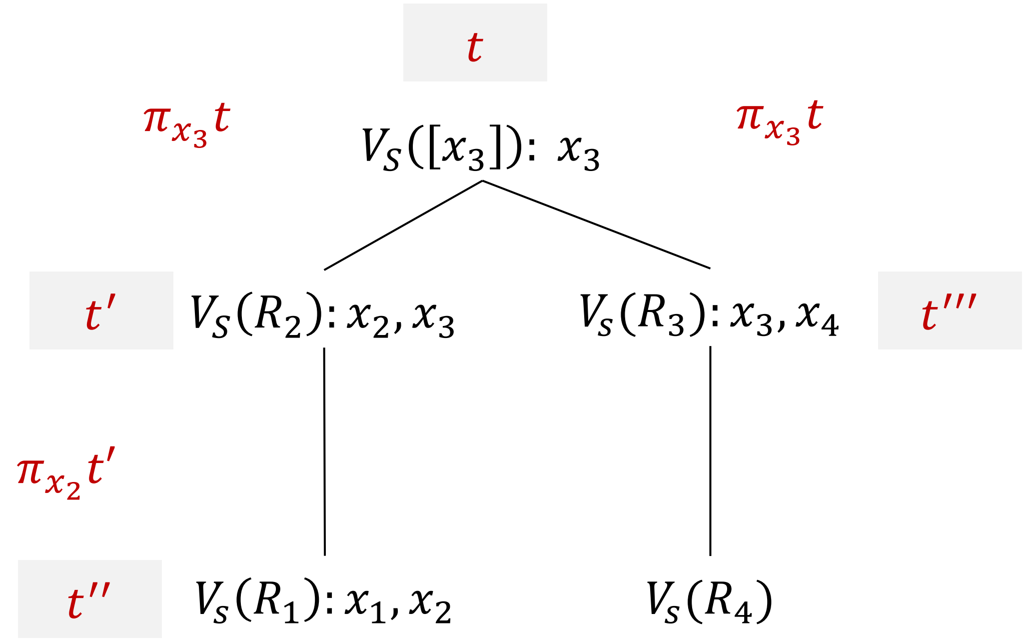
$V_s(R_4)$

x_4	x_5
~~1~~	~~1~~
2	2
3	3
4	4

1. delete (1,1) from $V_S(R_1)$
2. decrease $c[(1)] \in V_P(R_4)$ by 1
3. As $1 \notin V_p(R_4)$, decrease each $c[(*,1)] \in V_s(R_3)$ by 1
4. update $V_p(R_3)$ correspoindingly

Full Enumeration

- Lemma: $V_S(R_e) = \pi_e\left(\bowtie_{e' \in T_e} R_{e'}\right)$
  - T_e: the set of relations residing in the subtree of T rooted at node e
  - $V_S(R_r) = \pi_r q(D)$ for the root node r
- Lemma: $q(D) = \biguplus_{t \in V_S(R_r)} q(D \ltimes t)$
- Compute $q(D \ltimes t)$ by retrieving query results from $V_S(\cdot)$ in a top-down way



$\pi_{x_1,x_2,x_3,x_4} R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_3, x_4) \bowtie R_4(x_4, x_5)$

Running Example: Retrieve

$[x_3]$ — $V_s([x_3])$

x_3	$c[t]$
1	2
2	1
3	1
4	2

R_2 — $V_p(R_2)$

x_3	$c[t]$
4	2
2	2
1	1

$V_s(R_2)$

x_2	x_3	$c[t]$
1	2	1
2	2	1
4	3	0
1	1	1
2	4	1
1	4	1

R_3 — $V_p(R_3)$

x_3	$c[t]$
1	1
3	1
4	1

$V_s(R_3)$

x_3	x_4	$c[t]$
1	1	0
2	5	0
3	3	1
1	2	1
4	4	1

R_1 — $V_p(R_1)$

x_2	$c[t]$
2	2
3	1
1	2

$V_s(R_1)$

x_1	x_2
1	2
2	2
3	3
1	1
4*	1*

R_4 — $V_p(R_4)$

x_4	$c[t]$
2	1
3	1
4	1

$V_s(R_4)$

x_4	x_5
2	2
3	3
4	4

1

$\bowtie V_s(R_2)$: $(1) \rightarrow (1, 1)$

2

$\bowtie V_s(R_1)$: $(1, 1) \rightarrow (1, 1, 1)$

$(1, 1) \rightarrow (4, 1, 1)$

3

$\bowtie V_s(R_3)$:

$(1, 1, 1) \rightarrow (1, 1, 1, 2)$

$(4, 1, 1) \rightarrow (4, 1, 1, 2)$

Enclosureness [SIGMOD'20]

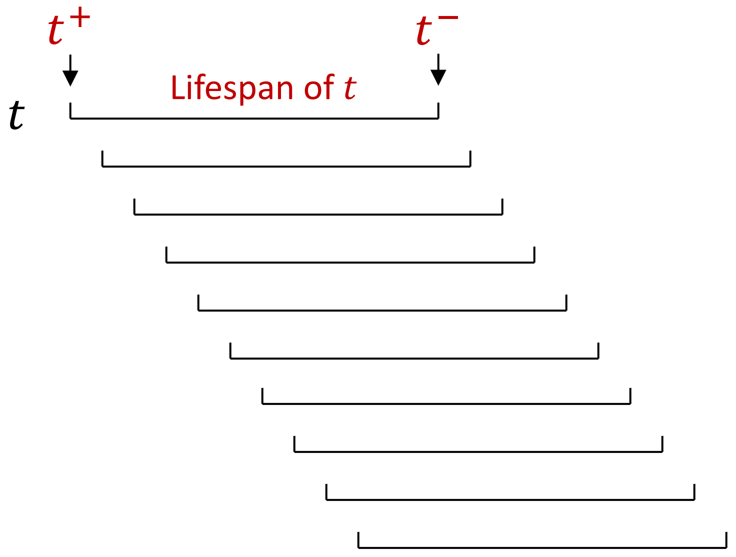
Given an update sequence S:

- A tuple t has a lifespan $[t^+, t^-]$
- Enclosureness of t:

  $\lambda_S(t)$ =max # disjoint lifespans contained in $[t^+, t^-]$

- Enclosureness of S: $\lambda_S = \max\left(1, \sum_t \frac{\lambda_S(t)}{|S|}\right)$

- Foreign-key acyclic query can be updated in $O(\lambda_S)$ time



FIFO sequence with $\lambda_S = 1$

Is this notion of Enclosureness Good?

- Consider $q = R_1(x_1) \bowtie R_2(x_1, x_2) \bowtie R_3(x_2, x_3) \bowtie R_4(x_3, x_4) \bowtie R_5(x_4)$
  - $\Omega(\sqrt{N})$ update time over FIFO sequences, assuming the OuMv conjecture.

OuMv Conjecture [STOC'15]

For any $\gamma > 0$, no algorithm can solve the following problem in $O(n^{3-\gamma})$ time,

Input: An $n \times n$ Boolean matrix M and n pairs $(u_1, v_1), \cdots, (u_n, v_n)$ of Boolean column-vectors of size n arriving one after the other.

Goal: After seeing each pair (u_r, v_r), output $u_r^T M v_r$ before seeing (u_{r+1}, v_{r+1})

Join-tree-based Enclosureness

- Given an update sequence S and a generalized join tree T
- A tuple $t \in R_e$ has two effective lifespans under T:
  - D_e: the set of tuples from any descendant node of e
  - $\left[t^+, \ \min\left(t^-, \min_{t_1 \in D_e : t_1^- > t^+} t_1^-\right)\right]$
  - $\left[\max\left(t^+, \max_{t_2 \in D_e : t_2^+ < t^-} t_2^+\right), t^-\right]$
- Enclosureness of tuple $t \in R_e$ under T:

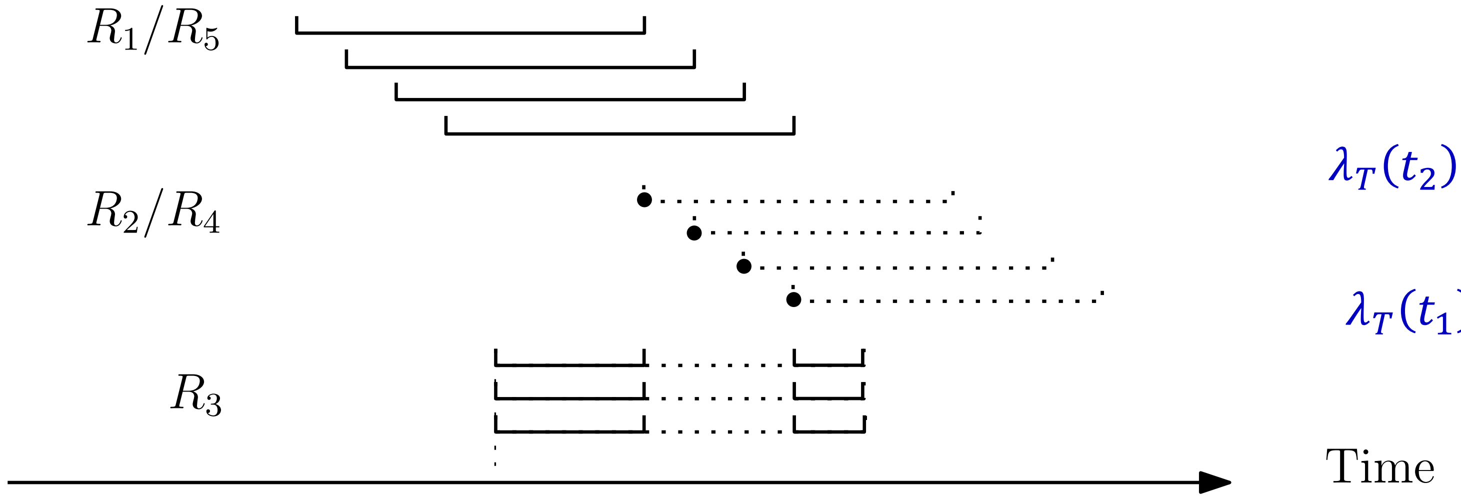
whose corresponding tuples are from the descendants of e in T

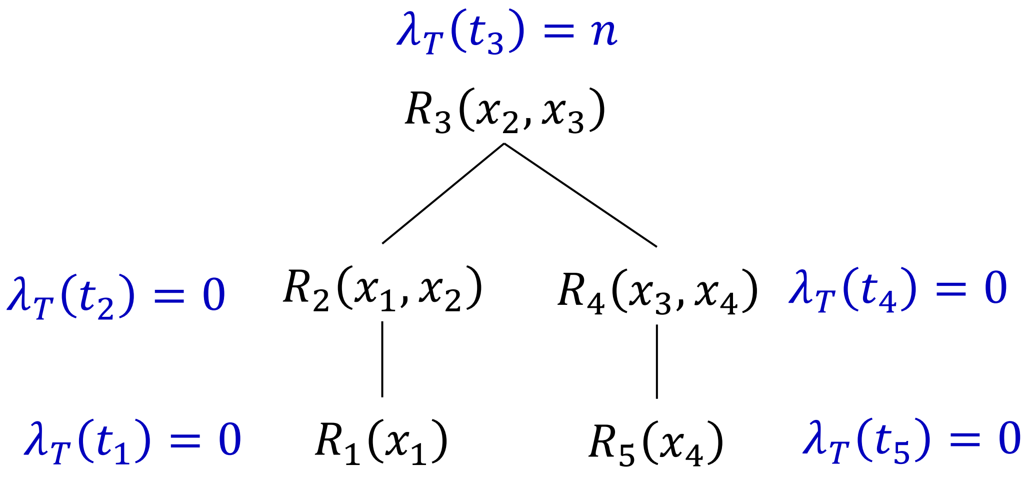
$$\lambda_{S,T}(t) = \text{max \# disjoint effective lifespans that are contained in } [t^+, t^-]$$

- Enclosureness $\lambda_{S,T}$ of S under T: $\lambda_{S,T} = \max\left(1, \sum_t \frac{\lambda_{S,T}(t)}{|S|}\right)$

Join-tree-based Enclosureness

- Revisit $q = R_1(x_1) \bowtie R_2(x_1, x_2) \bowtie R_3(x_2, x_3) \bowtie R_4(x_3, x_4) \bowtie R_5(x_4)$



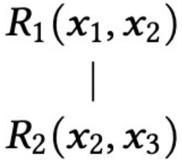


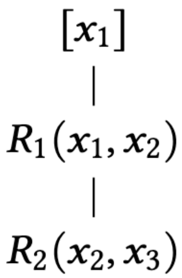
$$\lambda_{S,T} = \frac{n \cdot n^2}{n^2} = n \geq \sqrt{N}$$

Is Join-tree-based Enclosureness Good?

- For any free-connex CQ, the data structure built on T can be updated in $O(\lambda_{S,T})$ amortized time over update sequence S with enclosureness $\lambda_{S,T}$

- Consider $q = \pi_{x_1} R_1(x_1, x_2) \bowtie R_2(x_2)$
  - $\Omega(\lambda)$ update time over update sequence with enclosureness λ, assuming the OMv conjecture





OMv Conjecture [STOC'15]

For any $\gamma > 0$, no algorithm can solve the following problem in $O(n^{3-\gamma})$ time:

Input: An $n \times n$ Boolean matrix M and n Boolean column-vectors $v_1, v_2, \cdots, v_n$ of size n arriving one after the other.

Goal: After seeing each v_r, output Mv_r before seeing v_{r+1}

Proof Idea: $q = \pi_{x_1} R_1(x_1, x_2) \bowtie R_2(x_2)$

$\lambda(t_1) = n$

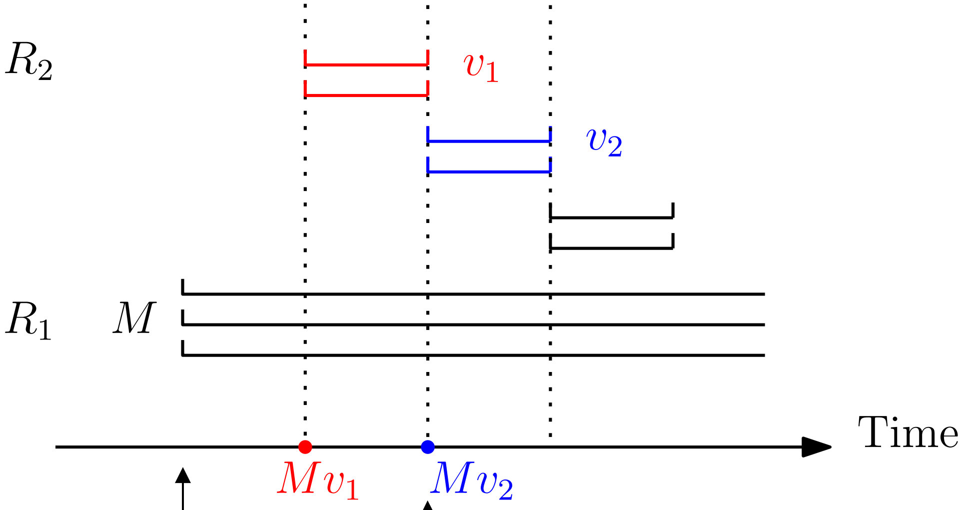
$R_1(x_1, x_2)$

|

$R_2(x_2, x_3)$

$\lambda(t_2) = 0$

$$\lambda \le \frac{n^2 \cdot n}{2n^2} = n$$



amortized update time ↓ delay ↓

$$\lambda^{1-\gamma} \cdot n^2 + \lambda^{1-\gamma} \cdot n^2 = O(n^{3-\gamma})$$

↑ # updates ↑ # query answers

For $(i, j) \in [n] \times [n]$, insert (i, j) into R_1 if $M[i, j] = 1$

For $j \in [n]$, insert (j) into R_2 if $v_1[j] = 1$;

$i \in q \Leftrightarrow (Mv_1)[i] = 1$

Delete (1) from R_1 and R_5

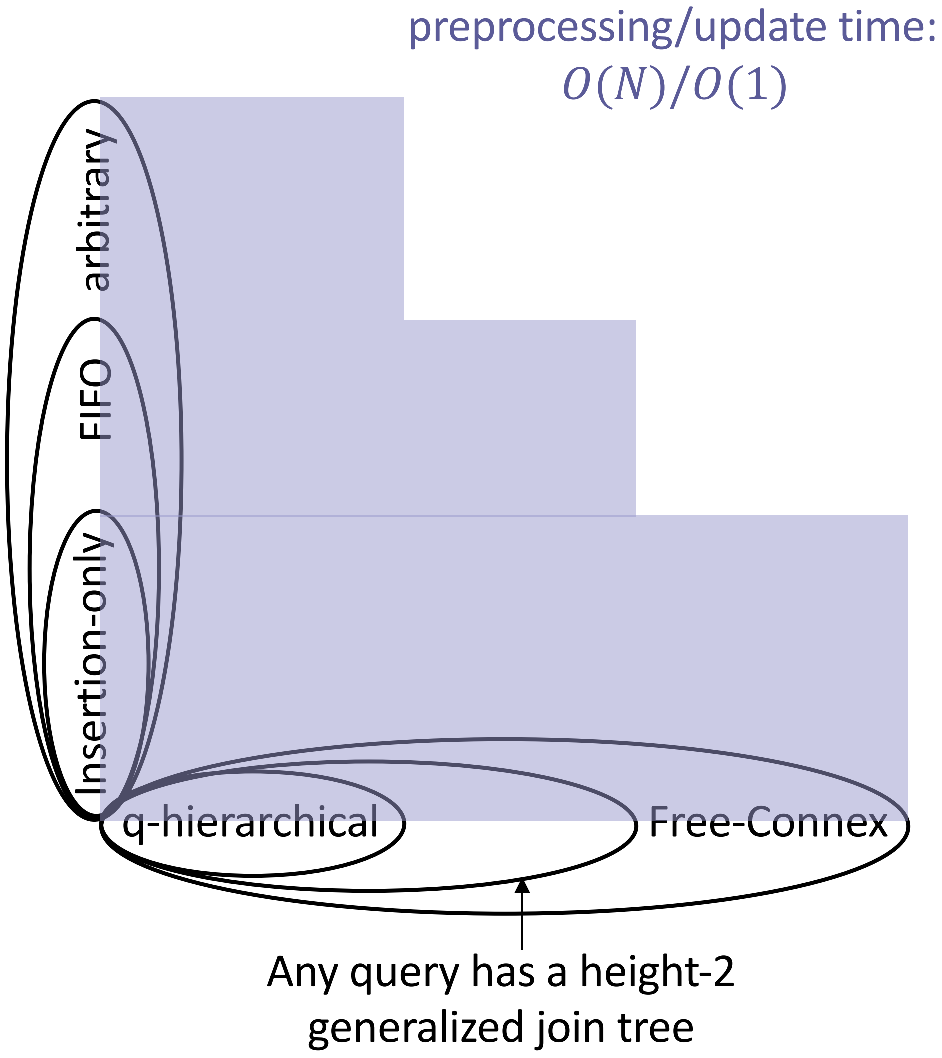
For $j \in [n]$, insert (j) into R_2 if $v_2[j] = 1$;

$i \in q \Leftrightarrow (Mv_2)[i] = 1$

Implications

- A data structure can be built in $O(N)$ time and updated in $O(1)$ amortized time while supporting $O(1)$-delay enumeration
  - q has a height-1 generalized join tree T
  - S is FIFO and q has a height-2 generalized join tree T since $\lambda_{S,T} = 1$
  - S is insertion-only and q is free-connex since $\lambda_{S,T} = 1$ for any T

- A nice structural characterization of CQs with height-2 generalized join tree?
- Some guidance for practical update sequences



Mixed Update Sequence?

Consider $q = R_1(x_1) \bowtie R_2(x_1, x_2) \bowtie R_3(x_2)$

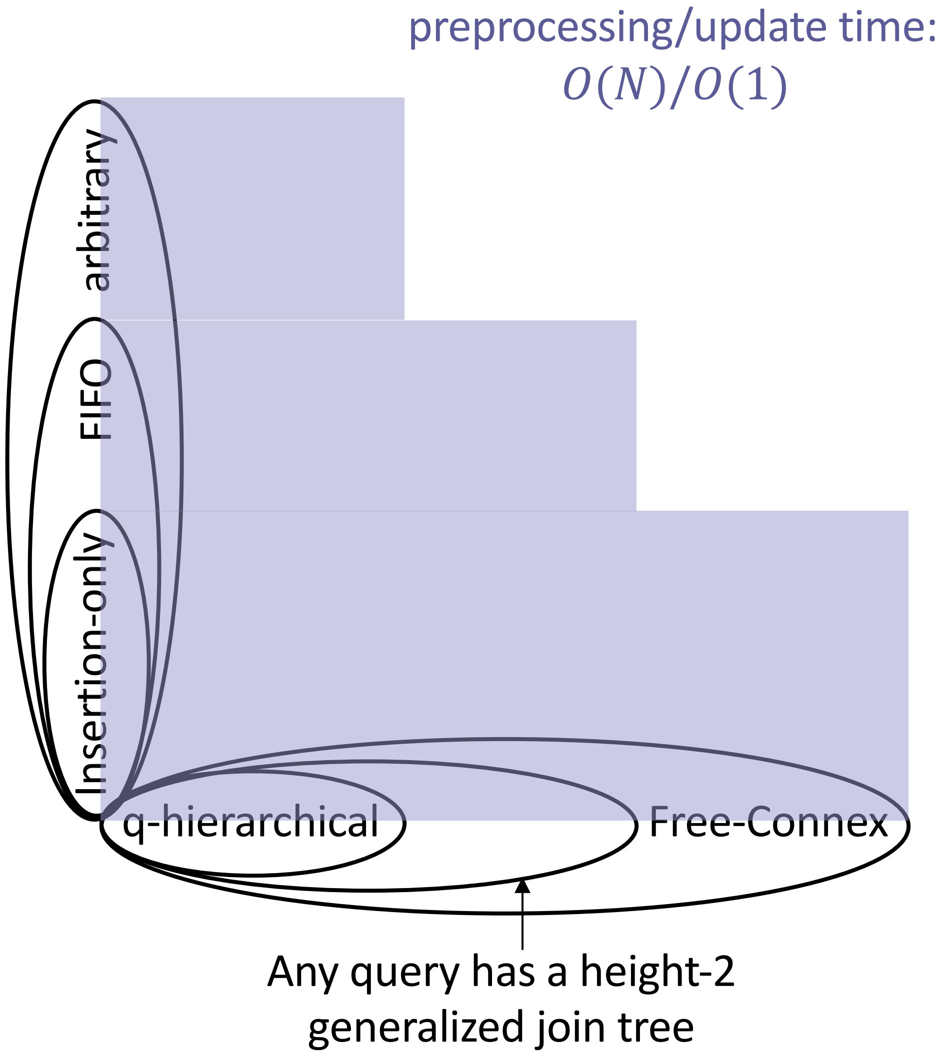
- If updates on R_1, R_2, R_3 are arbitrary?
- If updates on R_1, R_2, R_3 are all FIFO?
- If updates on R_1, R_2, R_3 are all insertion-only?
- If updates on R_1, R_3 are arbitrary but on R_2 are insertion-only?
- If updates on R_1, R_3 are insertion-only but on R_2 are arbitrary?
- If updates on R_1 are FIFO, on R_2 are arbitrary and on R_3 are insertion-only?
- ……

Can we have a more fine-grained analysis of update sequences?

Lower Bounds

- $\Omega(1)$ update time for non-free-connex CQ over insertion-only update sequence, assuming the BMM, triangle and hyper-clique conjectures.

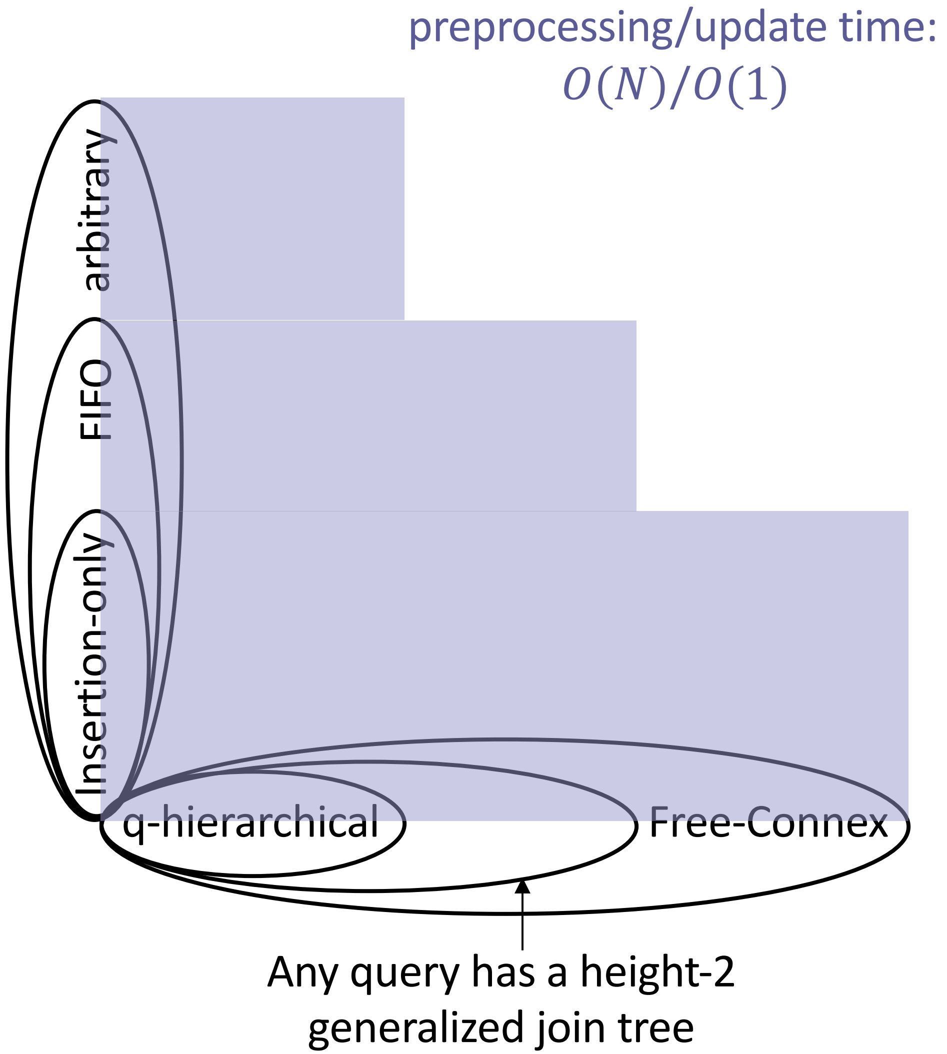
- $\Omega(\sqrt{N})$ update time for non-q-hierarchical CQ over arbitrary update sequences, assuming the OMv and OuMv conjectures.



CQs Without a Height-2 Generalized Join Tree

- Consider $q = R_1(x_1) \bowtie R_2(x_1, x_2) \bowtie R_3(x_2, x_3) \bowtie R_4(x_3, x_4) \bowtie R_5(x_4)$ or its Boolean version
  - $\Omega(\sqrt{N})$ update time over FIFO sequences, assuming OuMv conjecture.

- Consider $q = \pi_{x_1} R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_3)$
  - $\Omega(\sqrt{N})$ update time over FIFO sequences, assuming OMv conjecture.



Outline

- Part I: Full Enumeration for Free-Connex Query
- Part II: Full Enumeration for Free-Connex Query with Aggregations
- Part III: Delta Enumeration for Free-Connex Query

Annotated Relations

- Annotated Relations are functions mapping tuples to elements from a ring (here, $\mathbb{Z}$)

$R_1(x_1, x_2)$

x_1	x_2	w
a_1	b_1	2
a_2	b_1	3

$R_2(x_2, x_3)$

x_2	x_3	w
b_1	c_1	2
b_1	C_2	1

$R_3(x_1, x_3)$

x_1	x_3	w
a_1	c_1	1
a_1	c_2	3
a_2	c_2	3

$R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_1, x_3)$

x_1	x_2	x_3	w
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

- Annotation of a join result $t' \in \bowtie_e R_e$:

$$w(t') = \prod_e w(\pi_e t')$$

- Annotation of a query result $t \in q(D)$:

$$w(t) = \sum_{t' \in \bowtie_e R_e : \pi_{\text{out}} t' = t} w(t')$$

$\pi_\emptyset R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_1, x_3)$

$\emptyset$	w
()	$4 + 6 + 9 = 19$

Annotated Relations

- Annotated Relations are functions mapping tuples to elements from a ring (here, Z)
- An update maps a tuple to a non-zero value (+ for insertions, - for deletions)

$R_1(x_1, x_2)$

x_1	x_2	w
a_1	b_1	2
a_2	b_1	3

$R_2(x_2, x_3)$

x_2	x_3	w
b_1	c_1	2
b_1	C_2	1

$R_3(x_1, x_3)$

x_1	x_3	w
a_1	c_1	1
a_1	c_2	3
a_2	c_2	3

$R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_1, x_3)$

x_1	x_2	x_3	w
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

$\delta R_1 = \{(a_2, b_1) \rightarrow -2\}$

x_1	x_2	w
a_2	b_1	-2

$\pi_\emptyset R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_1, x_3)$

$\emptyset$	w
()	$4 + 6 + 9 = 19$

Annotated Relations

- Annotated Relations are functions mapping tuples to elements from a ring (here, $\mathbb{Z}$)
- An update maps a tuple to a non-zero value (+ for insertions, - for deletions)

$R_1(x_1, x_2)$

x_1	x_2	w
a_1	b_1	2
a_2	b_1	1

$R_2(x_2, x_3)$

x_2	x_3	w
b_1	c_1	2
b_1	C_2	1

$R_3(x_1, x_3)$

x_1	x_3	w
a_1	c_1	1
a_1	c_2	3
a_2	c_2	3

$R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_1, x_3)$

x_1	x_2	x_3	w
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$1 \cdot 1 \cdot 3 = 3$

$\delta R_1 = \{(a_2, b_1) \rightarrow -2\}$

x_1	x_2	w
a_2	b_1	-2

$\pi_\emptyset R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_1, x_3)$

$\emptyset$	w
()	$4 + 6 + 3 = 13$

Dynamic Yannakakis Algorithm

- Enrich semi-join and projection with annotation information!
- But this is essentially the join!

$R_1(x_1, x_2)$

x_1	x_2	w
a_1	b_1	2
a_2	b_1	3

$R_2(x_2, x_3)$

x_2	x_3	w
b_1	c_1	2
b_1	C_2	1

$R_1 \ltimes R_2 = \pi_{x_1,x_2} (R_1 \bowtie R_2)$

x_1	x_2	w
a_1	b_1	$2 \cdot 2 + 2 \cdot 1 = 6$
a_2	b_1	$3 \cdot 2 + 3 \cdot 1 = 9$

$\pi_{x_1} R_1$

x_1	w
a_1	2
a_2	3

$\pi_{x_2} R_2$

x_2	w
b_1	$2 + 1 = 3$

$(\pi_{x_2} R_1) \cap (\pi_{x_2} R_2) = (\pi_{x_2} R_1) \bowtie (\pi_{x_2} R_2)$

x_2	w
b_1	$(2 + 3) \cdot (2 + 1) = 15$

Dynamic Yannakakis Algorithm

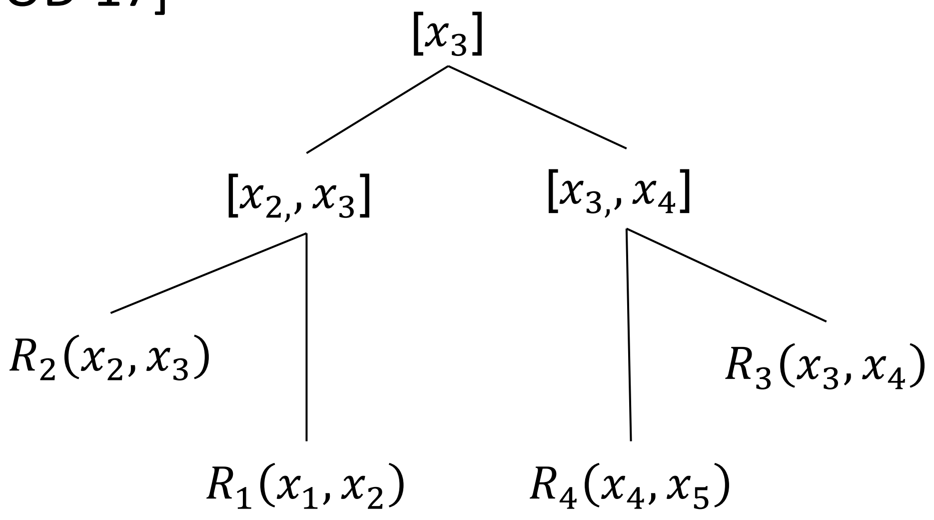
- A broader notion of generalized join tree [IUV, SIGMOD'17]
- Auxiliary counter for $t \in V_p(R_e)$:

$$c[t] = \sum_{t' \in V_s(R_e) : \pi_{e \cap \mathrm{par}(e)} t' = t} c[t']$$

- Auxiliary counter for $t \in V_s(R_e)$:
  - Leaf node : $c[t] = w(t)$
  - Internal node with children $e_1, e_2, \cdots, e_k$:

$$c[t] = \prod_{i \in [k]} c\left[\pi_{e \cap e_i} t\right]$$

- Update time is $O(N)$



Full Enumeration

- Full enumeration is almost the same
- When a query result is enumerated, compute its annotation on the fly
  - q is a full join:

$$w[t] = \prod_{e} w[\pi_e t]$$

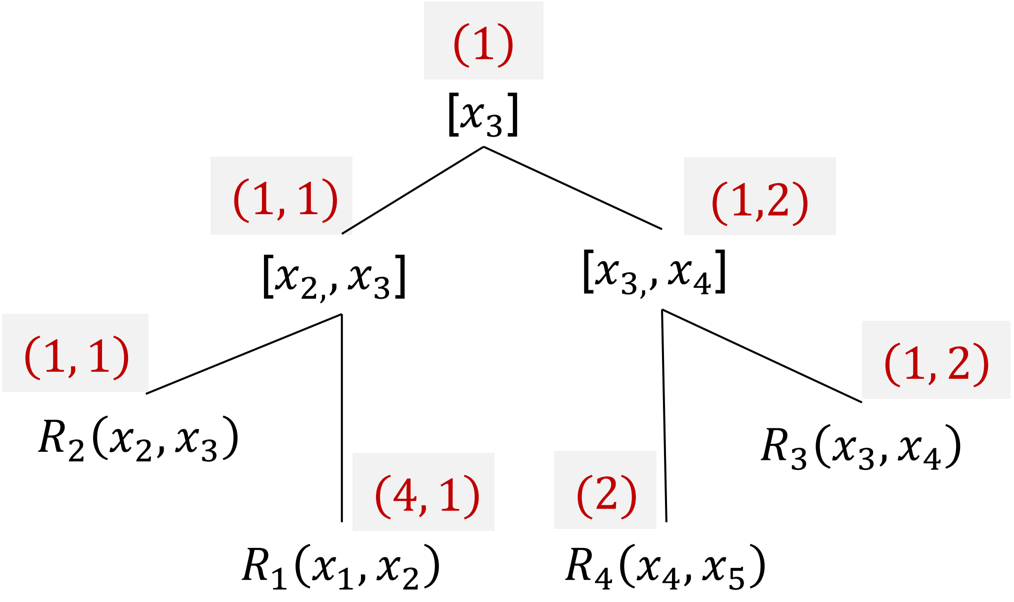
  - q is not a full join ($r \subseteq \text{out}$):

$$w[t] = \prod_{e:e\subseteq \text{out}} w[\pi_e t] \cdot \prod_{e:e-\text{out}\neq\emptyset,\text{par}(e)\subseteq\text{out}} c[\pi_{e\cap\text{out}} t]$$

$\pi_e t \in R_e$ for some original relation R_e

The "boundary" of the upper subtree whose nodes have all full output attributes

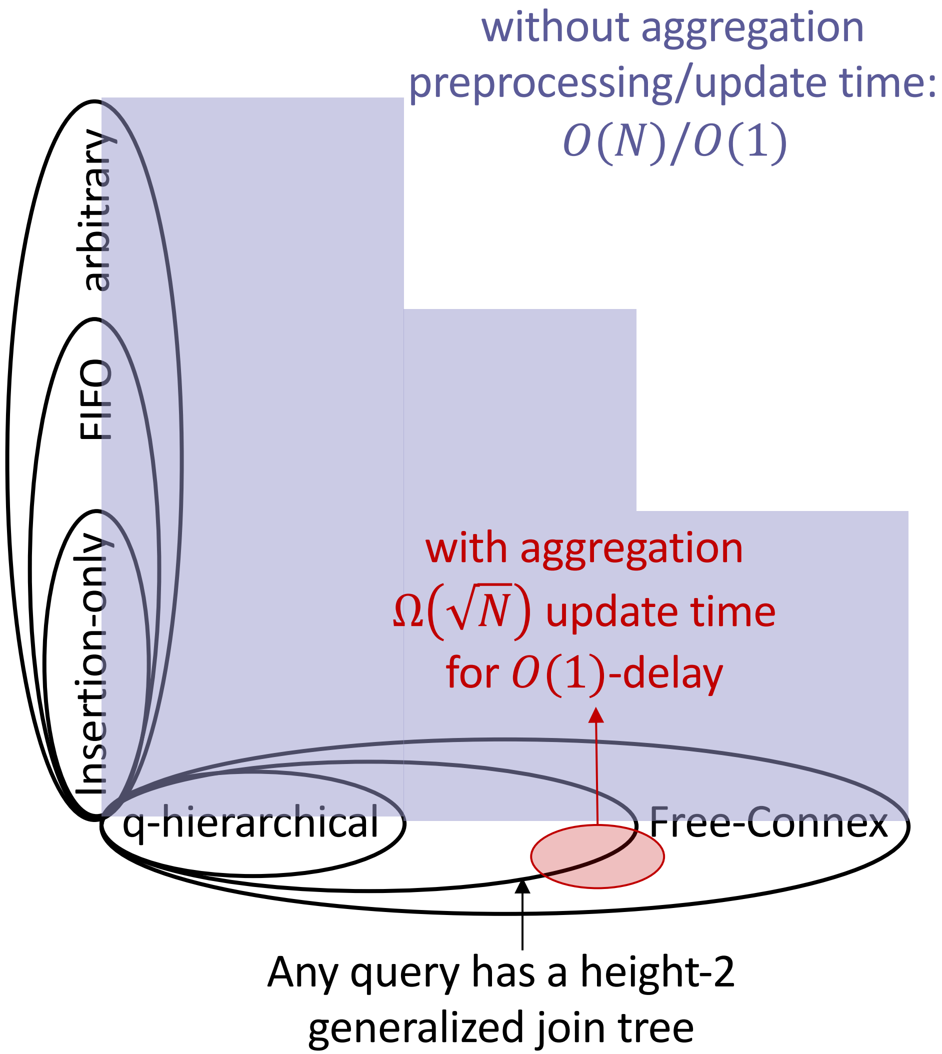
$\pi_{e\cap\text{out}} t \in V_p[\cdot]$



$$c[4,1,1,2] = w[4,1] \cdot w[1,1] \cdot w[1,2] \cdot c[2]$$

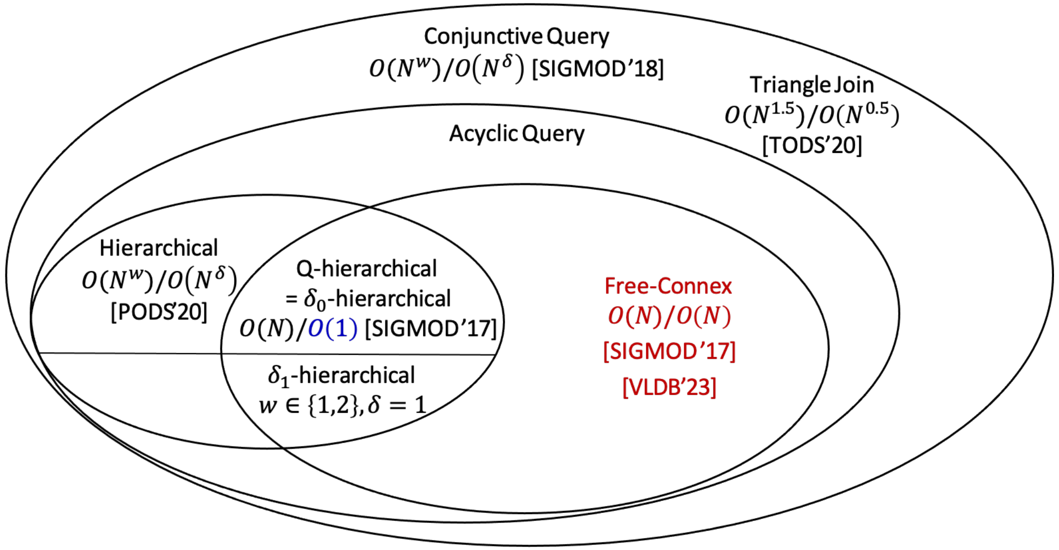
How Aggregate Increases Hardness

- Consider $q = \pi_{x_1} R_1(x_1, x_2) \bowtie R_2(x_2)$
  - $\Omega(\sqrt{N})$ update time over insertion-only update sequences, assuming the OMv conjecture.
- Consider $q = \pi_{\emptyset} R_1(x_1, x_2) \bowtie R_2(x_1) \bowtie R_3(x_2)$
  - $\Omega(\sqrt{N})$ update time over insertion-only update sequences, assuming the OuMv conjecture.



Outline

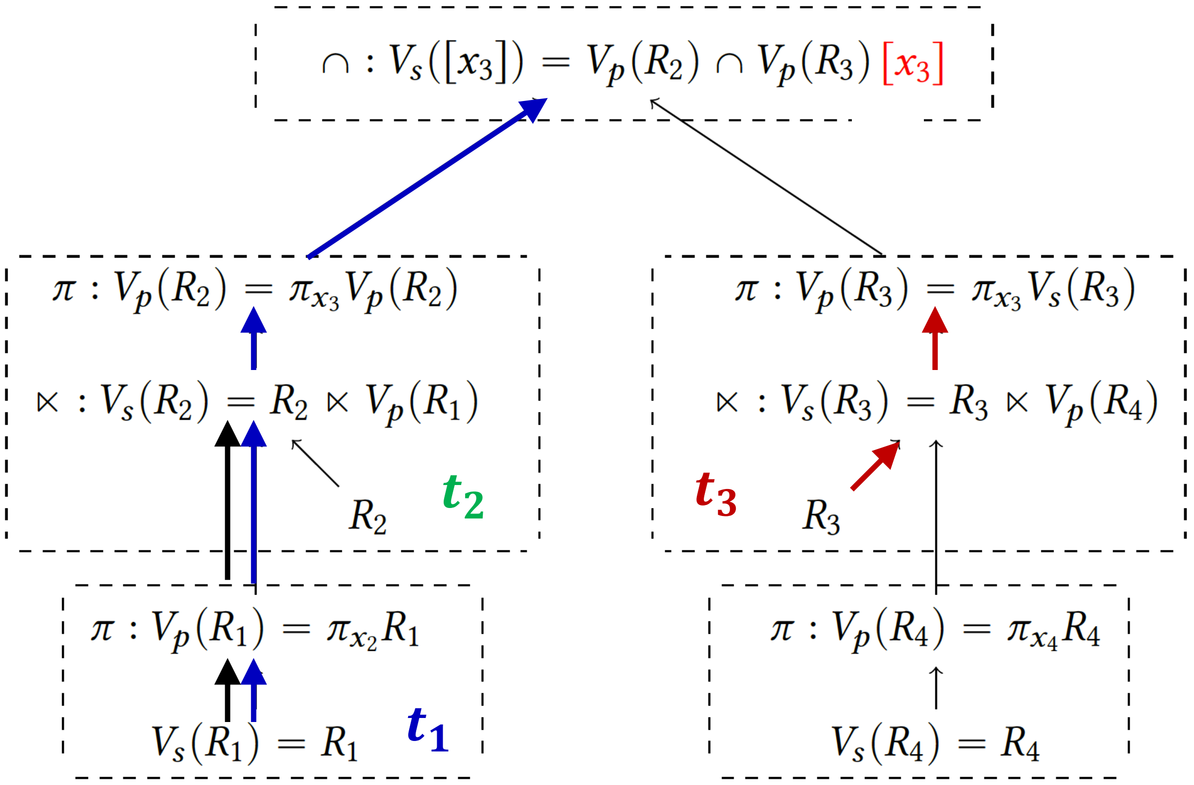
- Part I: Full Enumeration for Free-Connex Query
- Part II: Full Enumeration for Free-Connex Query with Aggregations
- Part III: Delta Enumeration for Free-Connex Query



Delta Enumeration without Aggregation

- Propagation paths:
  - $t_1 \to V_p(R_1) \to V_s(R_2) \to V_p(R_2) \to V_s([x_3])$
  - $t_1 \to V_p(R_1) \to V_s(R_1)$
  - t_2
  - $t_3 \to V_s(R_3) \to V_p(R_3)$

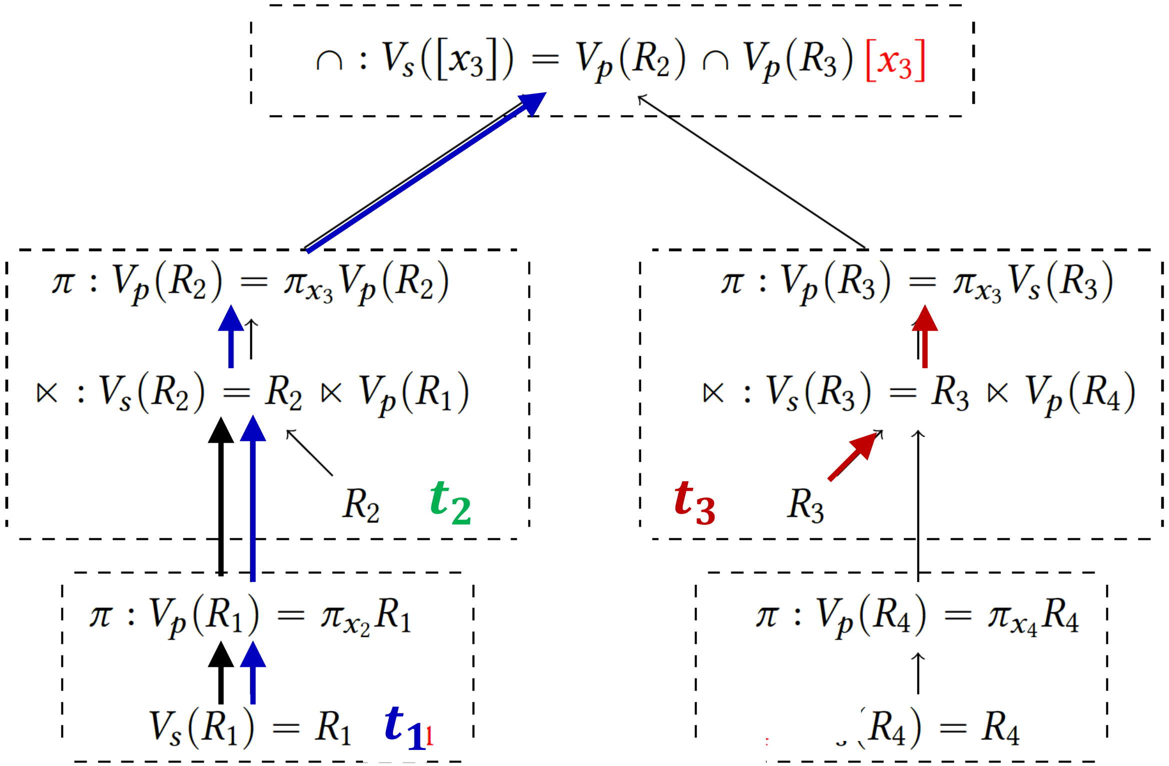
- If a propagation path can induce some delta query results, its ending tuple must be in some $\Delta V_s(\cdot)$



Live View

- $V_l(R_e) = \pi_{e \cap \text{out}} q(D)$
  - $t \in \pi_{e \cap \text{out}} V_s(R_e)$
  - $t \bowtie V_l\left(R_{\text{par}(e)}\right) \neq \emptyset$

- Maintain $V_l(R_e)$ during enumeration

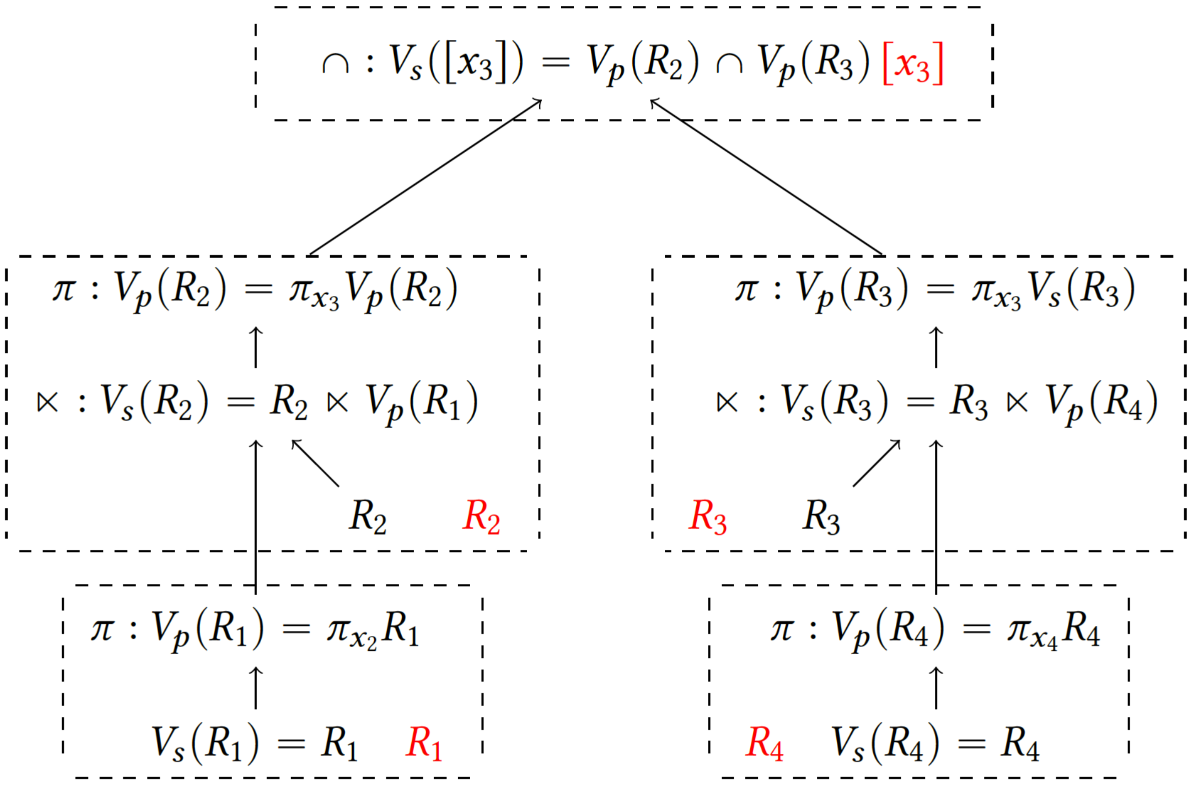


Witness Tuple

- t' is a witness of t if it is the ending tuple of a propagation path starting from t, and
  - $t' \in \Delta V_s(R_r, t)$, or
  - $t' \in \pi_{e \cap \text{out}} \Delta V_s(R_e, t)$ and $t' \bowtie V_l(R_{\text{par}(e)}) \neq \emptyset$ for some non-root e with $e \cap \text{out} \neq \emptyset$

$\Downarrow$

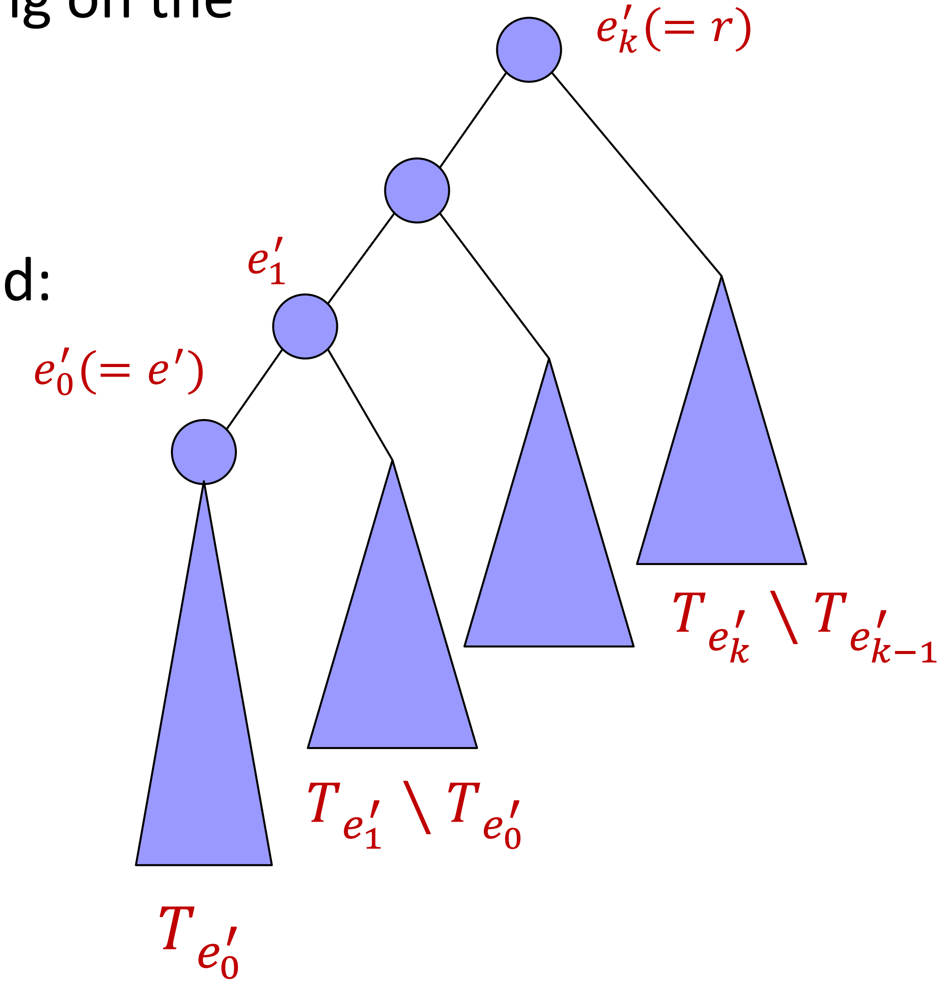
This path stops at t' since $\pi_{e \cap \text{par}(e)} t' \in V_p(R_e)$.



- Lemma: $\Delta q(D, t) = \biguplus_{t' \text{is a witness of } t} q(D \ltimes t')$

Enumerate $q(D \ltimes t')$ for $t' \in R_{e'}$

- Let $e'_0(= e'), e'_2, \cdots, e'_k(= r)$ be the set of nodes lying on the path from e' to root r
- Retrieve $t' \bowtie V_l(R_{e'_1}) \bowtie \cdots \bowtie V_l(R_{e'_k})$
- For every partial query result $(t', t_1, \cdots, t_k)$ retrieved:
  - Enumerate results for t' in $T_{e'_0}$
  - Enumerate results for t_1 in $T_{e'_1} \setminus T_{e'_0}$
  - ….
  - Enumerate results for t_k in $T_{e'_k} \setminus T_{e'_{k-1}}$
  - Combine them as Cartesian product



Running Example: initialization

R_1

x_1	x_2
1	2
2	2
3	3

R_2

x_2	x_3
1	2
2	2
4	3
1	1
2	4
1	4

R_3

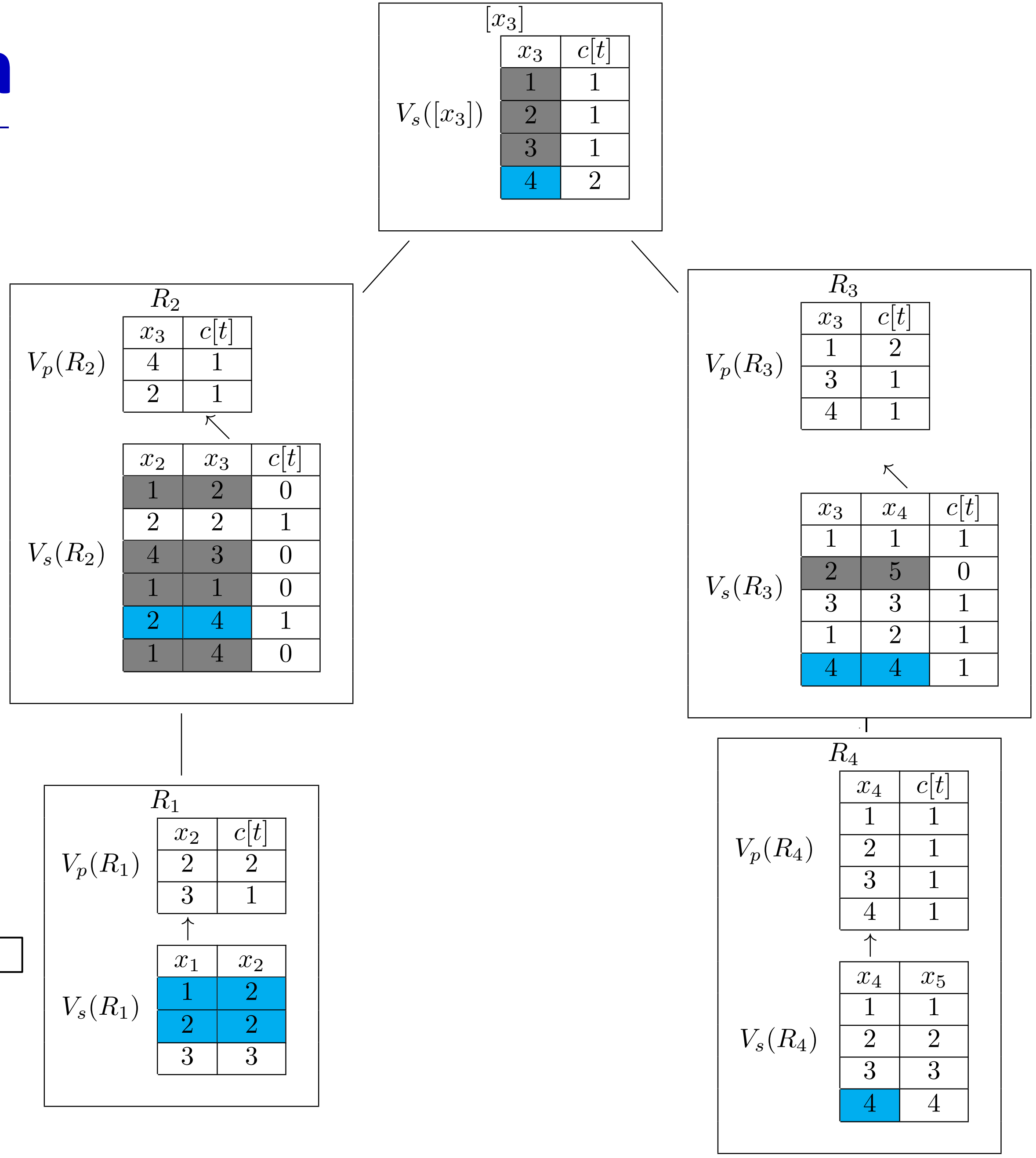
x_3	x_4
1	1
2	5
3	3
1	2
4	4

R_4

x_4	x_5
1	1
2	2
3	3
4	4

$q(D)$

x_1	x_2	x_3	x_4
1	2	4	4
2	2	4	4

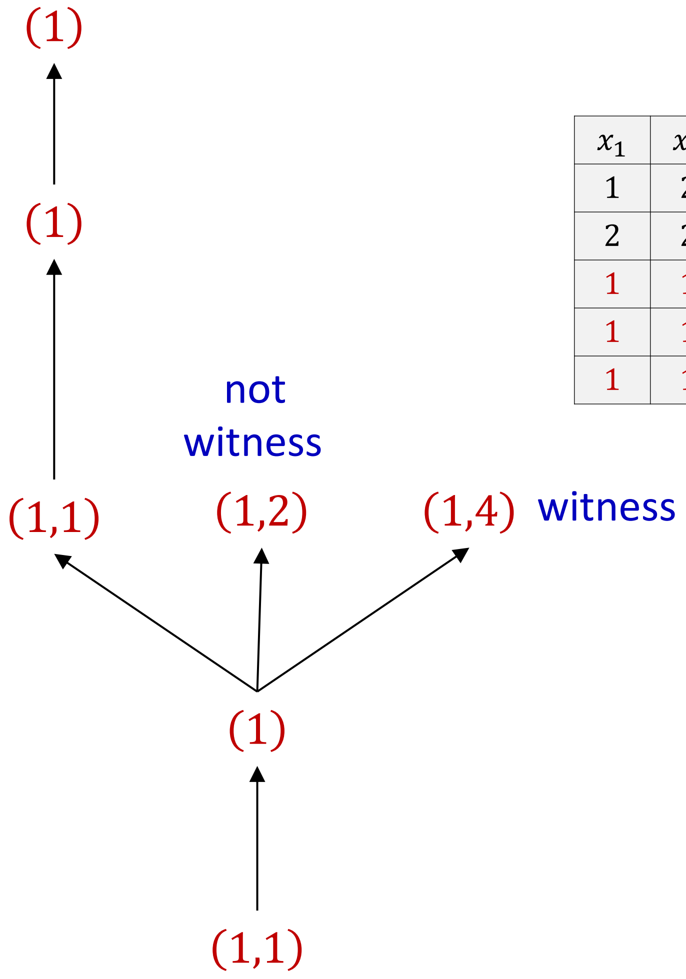


Tuples in base relation but not “exist” in V_s: (grey)

Tuples exist in V_s but not participate in query results: (white)

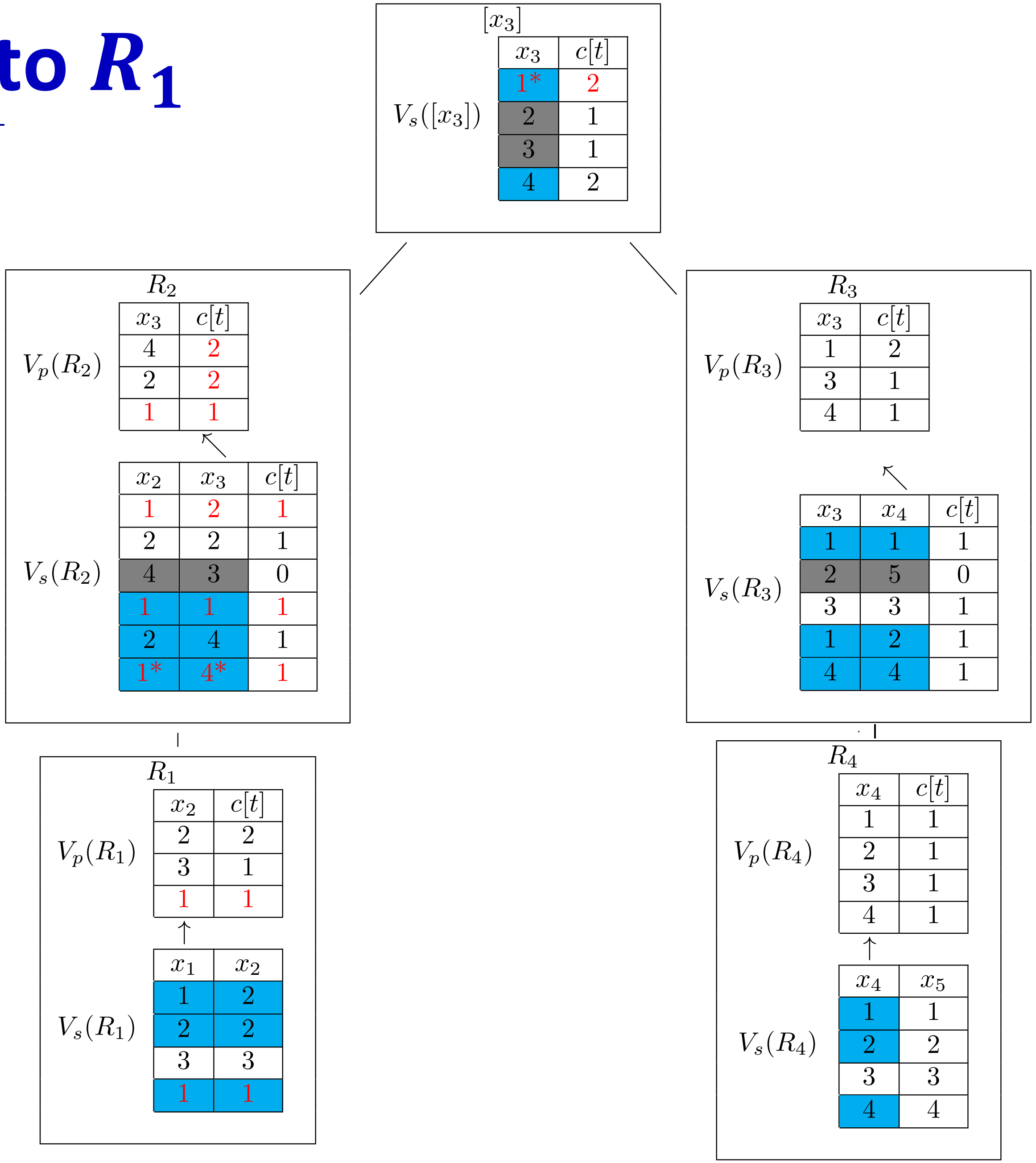
Tuples participate in query results: (blue)

Running Example: insert (1, 1) to R_1

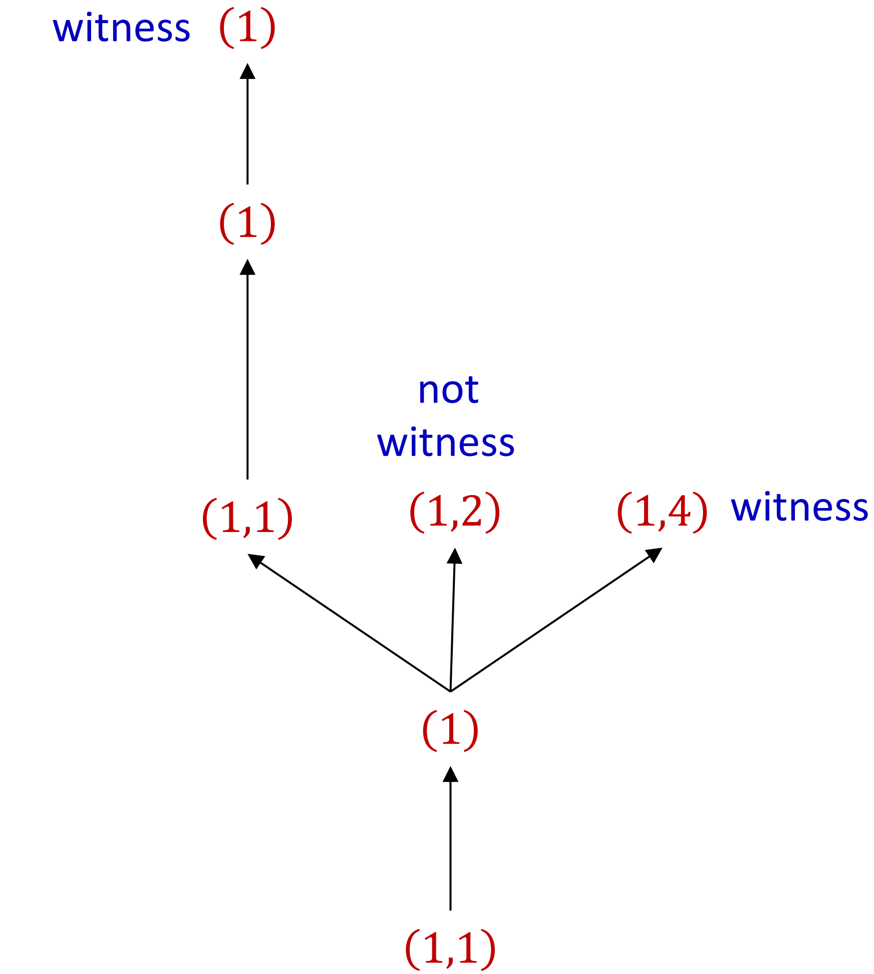


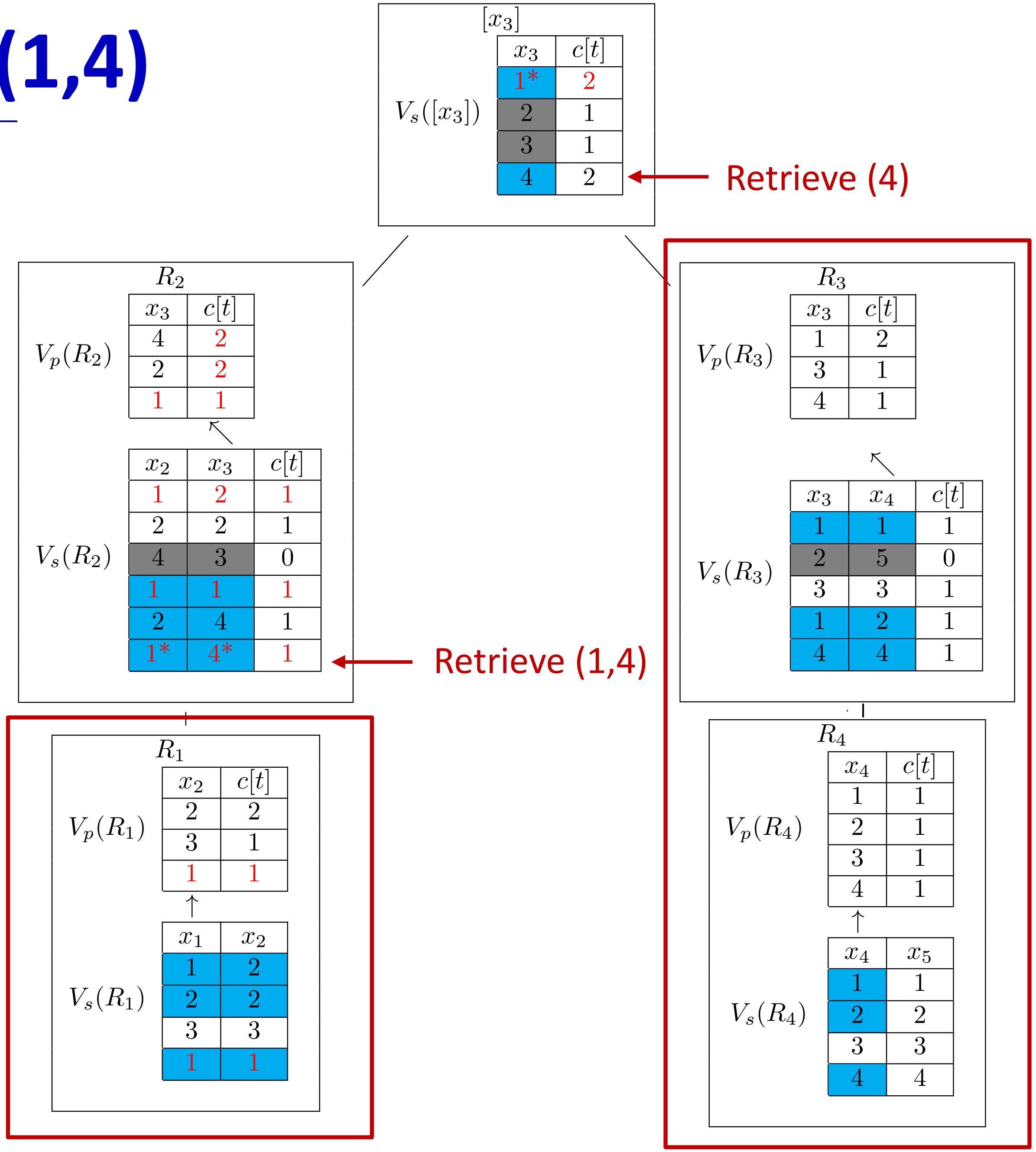
$q(D)$

x_1	x_2	x_3	x_4
1	2	4	4
2	2	4	4
1	1	1	1
1	1	1	2
1	1	4	4



Running Example: Enumerate (1,4)





$V_s([x_3])$

x_3	$c[t]$
1*	2
2	1
3	1
4	2

$V_p(R_2)$

x_3	$c[t]$
4	2
2	2
1	1

$V_s(R_2)$

x_2	x_3	$c[t]$
1	2	1
2	2	1
4	3	0
1	1	1
2	4	1
1*	4*	1

$V_p(R_3)$

x_3	$c[t]$
1	2
3	1
4	1

$V_s(R_3)$

x_3	x_4	$c[t]$
1	1	1
2	5	0
3	3	1
1	2	1
4	4	1

$V_p(R_1)$

x_2	$c[t]$
2	2
3	1
1	1

$V_s(R_1)$

x_1	x_2
1	2
2	2
3	3
1	1

$V_p(R_4)$

x_4	$c[t]$
1	1
2	1
3	1
4	1

$V_s(R_4)$

x_4	x_5
1	1
2	2
3	3
4	4

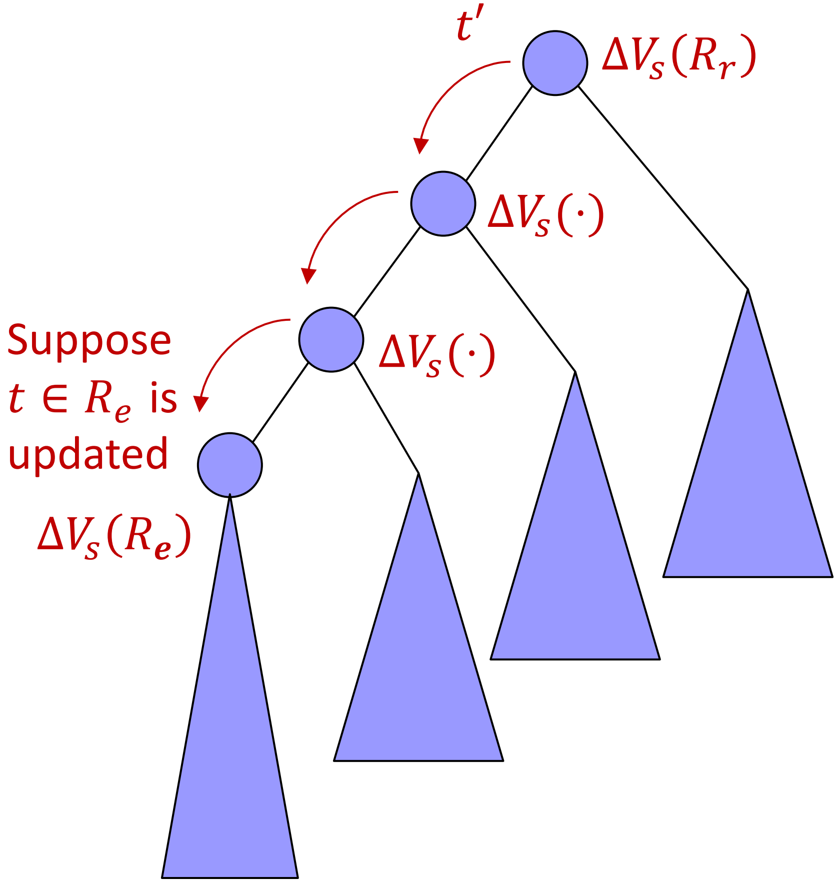
Delta Enumeration With Aggregation

- Lemma: For $t \in V_S(R_e)$,

$$c[t] = \sum_{t' \in \bowtie_{e' \in T_e} R_{e'} : \pi_e t' = t} \; \prod_{e' \in T_e} w[\pi_{e'} t']$$

  - T_e: the set of relations residing in the subtree of T rooted at node e

- Every delta result must include $t' \in \Delta V_s(R_{e'})$ for each ancestor node e' of e
- Retrieve $t \bowtie \Delta V_s(\cdot) \bowtie \cdots \bowtie \Delta V_s(R_r)$
- For every partial query result retrieved, enumerate results in the corresponding subtree similarly.



Other Questions

- Is join-tree-based enclosureness good enough?
- Enclosureness of update-sequence for aggregations?
- How to handle more complicated update sequences?
- How to adaptatively find a good generalized join tree?
- How to support more general joins? [IUVVL, VLDBJ’2020]
- How to handle batch updates more efficiently?
- What is the hardness result when self-join exists?