CONJUNCTIVE QUERIES WITH COMPARISONS

Qichen Wang

Nanyang Technological University

CS 848

Conjunctive Queries

$$ans(\bar{y}) \leftarrow R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$$

- Relations: $R_1, \ldots, R_n$
- Attributes/Variables: $\bar{x}_1, \ldots, \bar{x}_n$
- Output attributes: $\bar{y}$
- Full Query: $\bar{y} = \bar{x}_1 \cup \cdots \cup \bar{x}_n$, the head "$ans(\bar{y}) \leftarrow$" can be omitted
- Non-full Query: $\bar{y} \subset \bar{x}_1 \cup \cdots \cup \bar{x}_n$

Conjunctive Query and SQL

- A conjunctive query is equivalent to a (Nature) Join-Project SQL query.

$$ans(\bar{y}) \leftarrow R_1(\bar{x}_1), \dots, R_n(\bar{x}_n)$$

SELECT DISTINCT $\bar{y}$
FROM R_1 NATURAL JOIN R_2 … NATURAL JOIN R_n

- If the query is full:

$$R_1(\bar{x}_1), \dots, R_n(\bar{x}_n)$$

SELECT *
FROM R_1 NATURAL JOIN R_2 … NATURAL JOIN R_n

Conjunctive Queries with Selection

- Another important relational operator is selection (predicate)

$$ans(\bar{y}) \leftarrow \psi_c(R_1(\bar{x}_1), \dots, R_n(\bar{x}_n))$$

- Predicates: C
- We distinguish the predicates into two types:
  - Type-1: involve attributes from one relation $R_1.x_1 < 5$
  - Type-2: involve attributes from two or more relations $R_1.x_1 < R_2.x_3$

- Type-1 predicate can be checked by scanning the relation in linear time (or using an index if available);
- Note: Equalities can be rewritten into the CQ

Definition: CQC

- We consider type-2 filters in the form of
$$C \coloneqq f(\bar{x}_a) \leq g(\bar{x}_b)$$
- f, g: a function mapping $\boldsymbol{dom}(\bar{x}_a)$ (resp. $\boldsymbol{dom}(\bar{x}_b)$) to $\mathbb{R}$
- Assume $\bar{x}_a$ (resp. $\bar{x}_b$) are the attributes for relation R_a (resp. R_b)
- We say the comparison C incident to R_a and R_b

CQC and SQL

- A CQC is equivalent to a Select-Project-Join(SPJ) SQL query.

$$ans(\bar{y}) \leftarrow \psi_c(R_1(\bar{x}_1), \dots, R_n(\bar{x}_n))$$

SELECT DISTINCT $\bar{y}$
FROM R_1 NATURAL JOIN R_2 … NATURAL JOIN R_n
WHERE C

- If the query is full:

$$\psi_c(R_1(\bar{x}_1), \dots, R_n(\bar{x}_n))$$

SELECT *
FROM R_1 NATURAL JOIN R_2 … NATURAL JOIN R_n
WHERE C

Acyclicity of Conjunctive Queries

- A CQ can be evaluated in polynomial time (data complexity)

- A class of full conjunctive queries can be solved in linear time in data complexity, i.e., $O(N + OUT)$ time.
  - N: size of the database, OUT: size of the query result $|q(D)|$

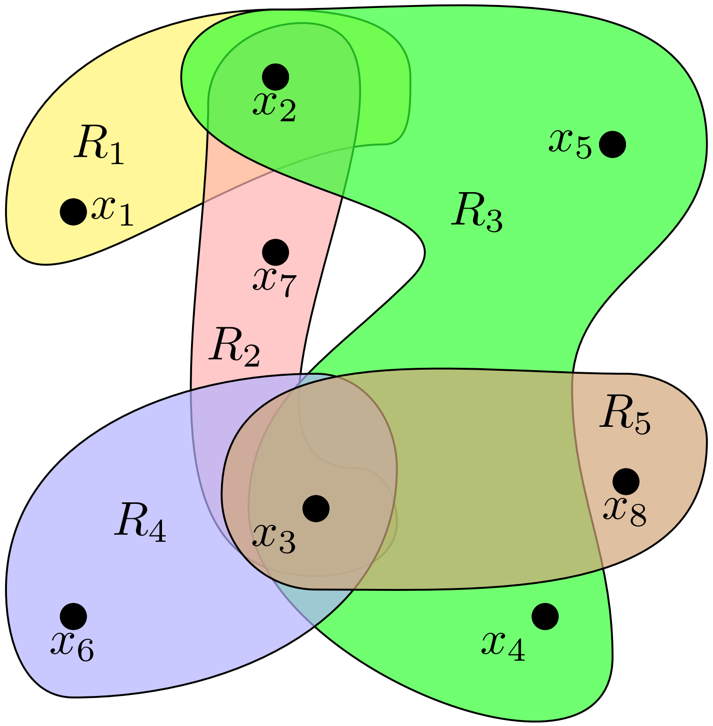
acyclic conjunctive queries

- The acyclicity of a CQ q is defined by the *α-acyclicity* of its *relation hypergraph*

CQ as a Hypergraph

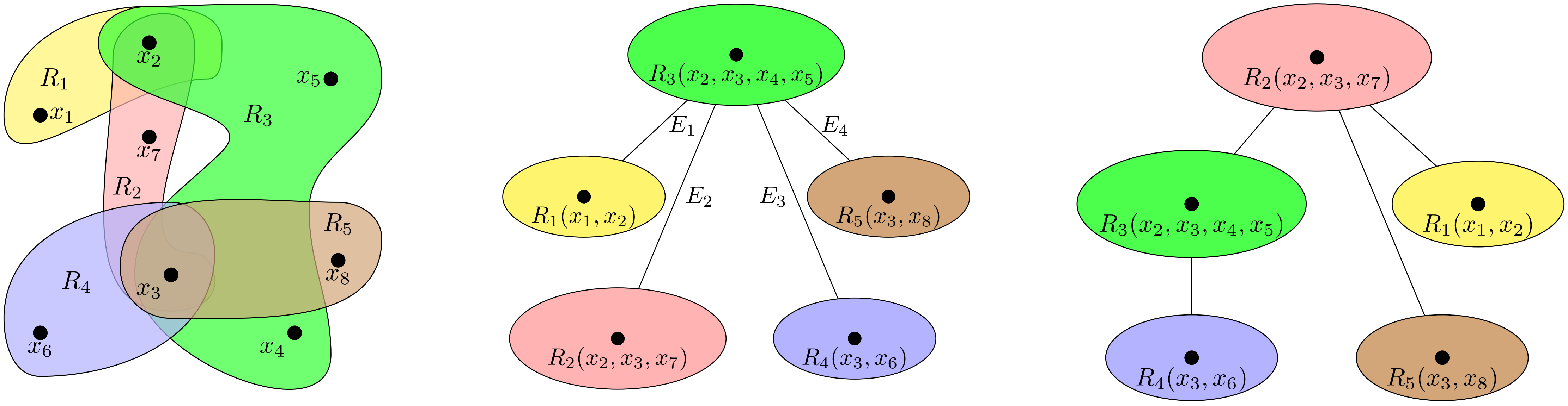
- Relational Hypergraph for query q: $H(q) = (V, E)$
- V: All attributes in query q
- E: All relations in query q

$$
\begin{aligned}
ans(x_1, x_2, x_3, x_4, x_7) \leftarrow\ & R_1(x_1, x_2), \\
& R_2(x_2, x_3, x_7), \\
& R_3(x_2, x_3, x_4, x_5), \\
& R_4(x_3, x_6), \\
& R_5(x_3, x_8)\ , \\
& C_1\colon\ x_1 - x_2\ \leq\ x_3 x_4 + 2, \\
& C_2\colon \min\{2x_2, x_7\} \leq\ x_6, \\
& C_3\colon x_2 \leq\ x_8
\end{aligned}
$$



α-Acyclicity

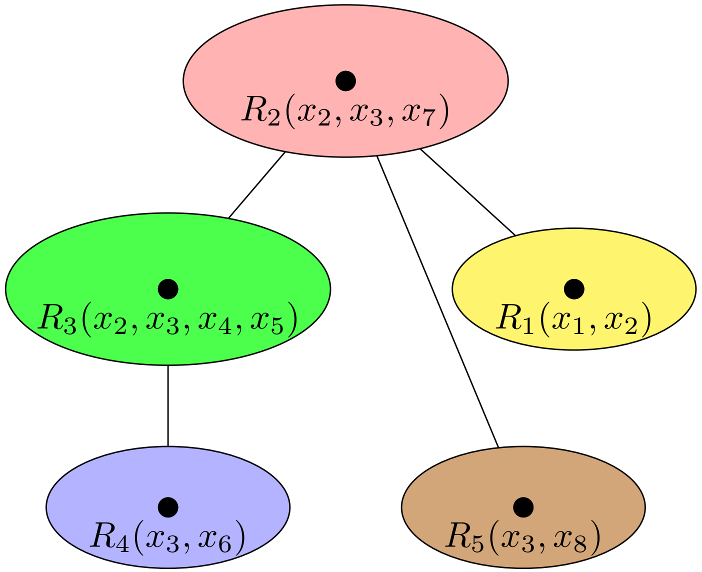
- A query is α-acyclic if it has a join tree.
- Join tree: an undirected tree whose nodes are in one-to-one correspondence with the edges(relations) in E such that for any vertex(attribute) $v \in V$, all nodes containing v form a connected subtree.



Yannakakis Algorithm [Yannakakis. '81]

- A linear-time algorithm for α-acyclic full queries.

- Algorithm:
  - Semijoin Phase: $O(N)$
    - Bottom-up: from leaves to root, semijoin each relation with each of its children
    - Top-down: from root to leaves, semijoin each relation with its parent
  - Join Phase: in arbitrary order $O(OUT)$



Bottom-up:

$R_3 := R_3 \ltimes R_4$
$R_2 := R_2 \ltimes R_3$
$R_2 := R_2 \ltimes R_1$
$R_2 := R_2 \ltimes R_5$

Top-down:

$R_3 := R_3 \ltimes R_2$
$R_4 := R_4 \ltimes R_3$
$R_1 := R_1 \ltimes R_2$
$R_5 := R_5 \ltimes R_2$

Type-2 Predicate

The query

$$R_1(x_1, x_2), R_2(x_2, x_3), C_1: x_1 \leq x_2, C_2: x_1 \geq x_3$$

has two comparisons, where $x_1 \leq x_2$ is a type-1 predicate and $x_1 \geq x_3$ is a type-2 predicate

- Type-2 predicate-> check on every join result and drop all results that cannot pass the predicate
- In this example,
  - compute

$$\mathrm{R}_1(x_1, x_2), R_2(x_2, x_3), C: x_1 \leq x_2$$

using the Yannakakis algorithm in $O(N + J)$ time, where J is the join size

  - For every query result (x_1, x_2, x_3), check whether the $x_1 \geq x_3$ is satisfied, taking $O(J)$ time

- However, it's possible that OUT (# join results after applying the predicates) $\ll J$

Example

$$R_1(x_1, x_2), R_2(x_2, x_3), C_1\colon x_1 \geq x_3, C_2\colon x_1 \leq x_2$$

- Let R_1 contains n tuples as $(1, 2n), \cdots, (n, 2n)$, R_2 contains n tuples as $(2n, 2n + 1), \cdots, (2n, 3n)$

R_1

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

$\bowtie$

R_2

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

$=$

x_1	x_2	x_3
1	$2n$	$2n+1$
1	$2n$	$\ldots$
1	$2n$	$3n$
$\ldots$		
n	$2n$	$2n+1$
n	$2n$	$\ldots$
n	$2n$	$3n$

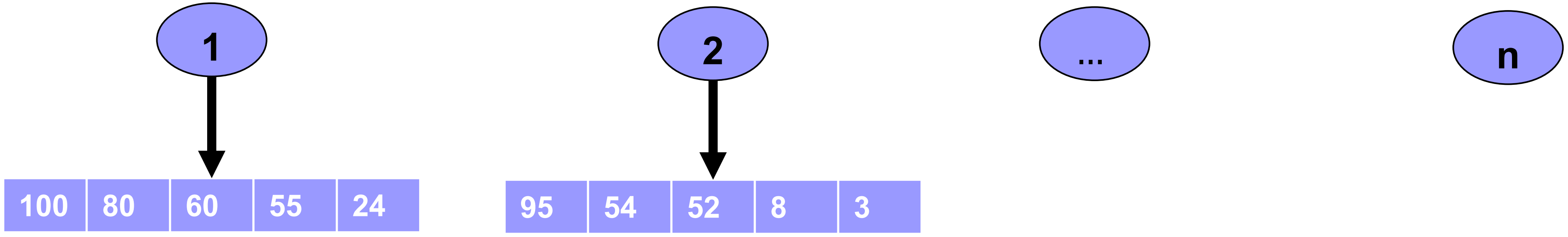
- $J = n^2, OUT = 0$

- The cost is not linear.

Willard's Approach [Willard. '02]

$$R_1(x_1, x_2), R_2(x_2, x_3), C: x_1 \geq x_3$$

1. Group R_1 by x_2, for each group, sort the x_1 value in descending order.



2. For every tuple $t = (a, b) \in R_2$, enumerate the list linked by $x_2 = a$ from the beginning until it is smaller than b.

For example, $t = (2, 60)$, the enumeration procedure will first find $x_1 = 95$, which corresponds to a join result $(95, 2, 60)$, then it will stop, as the second value $54 < 60$.

3. Total running time: O($N \log N + OUT$)

Willard's Approach [Willard. '02]

- Can support multiple (short) comparisons between R_1 and R_2
- Running time: $O\left(N \log^d N + OUT\right)$.
  - d: number of comparisons.

- Not appliable for long comparisons across multiple relations.

$$\boldsymbol{R_1(x_1, x_2), R_2(x_2, x_3), R_3(x_3, x_4), x_1 \leq x_4}$$

Main Contributions

- We identify the acyclicity conditions for CQCs.
- For any acyclic CQC, our algorithm can evaluate it in $\tilde{O}(N + OUT)$
- We implement the algorithm on top of Spark, the experiment results show the new algorithms can offer an order-of-magnitude improvement over competitors

Acyclicity of CQCs

- Observations: a CQC is ``hard'' if its comparisons form a ``cycle''.

- For example, $R_1(x_1, x_2), R_2(x_2, x_3), R_3(x_3, x_4), x_1 \leq x_4, x_1 \geq x_4$ cannot be solved in linear time.
- The query is equivalent to the triangle listing query

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

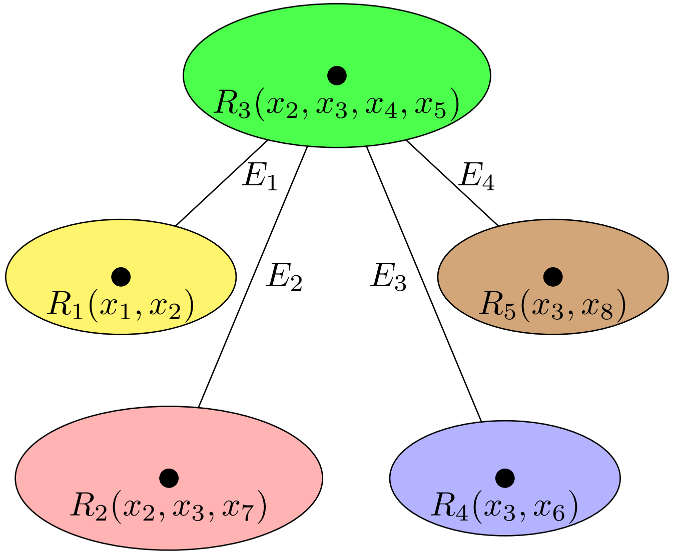
- The best-known algorithm for the problem: $O(N^{1.5})$
- Lower bound for the problem: $\Omega(N^{\frac{4}{3}})$, which is not linear.

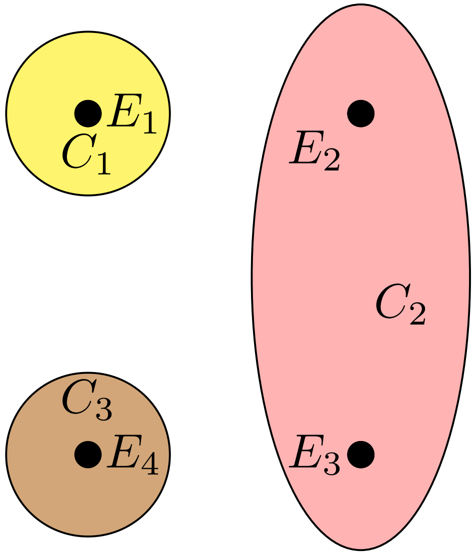
- Comparison Hypergraph: another hypergraph to characterize the acyclicity on the comparisons.

Comparison Hypergraph

- Fixing a join tree T of q, the comparison hypergraph $C(q, T) = (V, E)$
  - Each vertex $v \in V$ represents an edge on the join tree
  - Each hyperedge $e \in E$ represents a comparison C incident to R_i and R_j
  - A vertex $v \in e$ if v is in the path between R_i and R_j on the join tree T

$$
\begin{aligned}
ans(x_1, x_2, x_3, x_4, x_7) \leftarrow\ & R_1(x_1, x_2), \\
& R_2(x_2, x_3, x_7), \\
& R_3(x_2, x_3, x_4, x_5), \\
& R_4(x_3, x_6), \\
& R_5(x_3, x_8), \\
& C_1\!: x_1 - x_2 \le x_3 x_4 + 2, \\
& C_2\!: \min\{2x_2, x_7\} \le x_6, \\
& C_3\!: x_2 \le x_8
\end{aligned}
$$

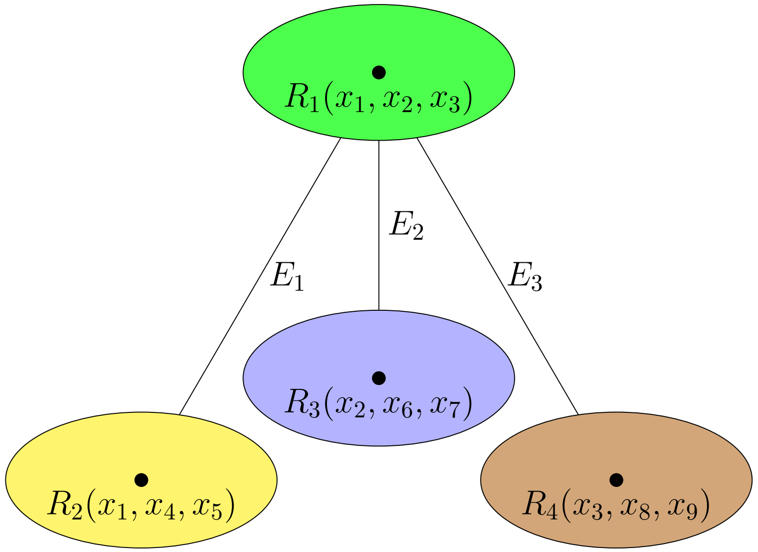


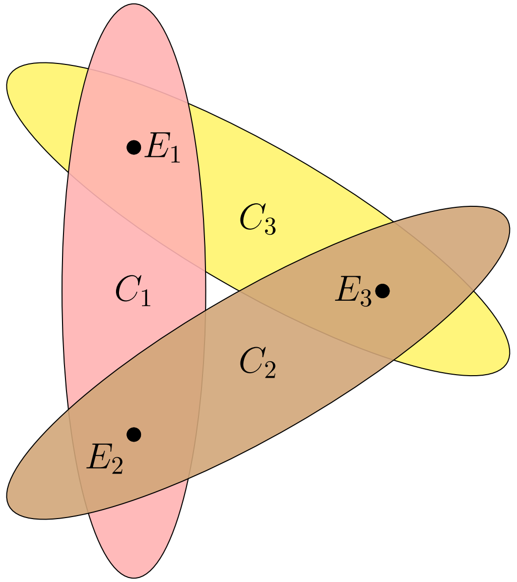


Acyclicity of CQCs

- A CQC is acyclic if
  - its relational hypergraph is α-acyclic
  - and there exists a join tree such that the comparison hypergraph is Berge-acyclic
- Berge-acyclic: for any pair of vertices v_1, v_2, there exists at most one simple path between v_1 and v_2
- A counter example:

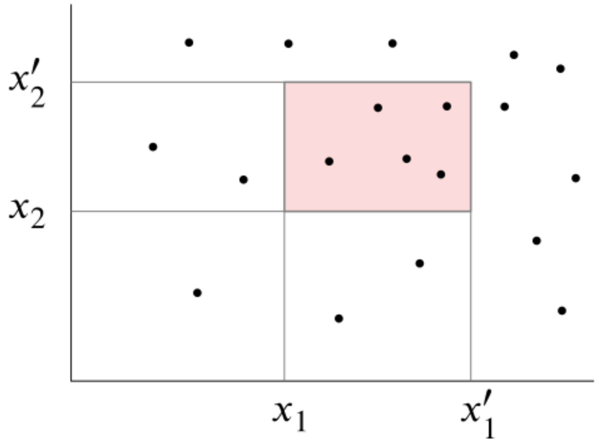
$$\begin{aligned} ans(x_1, x_2, x_3) \leftarrow\ & R_1(x_1, x_2, x_3), \\ & R_2(x_1, x_4, x_5), \\ & R_3(x_2, x_6, x_7), \\ & R_4(x_3, x_8, x_9), \\ & C_1\colon x_4 \le x_6, \\ & C_2\colon x_7 \le x_8, \\ & C_3\colon x_9 \le x_5 \end{aligned}$$





Comparisons as Orthogonal Range Searching

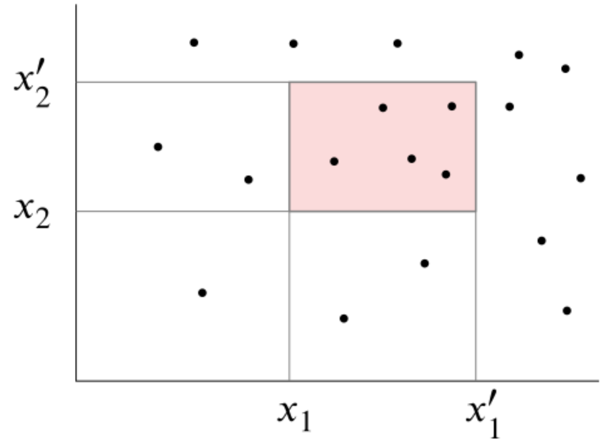
$$R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4) \bowtie R_3(x_4, x_5, x_6), x_1 \geq x_5, x_2 \geq x_6$$

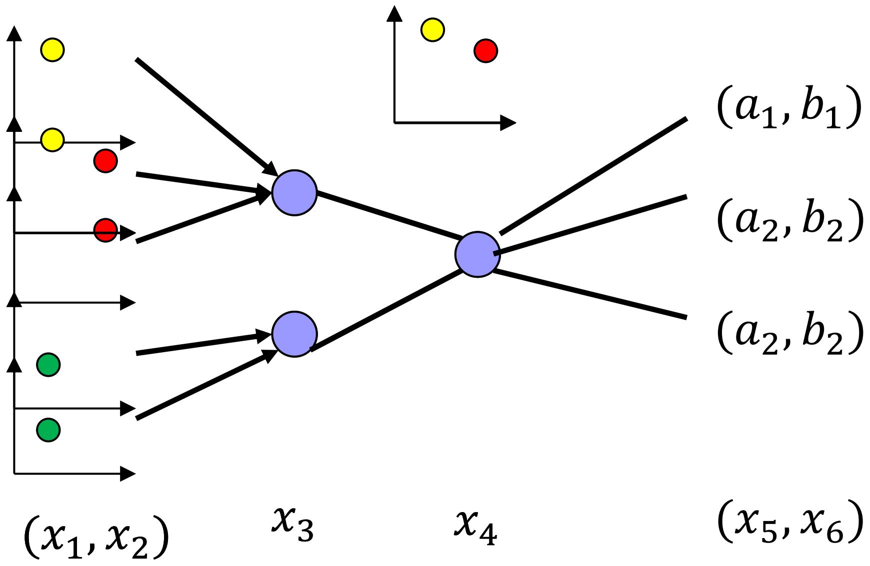


Not a very good example, as it cannot be evaluated in linear time.
Bonus Question: Why?

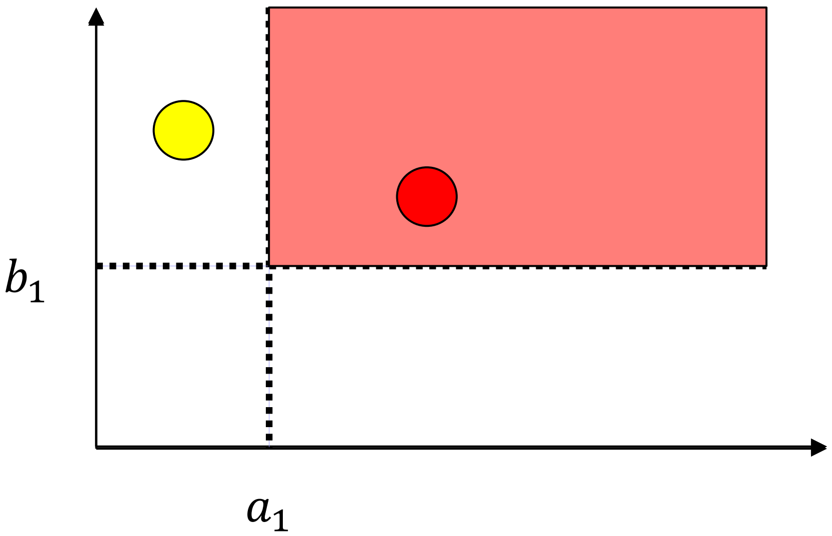
Comparisons as Orthogonal Range Searching

$$R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4) \bowtie R_3(x_4, x_5, x_6), x_1 \geq x_5, x_2 \geq x_6$$



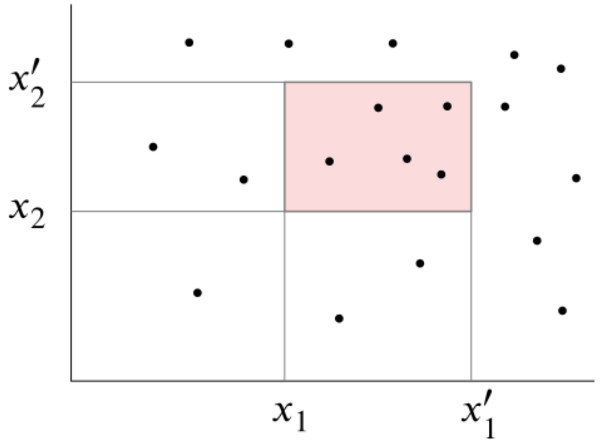


"Are there any points located in the orthogonal $([a_1, \infty], [b_1, \infty])$***?"***

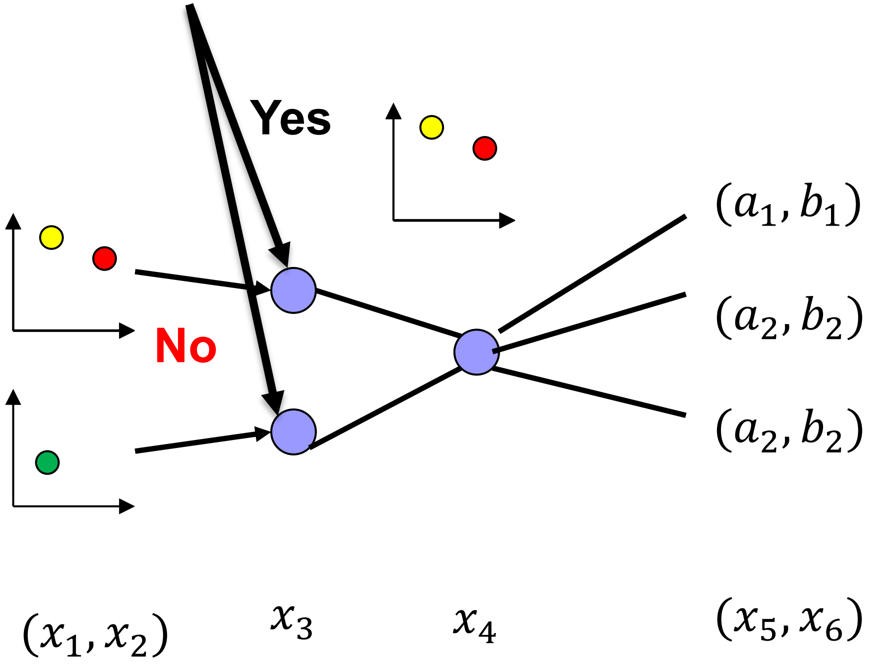


Comparisons as Orthogonal Range Searching

$$R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4) \bowtie R_3(x_4, x_5, x_6), x_1 \geq x_5, x_2 \geq x_6$$

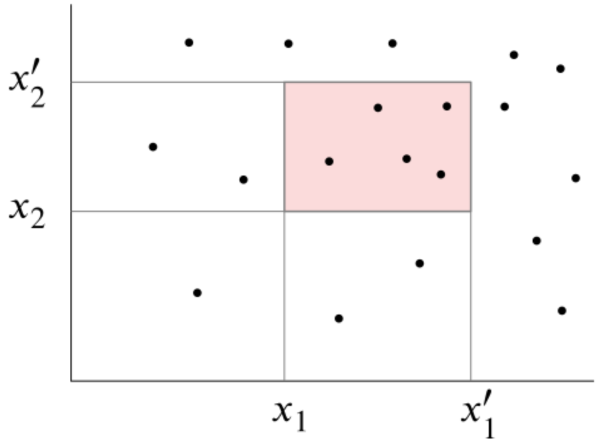


"Are there any points located in the orthogonal** $([a_1, \infty], [b_1, \infty])$**?"

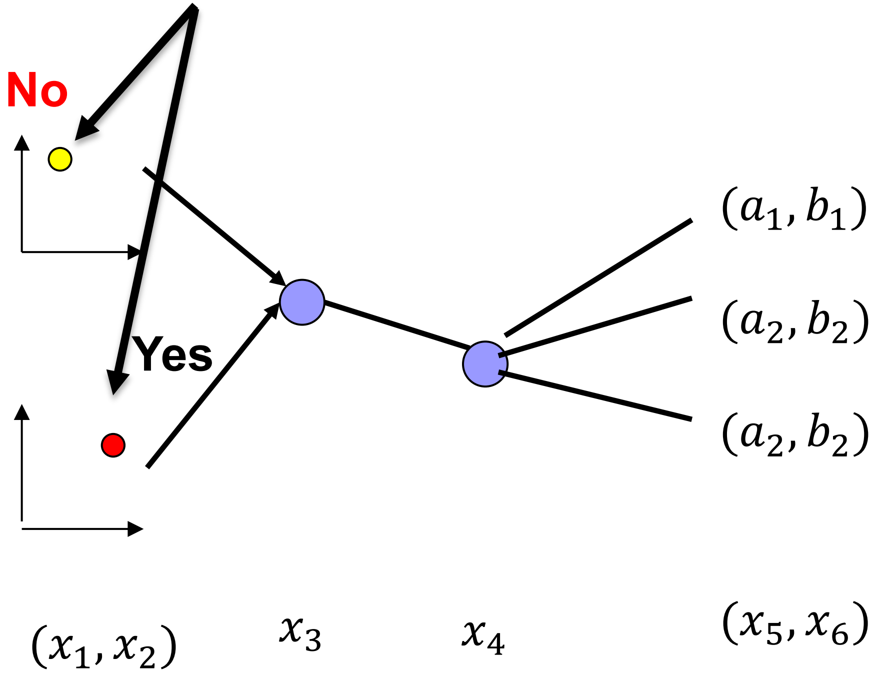


Comparisons as Orthogonal Range Searching

$$R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4) \bowtie R_3(x_4, x_5, x_6), x_1 \geq x_5, x_2 \geq x_6$$



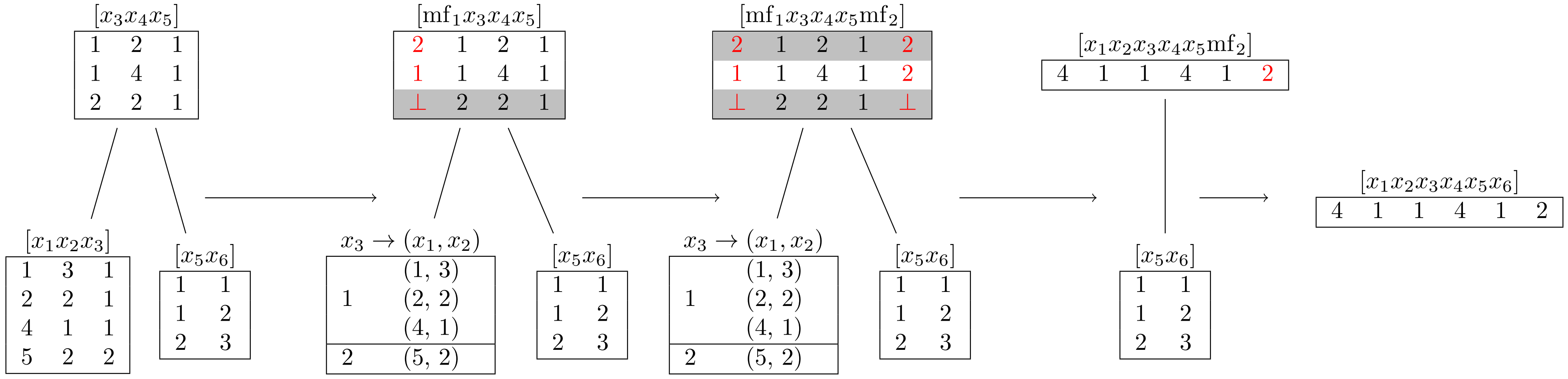
"Are there any points located in the orthogonal $([a_1, \infty], [b_1, \infty])$***?"***



Algorithms

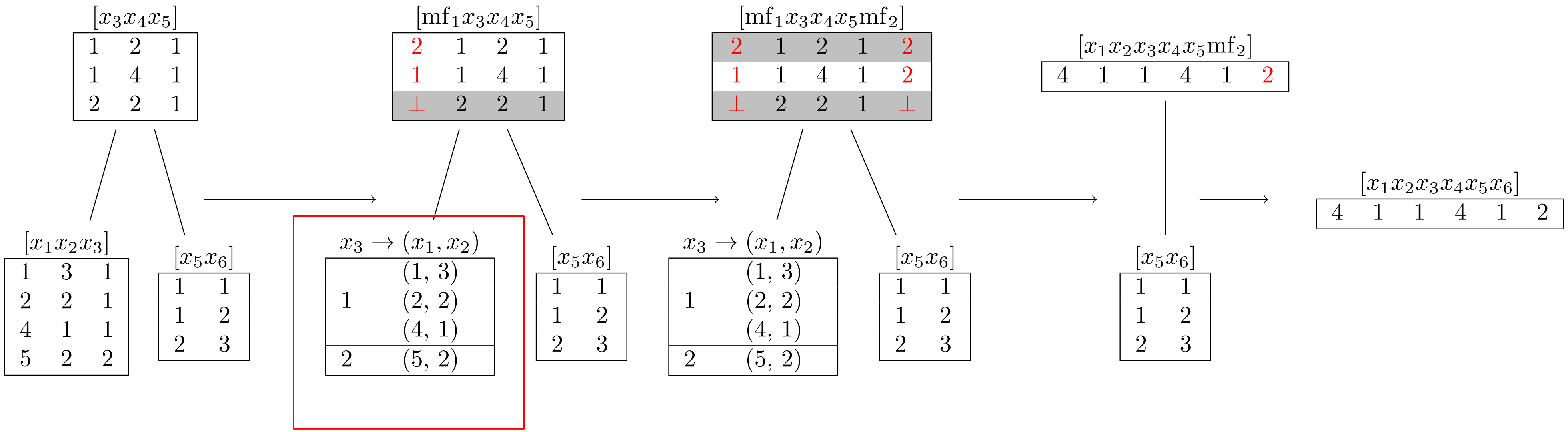
- For all acyclic CQC Q, there exists a reducible relation $R \in Q$.
  - A reducible relation is a leaf relation (degree = 1) that incident to at most one long comparison.
- We can perform a reduction from $Q \to Q'$, $E' \to E/R$.
  - Q' is still acyclic CQC;
  - Join result in Q can be built in $O(1)$ delay if the result of Q' can be built in $O(1)$ delay;
  - New attribute mf for the comparisons.

$$R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4, x_5) \bowtie R_3(x_5, x_6), x_1 \leq x_4, x_2 < x_6$$



Algorithms

- Step 1: Group R_1 on the join key and make an orthogonal range searching structure on $x_1 x_2$.

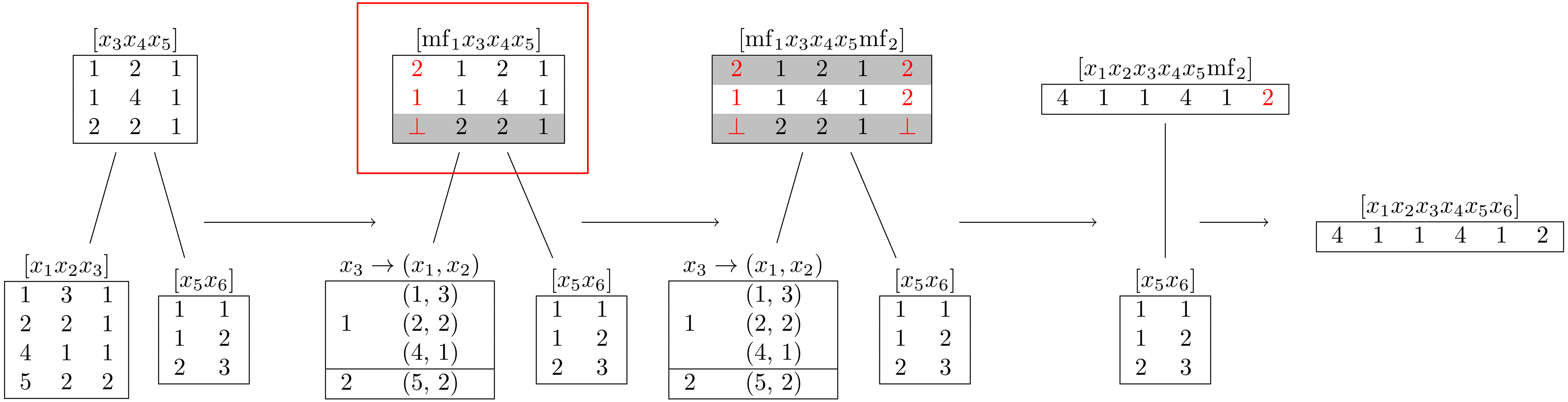


$$Q \coloneqq R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4, x_5) \bowtie R_3(x_5, x_6), x_1 \leq x_4, x_2 < x_6$$

Algorithms

- Step 2 (Reduce R_1): Reduction between R_2 and R_1. $R_2{}'$ is calculated by the following query:

SELECT $x_3, x_4, x_5,\ min(x_2)$ as mf_1 FROM R_1 NATURAL JOIN R_2 WHERE $R_1.x_1 \leq R_2.x_4$



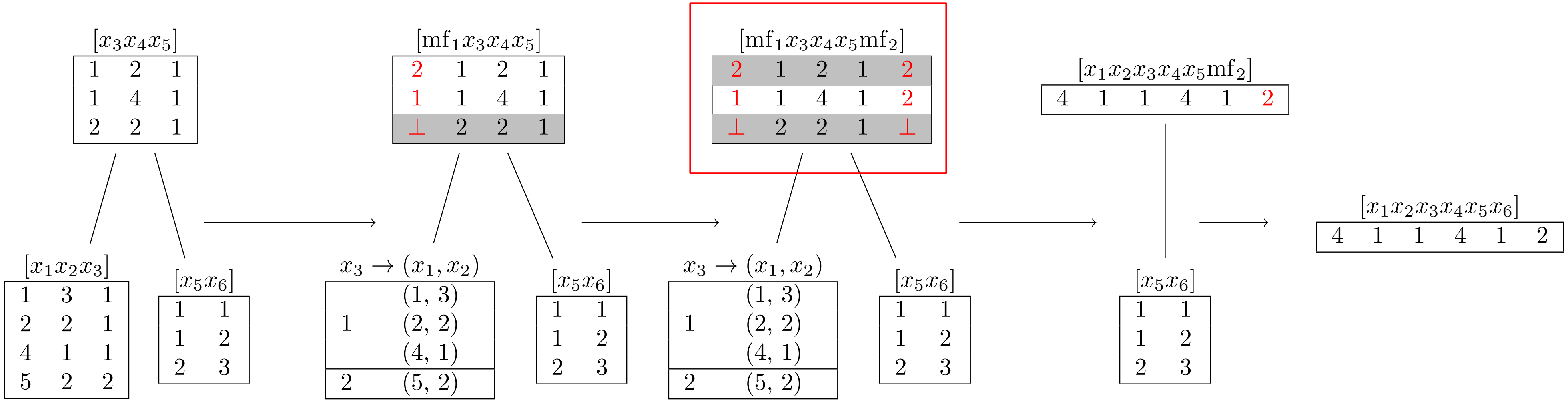
$$Q \coloneqq R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4, x_5) \bowtie R_3(x_5, x_6), x_1 \leq x_4, x_2 < x_6$$

$$Q_1 \coloneqq R_2'(x_3, x_4, x_5, mf_1) \bowtie R_3(x_5, x_6), mf_1 < x_6$$

Algorithms

- Step 3 (Reduce R_3): Reduction between $R_2{}'$ and R_3 by the following query:

SELECT $x_3, x_4, x_5, mf_1, max(x_6)$ as mf_2 FROM R_3 NATURAL JOIN $R_2{}'$



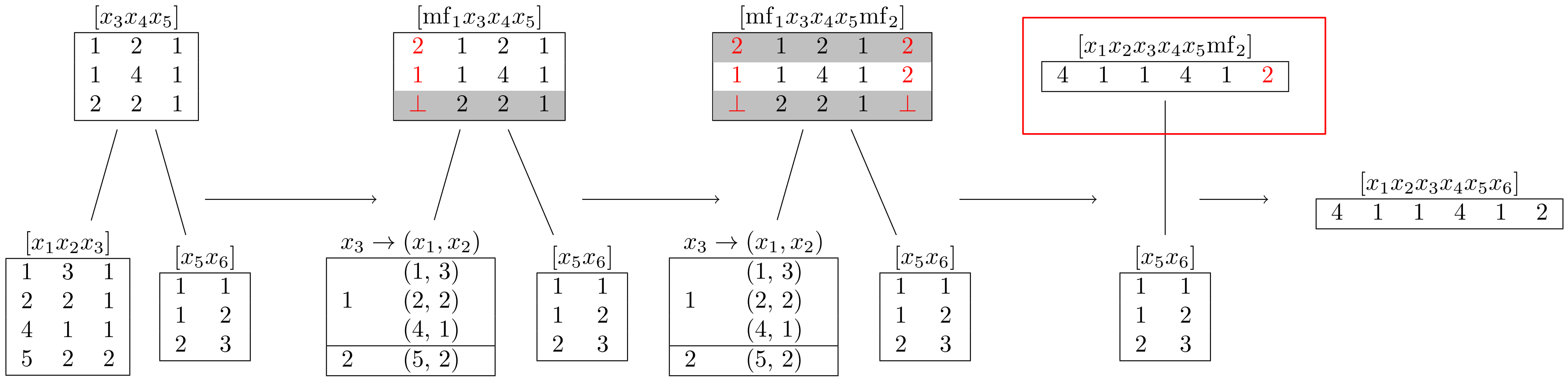
$$Q \coloneqq R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4, x_5) \bowtie R_3(x_5, x_6), x_1 \le x_4, x_2 < x_6$$

$$Q_1 \coloneqq R_2'(x_3, x_4, x_5, mf_1) \bowtie R_3(x_5, x_6), mf_1 < x_6$$

$$Q_2 \coloneqq R_2''(x_3, x_4, x_5, mf_1, mf_2), mf_1 < mf_2$$

Algorithms

- Step 4 (Evaluate Q_2): Remove all tuples in R_2'' that does not satisfy the filter condition $mf_1 < mf_2$



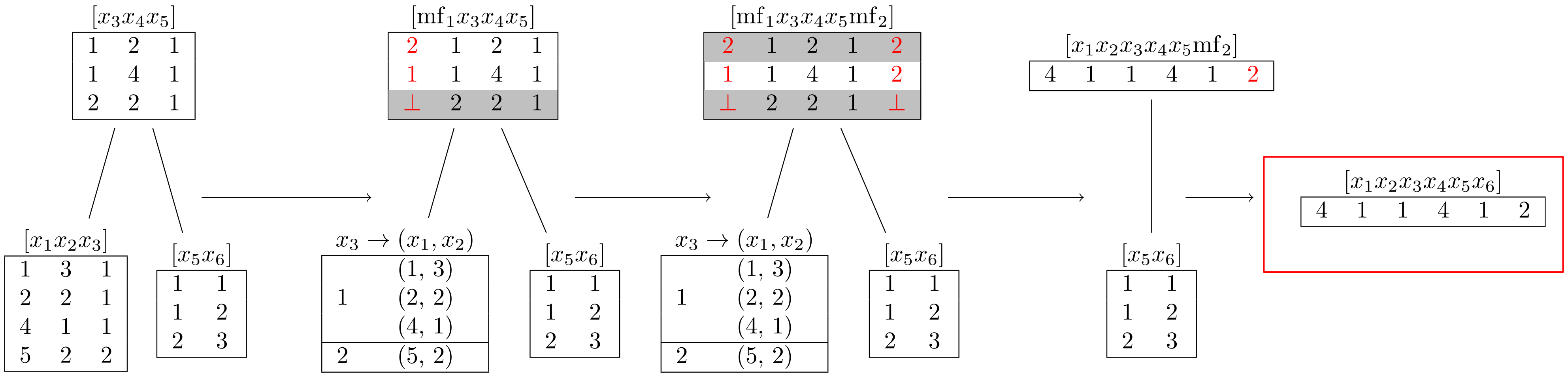
$$Q \coloneqq R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4, x_5) \bowtie R_3(x_5, x_6), x_1 \leq x_4, x_2 < x_6$$

$$Q_1 \coloneqq R_2'(x_3, x_4, x_5, mf_1) \bowtie R_3(x_5, x_6), mf_1 < x_6$$

$$Q_2 \coloneqq R_2''(x_3, x_4, x_5, mf_1, mf_2), mf_1 < mf_2$$

Algorithms

- Step 5 (Enumeration): Enumerate the query result from top-down.



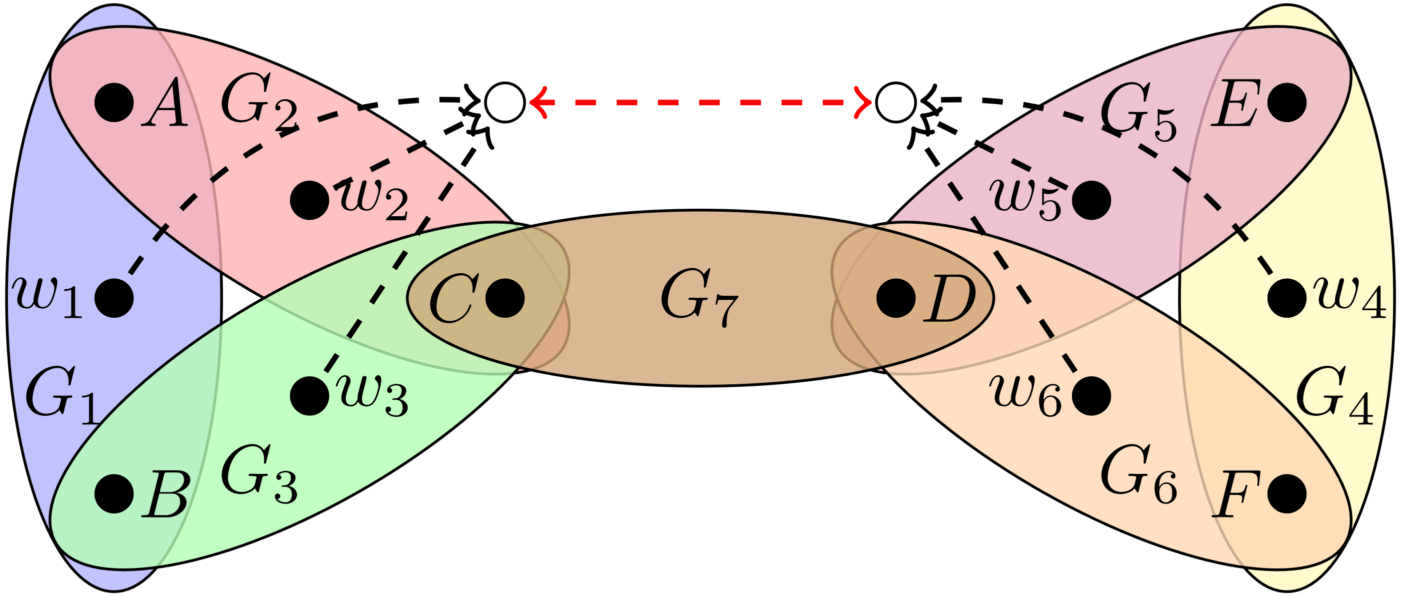
$$R_1(x_1, x_2, x_3) \bowtie R_2(x_3, x_4, x_5) \bowtie R_3(x_5, x_6), x_1 \leq x_4, x_2 < x_6$$

Extensions

- Support of non-full queries:
  - For free-connex queries: $\tilde{O}(N + OUT)$
  - Non-free-connex queries: $\tilde{O}(N^w + OUT)$
- Support for cyclic queries:
  - $\tilde{O}(N^w + OUT)$ with Generalized Hypertree Decomposition (GHD)
  - Thanks to the support of long comparisons, the tree weight w is smaller than previous approaches.

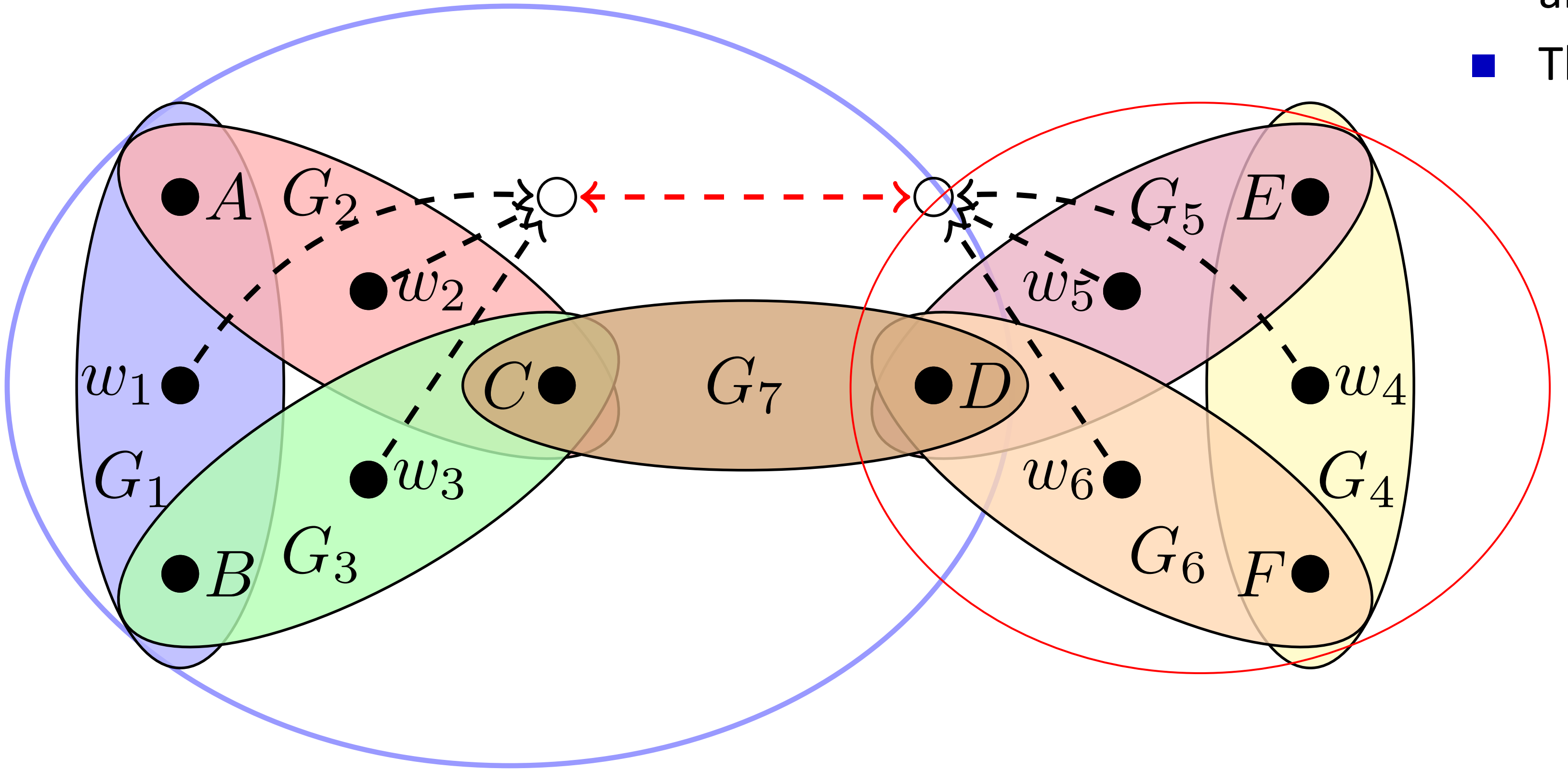
Cyclic Queries

$R(A, B, w_1), R(A, C, w_2), R(B, C, w_3), R(C, D),$
$R(E, F, w_4), R(D, E, w_5), R(D, F, w_6),$
$w_1 w_2 w_3 \leq w_4 w_5 w_6$



Cyclic Queries

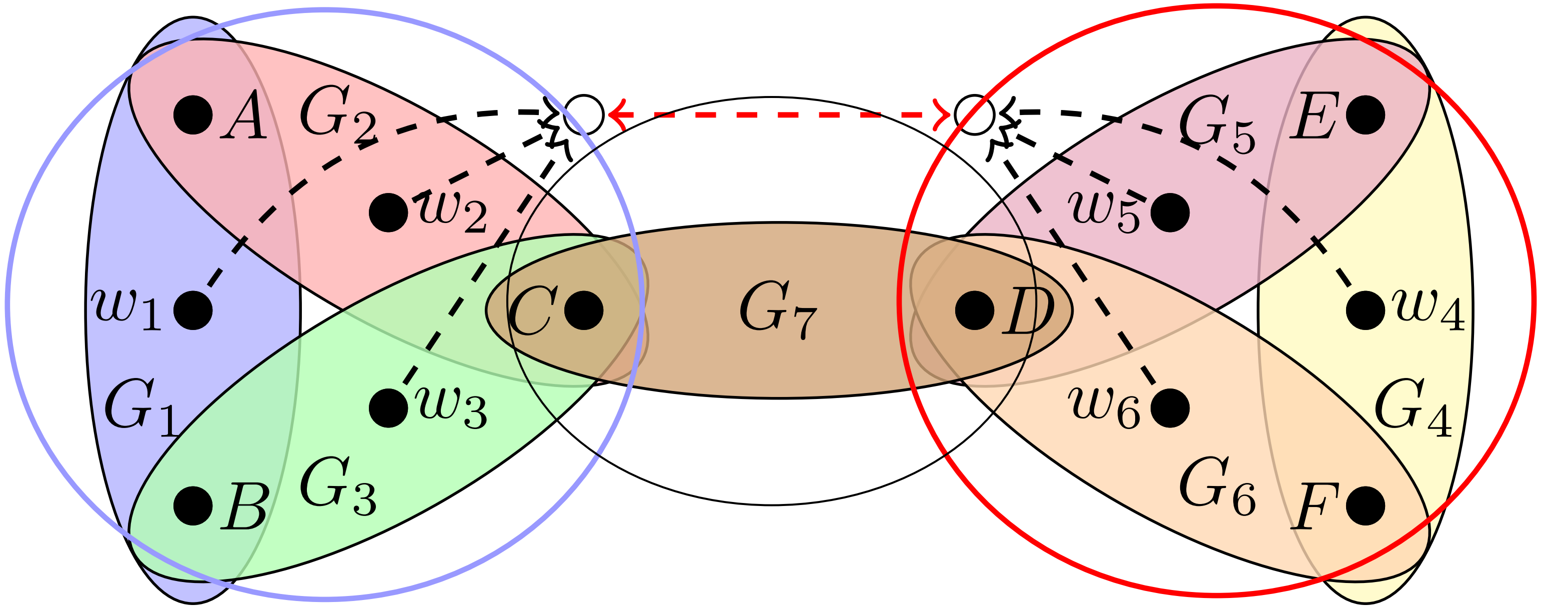
$R(A, B, w_1), R(A, C, w_2), R(B, C, w_3), R(C, D),$
$R(E, F, w_4), R(D, E, w_5), R(D, F, w_6),$
$w_1 w_2 w_3 \leq w_4 w_5 w_6$



- The previous approach does not support long comparisons.
- The best GHD contains two bags: ABCD and DEF (or ABC and CDEF)
- The width $w = 2$ for this query.

Cyclic Queries

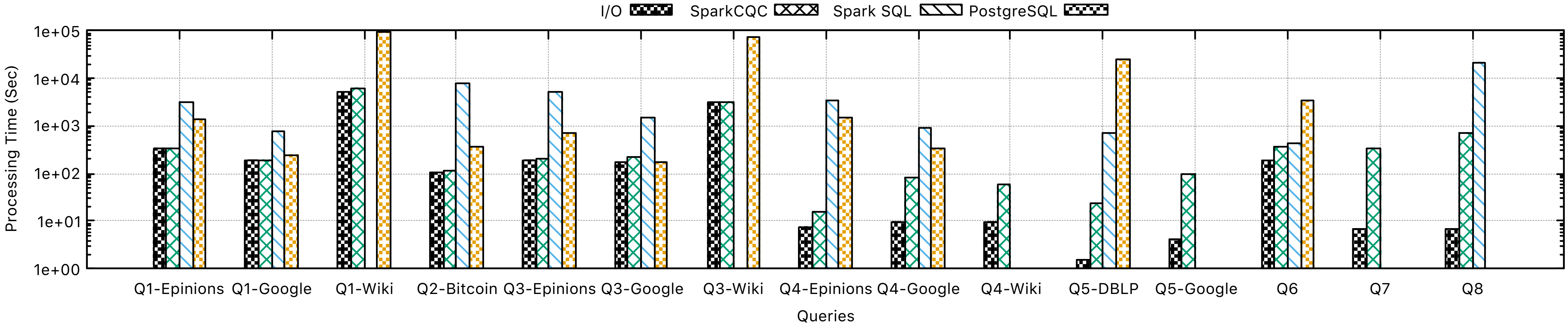
$$R(A,B,w_1), R(A,C,w_2), R(B,C,w_3), R(C,D),$$
$$R(E,F,w_4), R(D,E,w_5), R(D,F,w_6),$$
$$w_1 w_2 w_3 \le w_4 w_5 w_6$$



- Long comparisons are allowed in the new algorithm
- The best GHD contains three bags: ABC, CD and DEF
- The width $w = 1.5$ for this query.

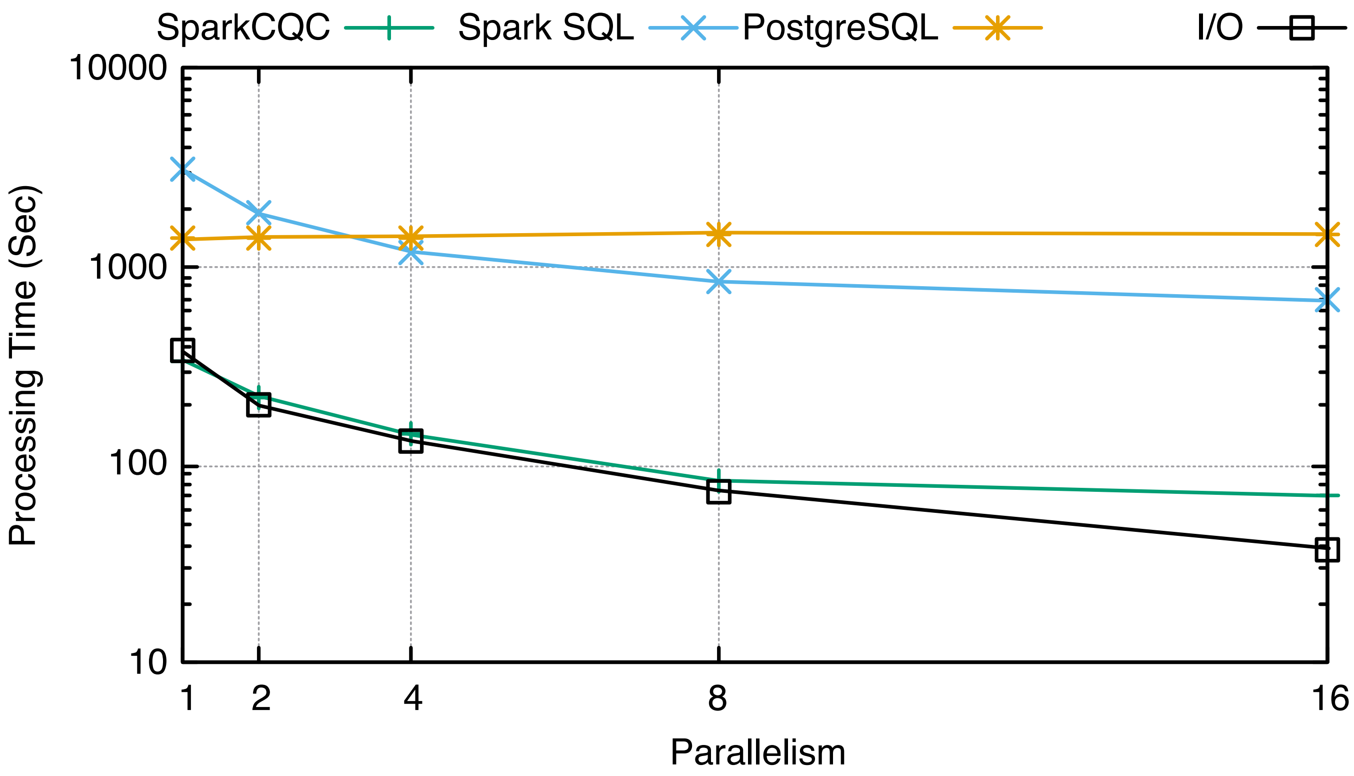
Experiment Results

- Build the algorithm on top of Spark.
- It requires only standard RDD operations.
- Compares with Spark SQL and PostgreSQL, we achieve order-of-magnitude improvement.



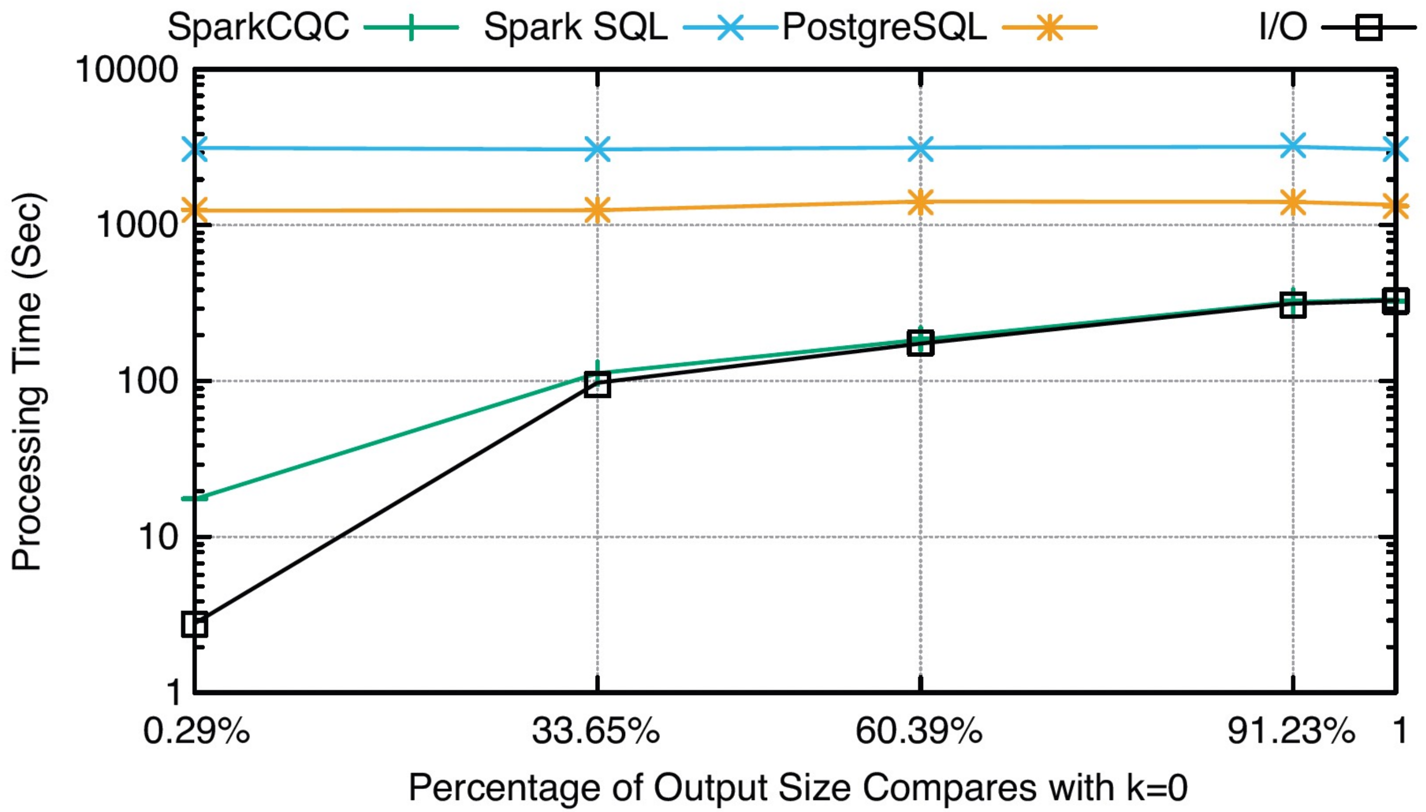
Experiment Results

- Achieve almost linear speedup when increasing the parallelism.



Experiment Results

- By evaluating the predicates during joins, the new algorithm can benefit from the low selectivity.



Thank You!