Scaling Up Symbolic Reasoning for Relational Queries

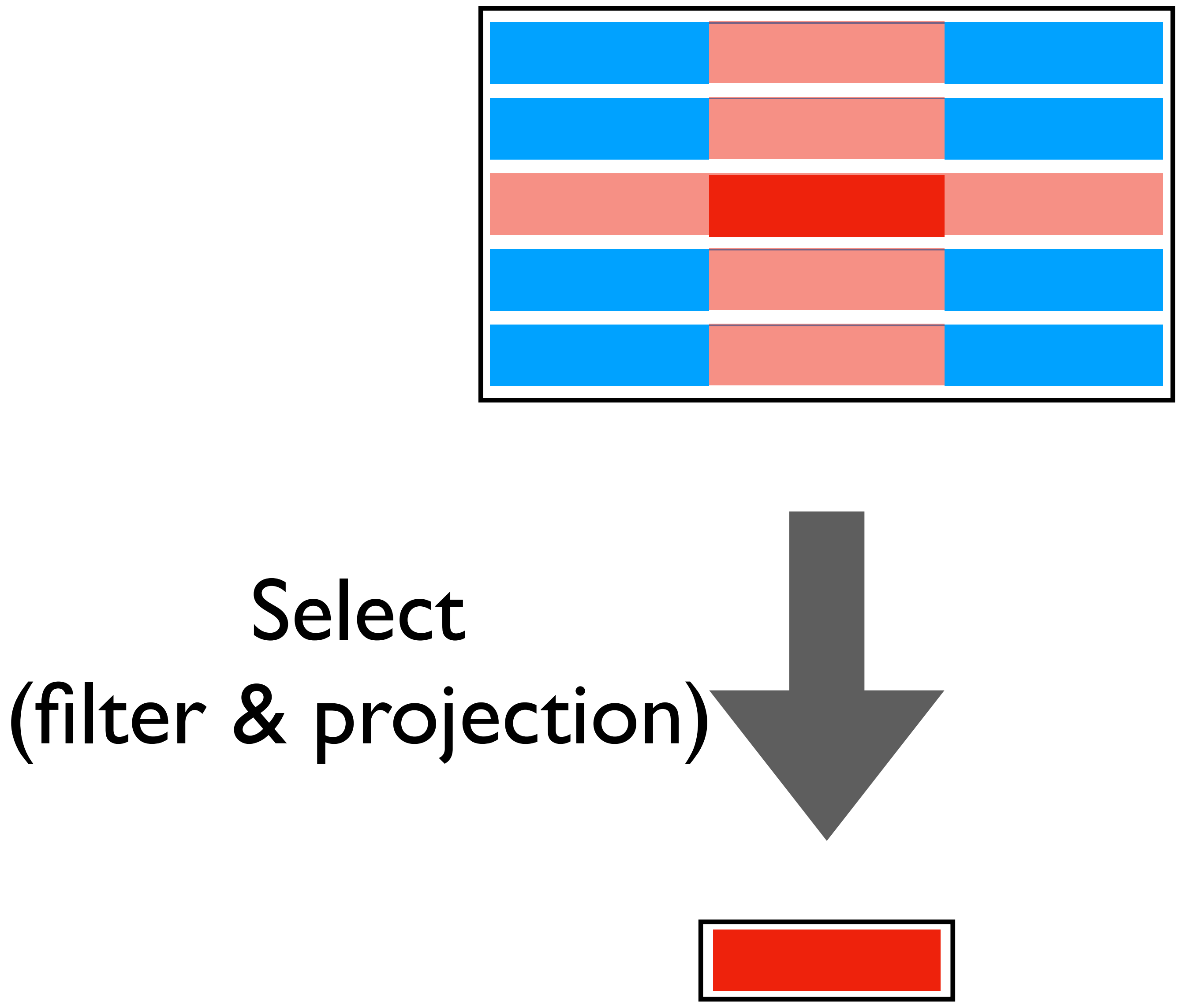
or, speeding up debugging & verification of database queries

Chenglong Wang, Alvin Cheung, Ras Bodik
University of Washington

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Relational Queries

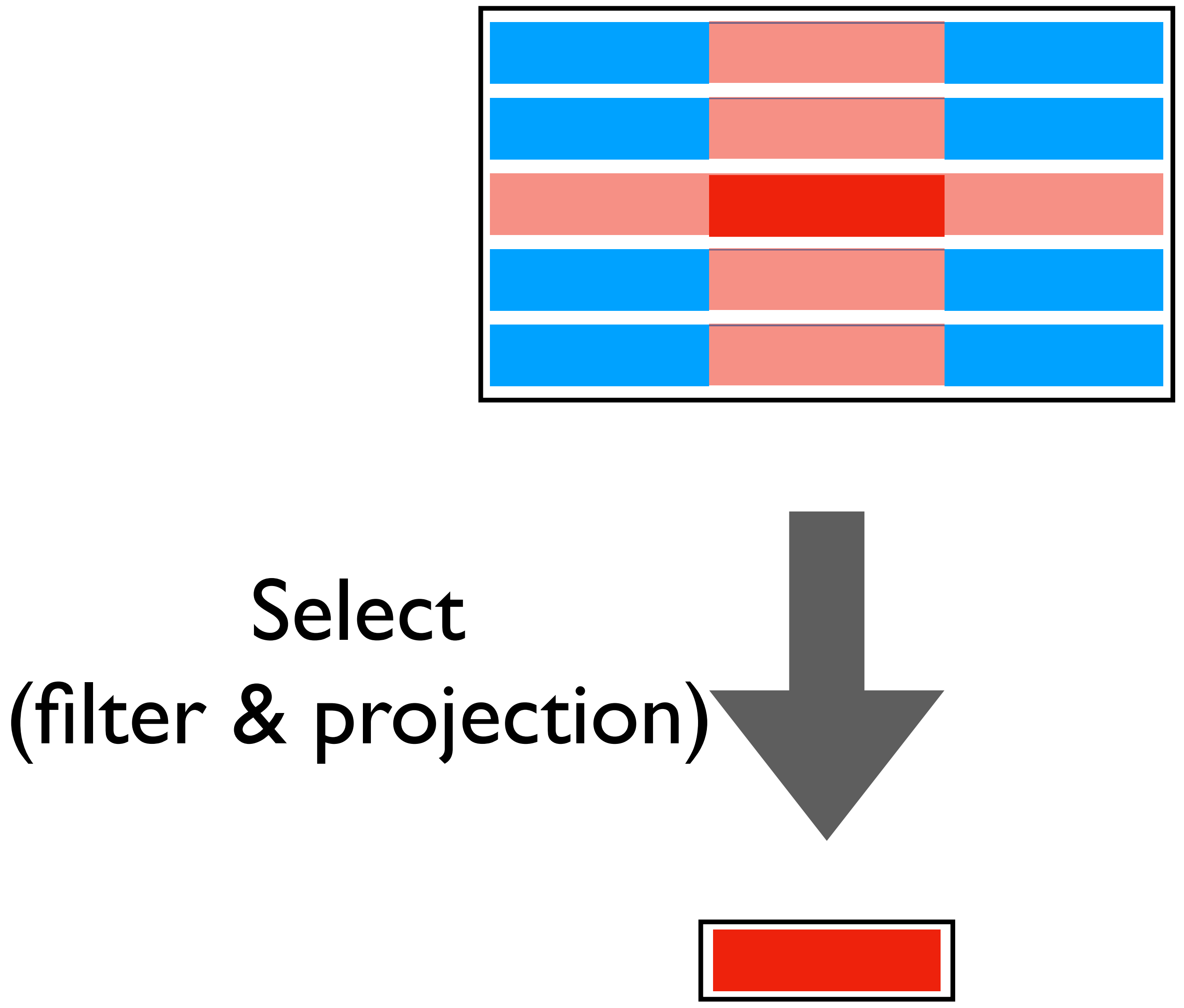
- The language between human and relational databases (tables)

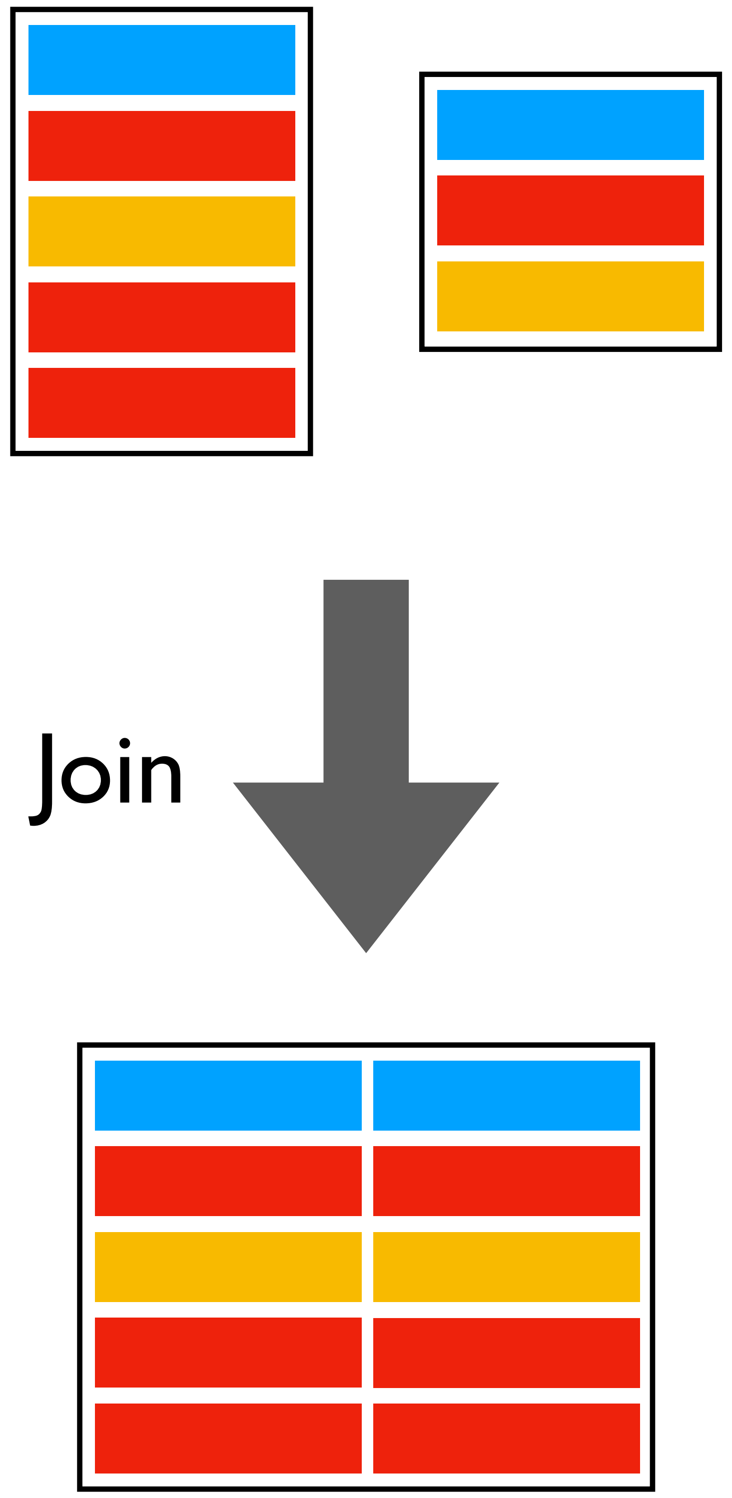


Select val
From T
Where color = red;

Relational Queries

- The language between human and relational databases (tables)



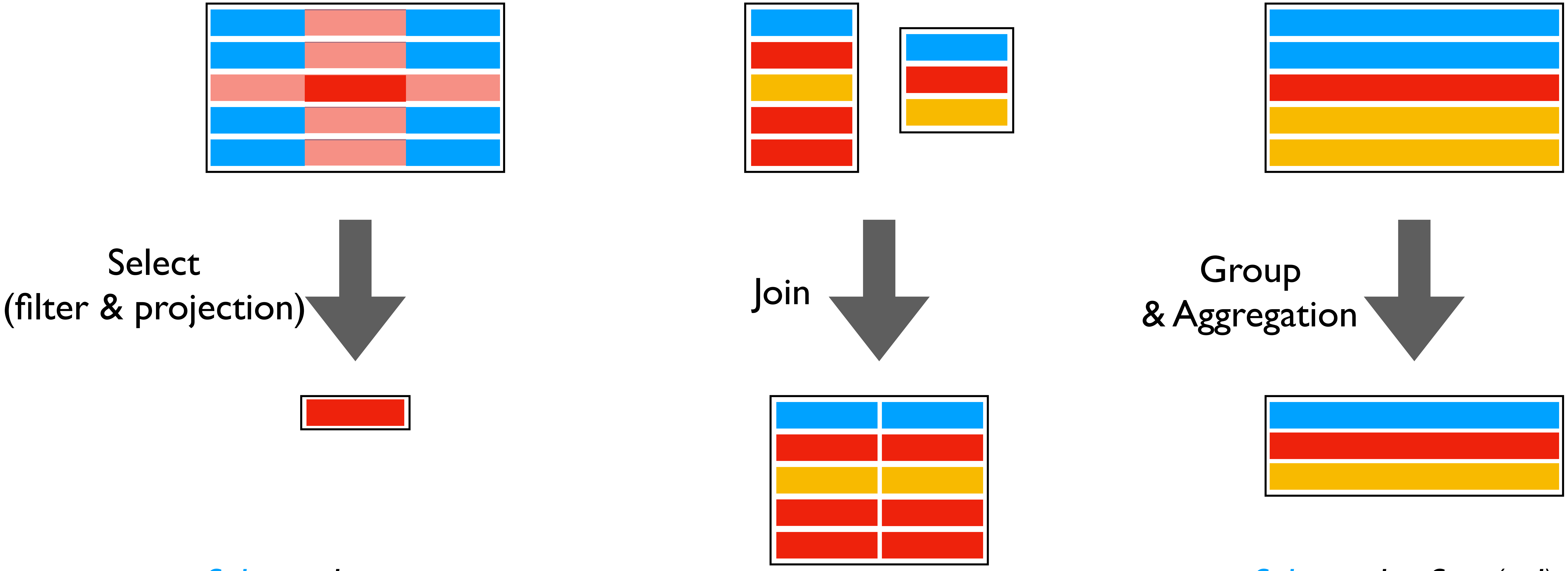


Select val
From T
Where color = red;

T_1 Join T_2 On T_1.color=T_2.color;

Relational Queries

- The language between human and relational databases (tables)

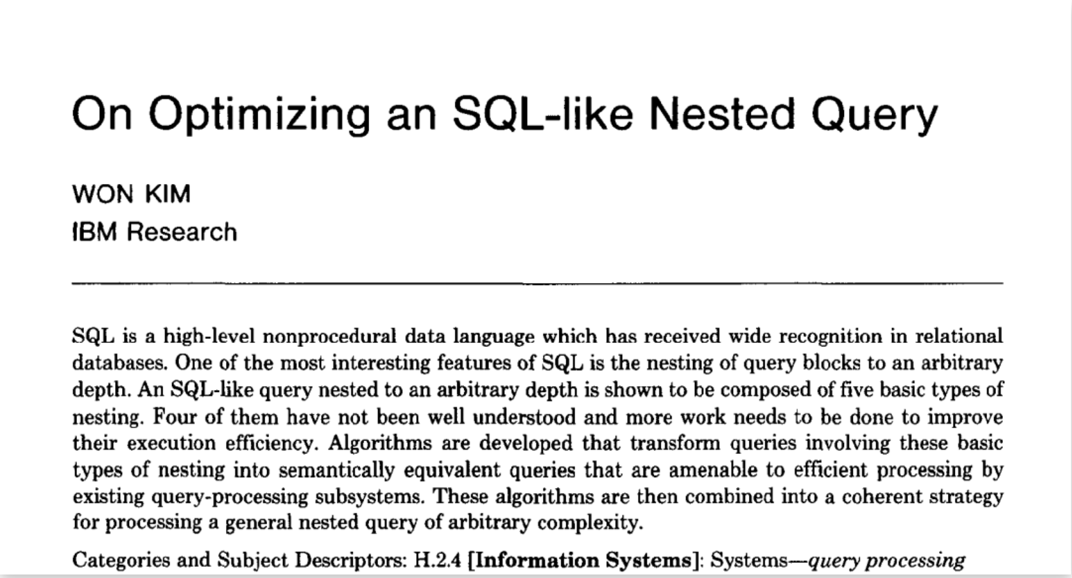


Select val
From T
Where color = red;

T_1 *Join* T_2 *On* T_1*.color=*T_2*.color;*

Select color, Sum(val)
From T
Group by color;

“The Count Bug”


On Optimizing an SQL-like Nested Query

WON KIM
IBM Research

SQL is a high-level nonprocedural data language which has received wide recognition in relational databases. One of the most interesting features of SQL is the nesting of query blocks to an arbitrary depth. An SQL-like query nested to an arbitrary depth is shown to be composed of five basic types of nesting. Four of them have not been well understood and more work needs to be done to improve their execution efficiency. Algorithms are developed that transform queries involving these basic types of nesting into semantically equivalent queries that are amenable to efficient processing by existing query-processing subsystems. These algorithms are then combined into a coherent strategy for processing a general nested query of arbitrary complexity.

Categories and Subject Descriptors: H.2.4 [**Information Systems**]: Systems—*query processing*

1982 “On Optimizing an SQL-like Nested Query” *(Kim Won)*

Rewrite rules for nested queries

q_1

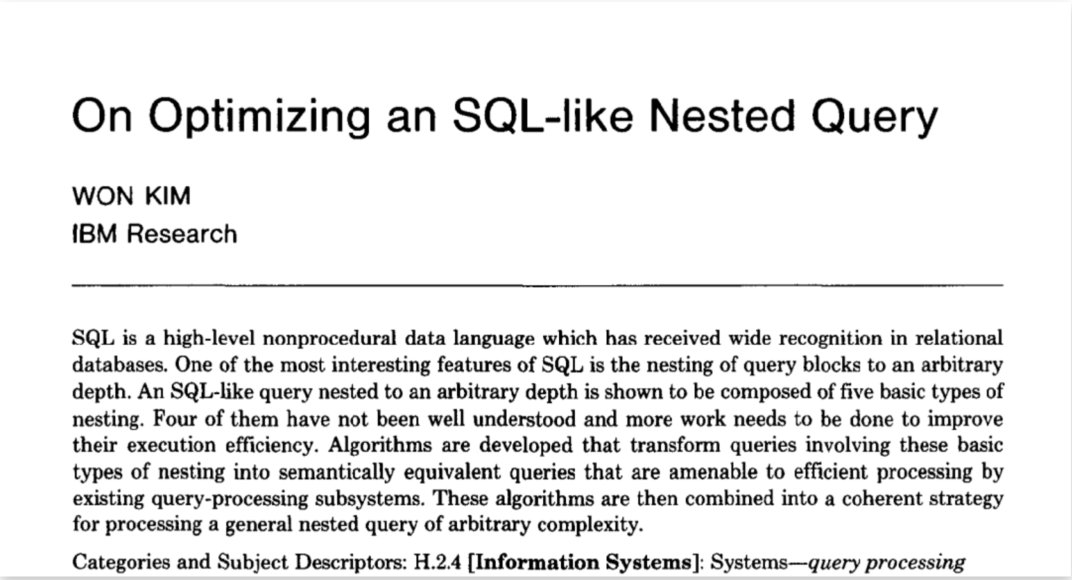
Select $R.c_k$
From R
Where $R.c_h$ = (*Select* **Agg**($S.c_m$)
From S
Where $S.c_n = S.c_p$);

q_2

S' = (*Select* $S.c_n$, **Agg**($S.c_m$)
FROM S
Group By S.cn):

Select $R.c_k$
From R
Where $R.c_h$ = (*Select* **Agg**($S'.c_m$)
From S’
Where $T'.c_n = R.c_p$);

"The Count Bug"



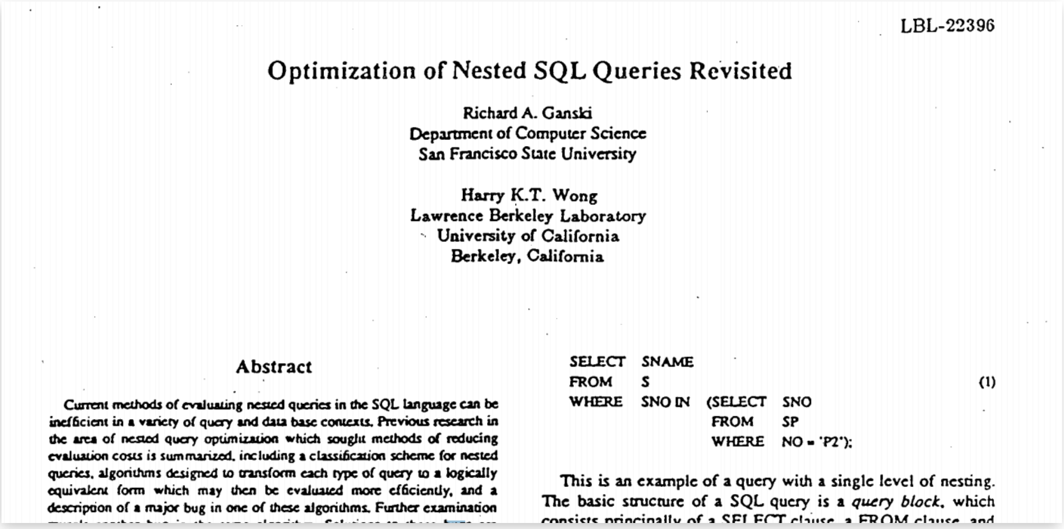
On Optimizing an SQL-like Nested Query

WON KIM
IBM Research

SQL is a high-level nonprocedural data language which has received wide recognition in relational databases. One of the most interesting features of SQL is the nesting of query blocks to an arbitrary depth. An SQL-like query nested to an arbitrary depth is shown to be composed of five basic types of nesting. Four of them have not been well understood and more work needs to be done to improve their execution efficiency. Algorithms are developed that transform queries involving these basic types of nesting into semantically equivalent queries that are amenable to efficient processing by existing query-processing subsystems. These algorithms are then combined into a coherent strategy for processing a general nested query of arbitrary complexity.

Categories and Subject Descriptors: H.2.4 [Information Systems]: Systems—*query processing*

1982 "On Optimizing an SQL-like Nested Query" *(Kim Won)*



LBL-22396

Optimization of Nested SQL Queries Revisited

Richard A. Ganski
Department of Computer Science
San Francisco State University

Harry K.T. Wong
Lawrence Berkeley Laboratory
University of California
Berkeley, California

Abstract

Current methods of evaluating nested queries in the SQL language can be inefficient in a variety of query and data base contexts. Previous research in the area of nested query optimization which sought methods of reducing evaluation costs is summarized, including a classification scheme for nested queries, algorithms designed to transform each type of query to a logically equivalent form which may then be evaluated more efficiently, and a description of a major bug in one of these algorithms. Further examination ...

```
SELECT  SNAME
FROM    S
WHERE   SNO IN  (SELECT  SNO
                 FROM    SP
                 WHERE   NO = 'P2');      (1)
```

This is an example of a query with a single level of nesting. The basic structure of a SQL query is a *query block*, which ...

1987 "Optimization of Nested SQL Queries Revisited" *(Ganski & Wong)*

Rewrite rules for nested queries

q_1

Select R.c_k
From R
*Where R.c_h = (Select **Agg**(S.c_m)*
From S
Where S.c_n = S.c_p);

q_2

*S' = (Select S.c_n, **Agg**(S.c_m)*
FROM S
Group By S.cn):

Select R.c_k
From R
*Where R.c_h = (Select **Agg**(S'.c_m)*
From S'
Where T'.c_n = R.c_p);

Found a bug in the 1982 paper!

PARTS:

PNUM	QOH
3	6
10	1
8	0

SUPPLY:

PNUM	QUAN	SHIPDATE
3	4	7-3-79
3	2	10-1-78
10	1	6-8-78
10	2	8-10-81
8	5	5-7-83

[KIE 84:2]

q_1

PARTS.PNUM
10
8

q_2

PARTS.PNUM
10

Reasoning Tasks

q, q' → **Verification** → $q \equiv q'$

"Are two queries equivalent on ALL inputs"

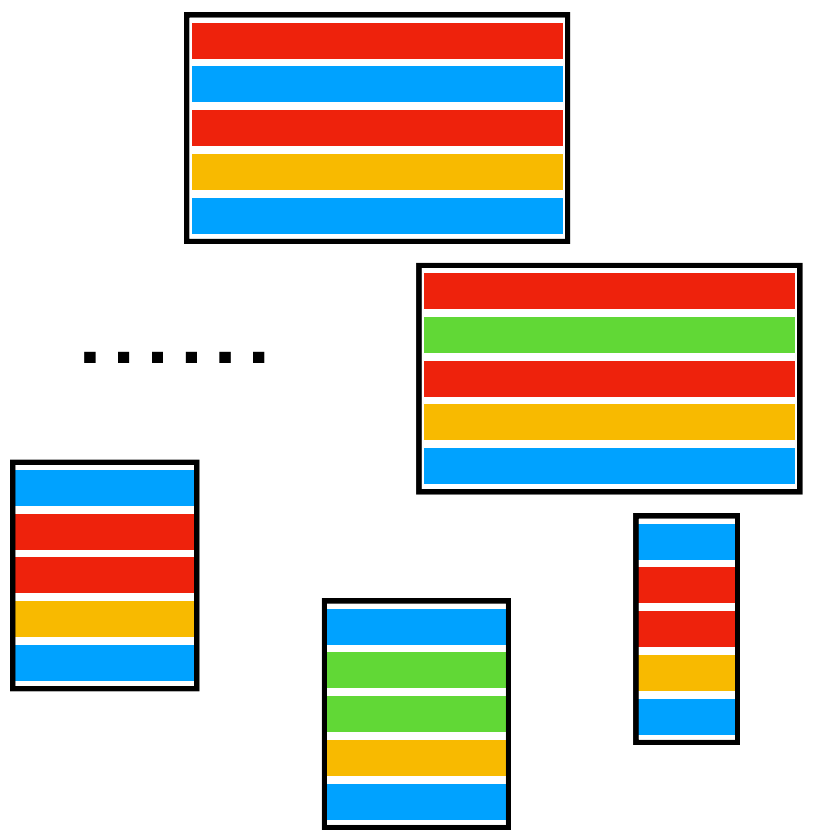
q → **Property Checking (for optimization)** → $q(T) = empty$

"Can the query return empty output on SOME input?"

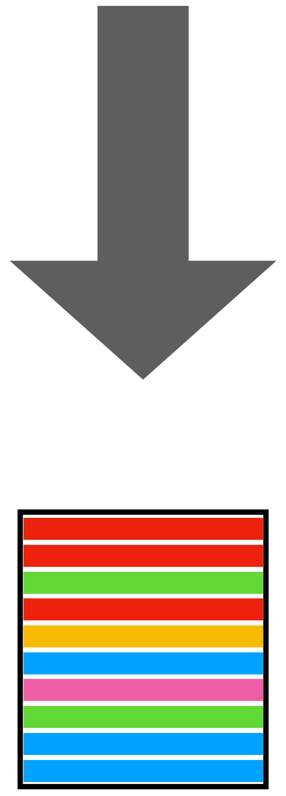
q, q' → **Mutation testing / Grading** → $q(T) \neq q'(T)$

"Find a distinguishing input between queries."

Relational Queries



“analyze transition history”



tens to hundreds of HUGE tables

complex analytical functions

generated by computer

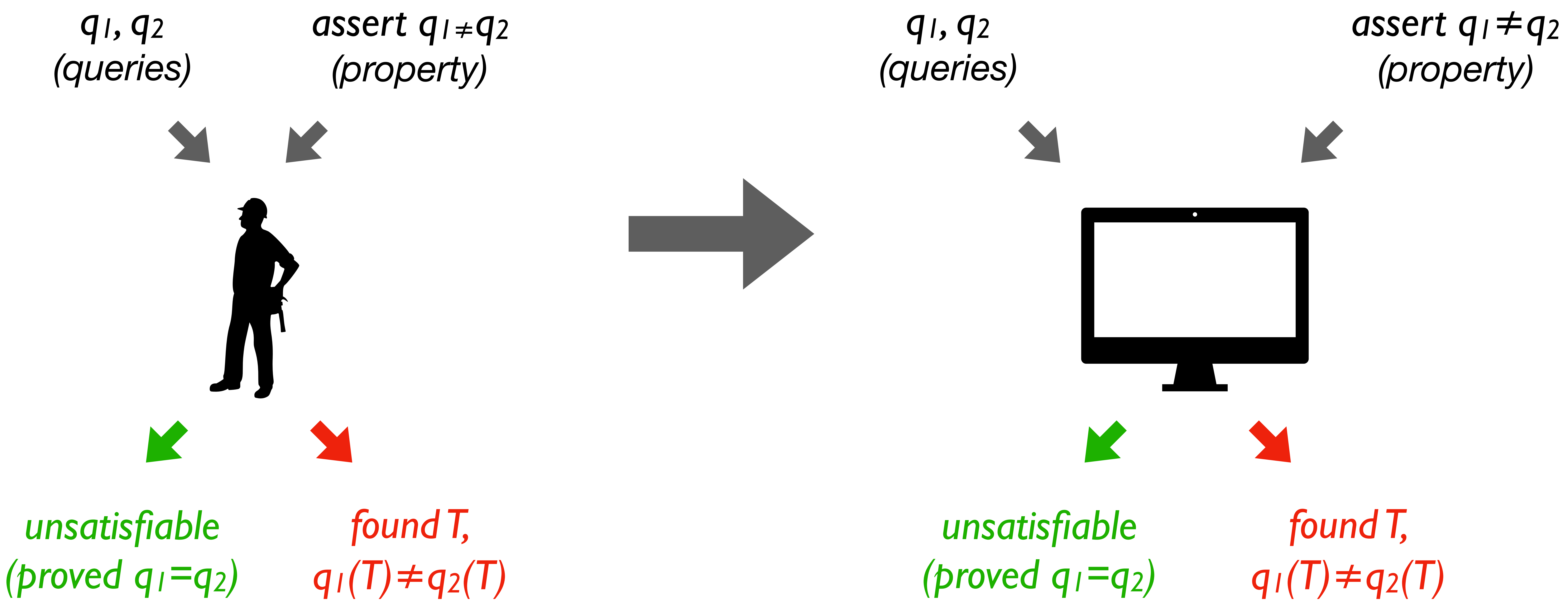
highly optimized

plays important roles in industry

can’t afford 5 years to find a bug!

Automatic Reasoning

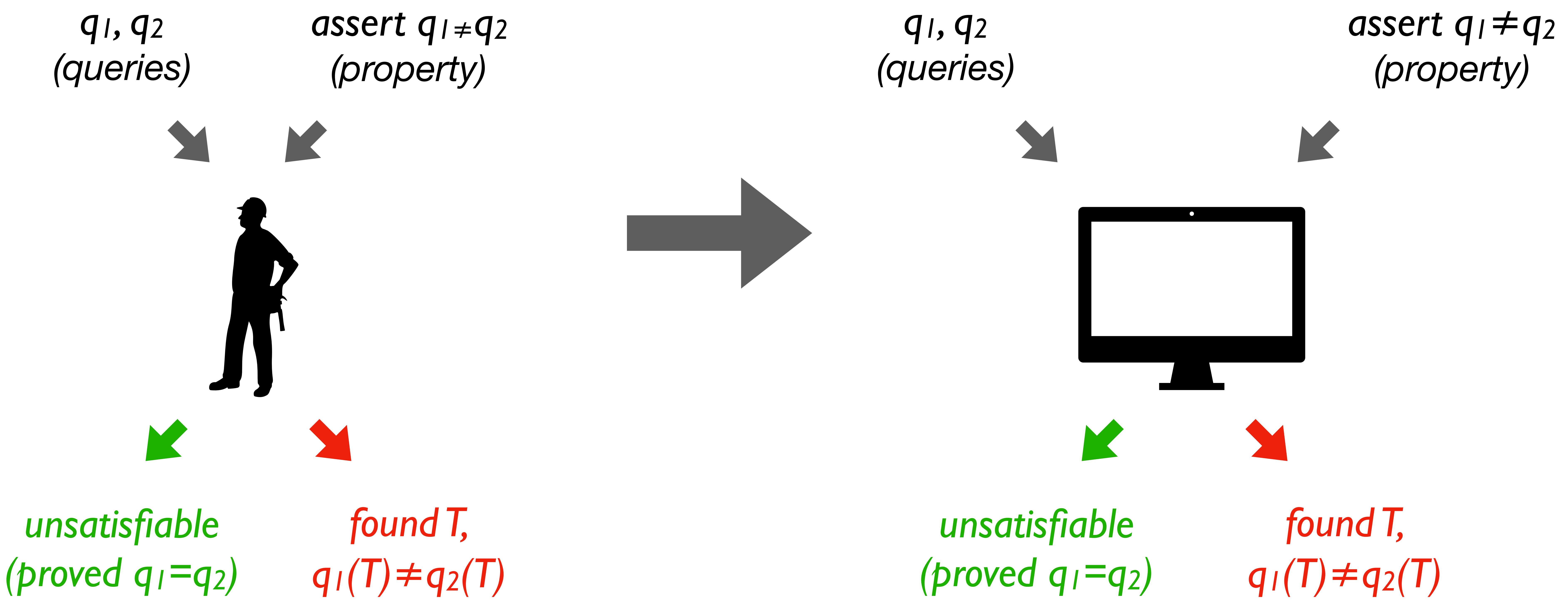
Check whether q_1 is equivalent to q_2 (on ALL inputs)



Automatic Reasoning

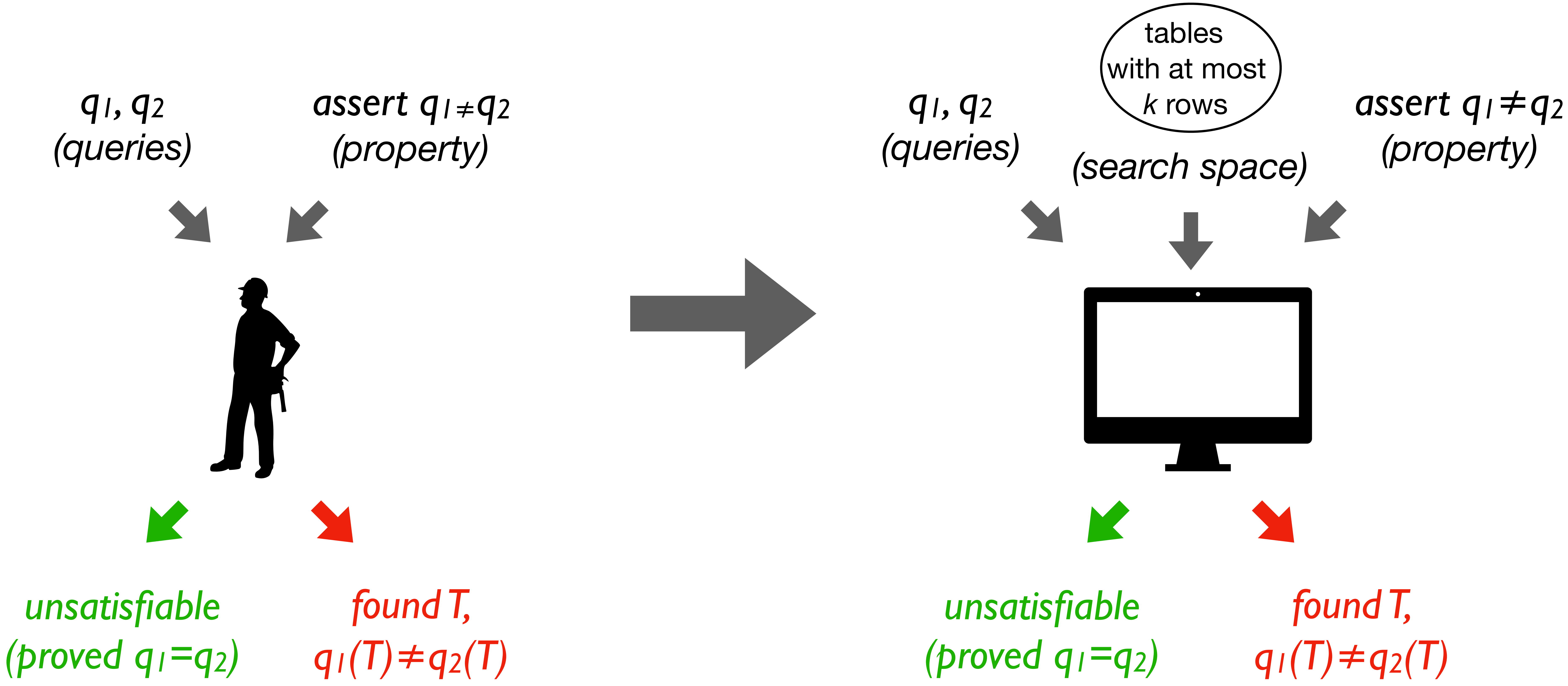
Check whether q_1 is equivalent to q_2 (on ALL inputs)

(unbounded) equivalence is undecidable



Symbolic Reasoning

Check whether q_1 is equivalent to q_2 (on ALL inputs within a search space)



Symbolic Reasoning

"Check whether q_1, q_2 are equivalent on ALL tables with at most k tuples"

(1) *Target queries*

q_1: Select id, val
From T
Where id > 1

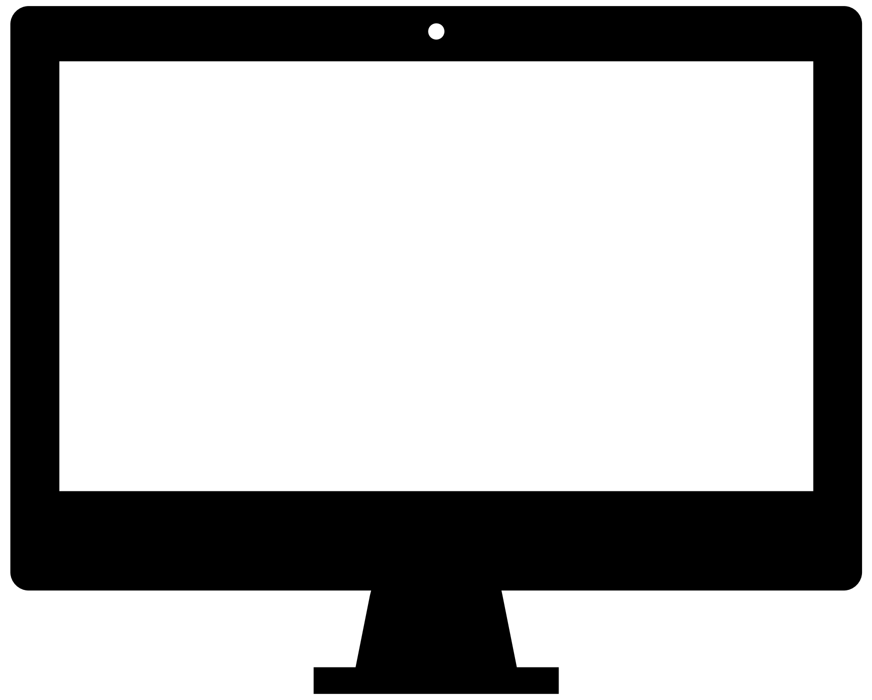
q_2: Select id, val
From T
Where id ≠ 1

(2) *Search space*

tables with at most k rows

(3) *Property*

$$q_1(T) \neq q_2(T)$$



Solver

Symbolic Reasoning

"Check whether q_1, q_2 are equivalent on ALL tables with at most k tuples"

(1) *Target queries*

q_1: Select id, val
From T
Where id > 1

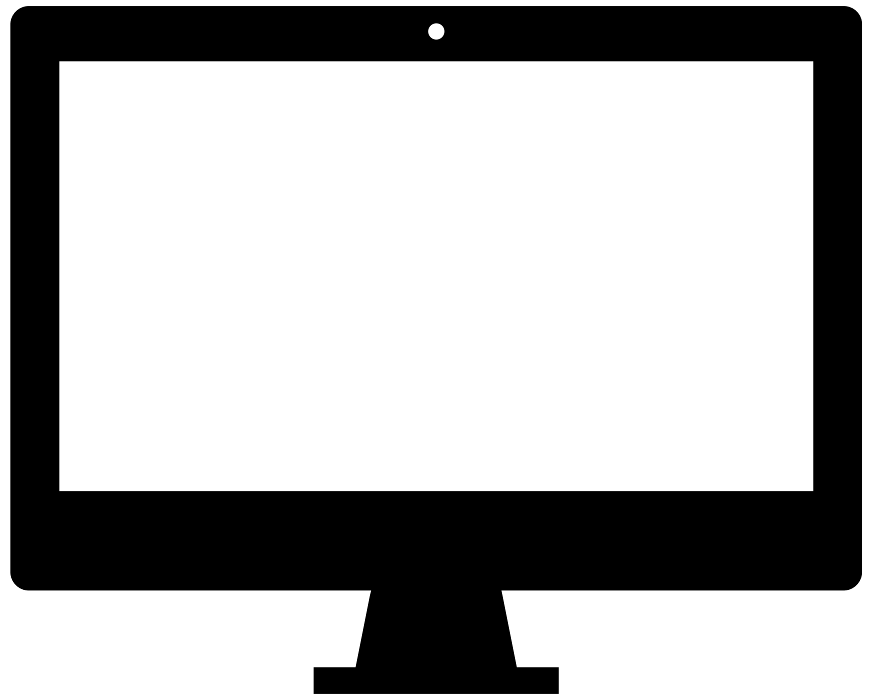
q_2: Select id, val
From T
Where id ≠ 1

(2) *Search space*

tables with at most k rows

(3) *Property*

$$q_1(T) \neq q_2(T)$$



Solver

id	val
0	1
1	2

q_1

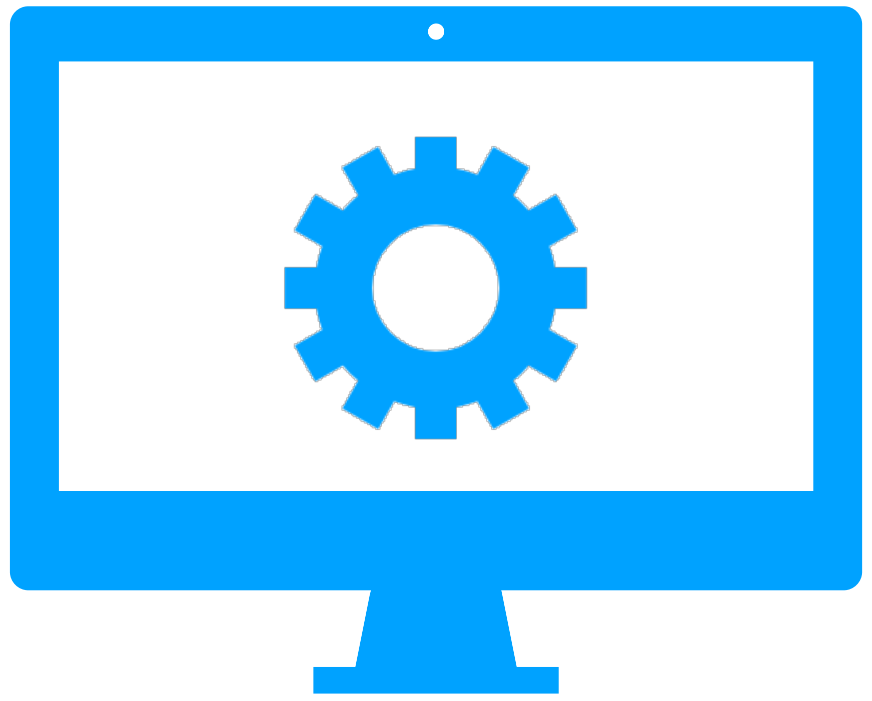
id	val
(empty)	

q_2

id	val
0	1

Symbolic Reasoning

"Check whether q_1, q_2 are equivalent on ALL inputs within size k"



(1) *Target queries*

q_1, q_2

(2) *Search space*

tables with at most k rows

(3) *Property*

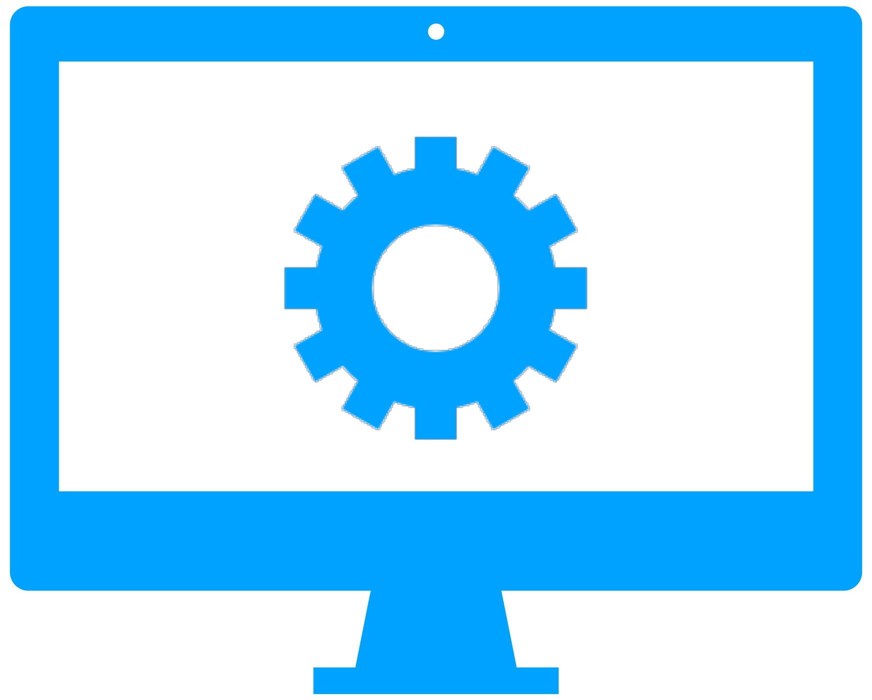
$T_{out1} \neq T_{out2}$

Grouping & aggregation

*" Select f(val)
From T
Group By id "*

Symbolic Reasoning

"Check whether q_1, q_2 are equivalent on ALL inputs within size k"



Solver

(1) *Target queries*

q_1, q_2

(2) *Search space*

tables with at most k rows

(3) *Property*

$T_{out1} \neq T_{out2}$

Grouping & aggregation

" Select f(val)
From T
Group By id "

Exponential ways to partition the table

id	val
x_1	y_1
x_2	y_2
…	
x_k	y_k

id	val
1	y_1
1	y_2
…	
1	y_k

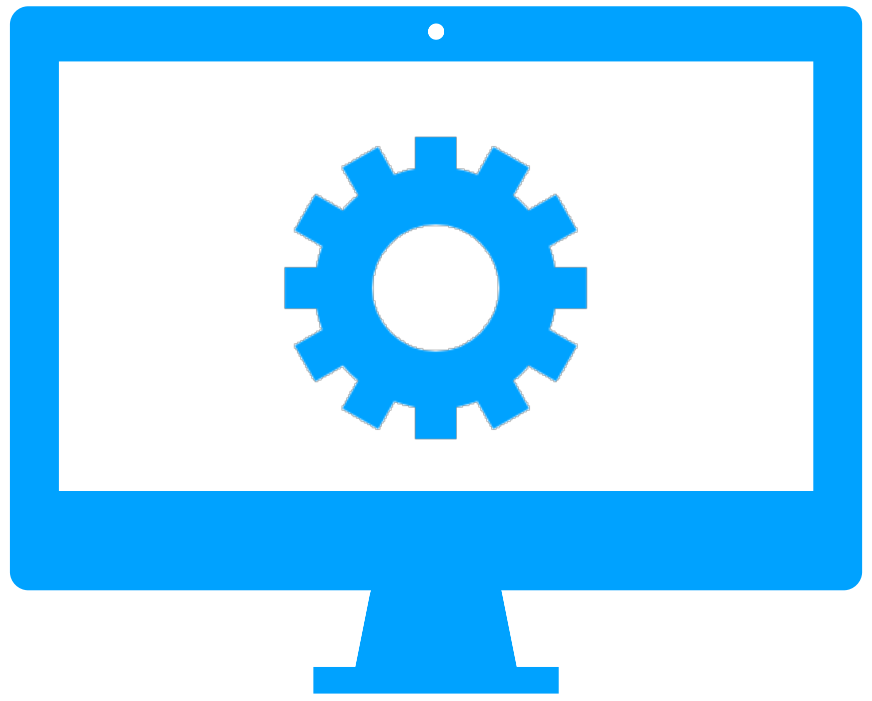
…

id	val
1	y_1
1	y_2
…	
2	y_k

id	val
1	y_1
2	y_2
…	
k	y_k

Symbolic Reasoning

"Check whether q_1, q_2 are equivalent on ALL inputs within size k"



Solver

(1) *Target queries*

q_1, q_2

(2) *Search space*

tables with at most k rows

(3) *Property*

$T_{out1} \neq T_{out2}$

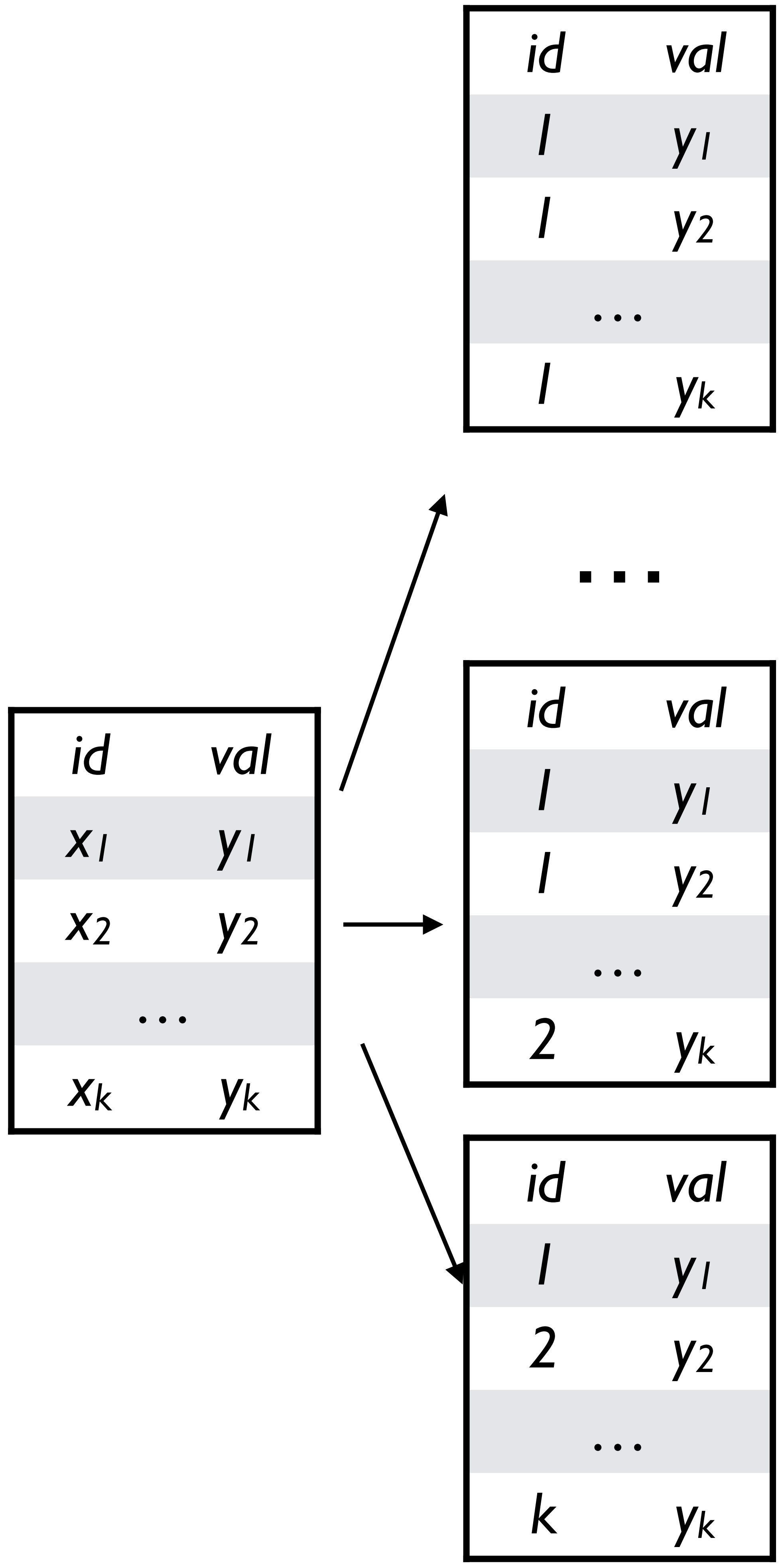
Grouping & aggregation

" Select f(val)
From T
Group By id "

Exponential ways to partition the table

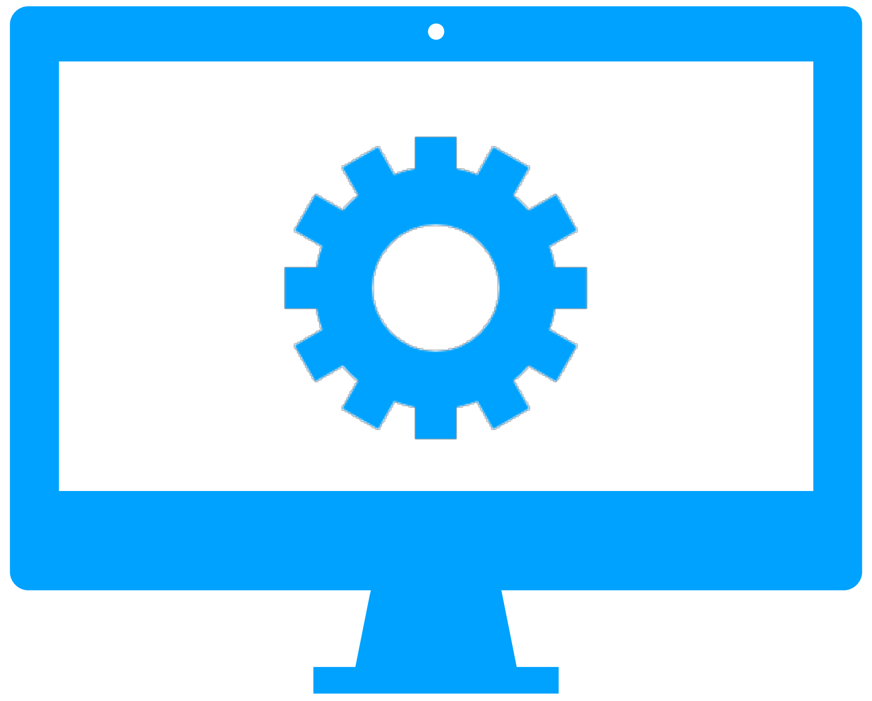
Computationally expensive

$T_{out1} \subset T_{out2}$ & $T_{out2} \subset T_{out1}$



Symbolic Reasoning

"Check whether q_1, q_2 are equivalent on ALL inputs within size k"



Solver

(1) *Target queries*

q_1, q_2

(2) *Search space*

tables with at most k rows

(3) *Property*

$T_{out1} \neq T_{out2}$

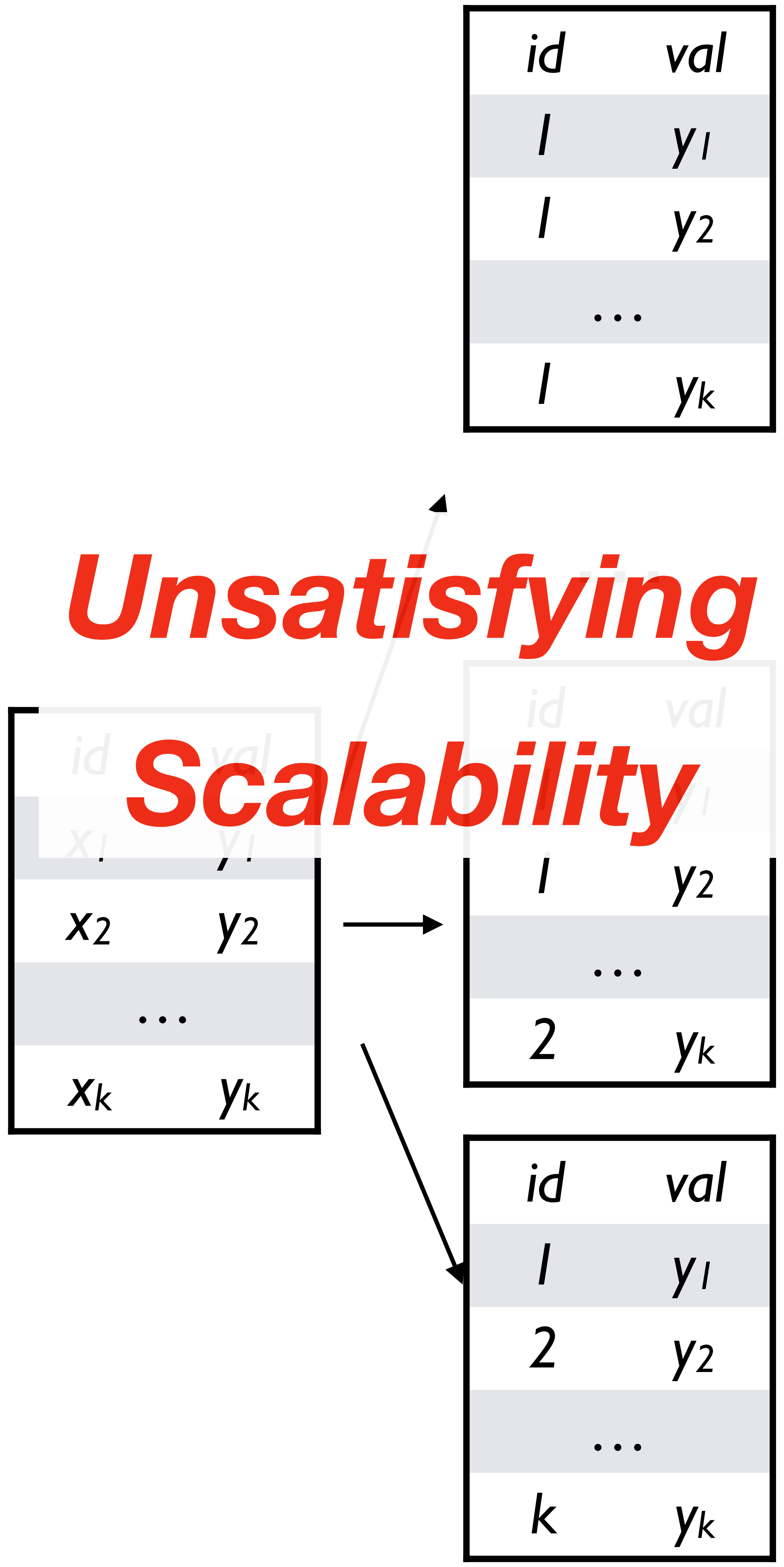
Grouping & aggregation

" Select f(val)
From T
Group By id "

Exponential ways to partition the table

Computationally expensive

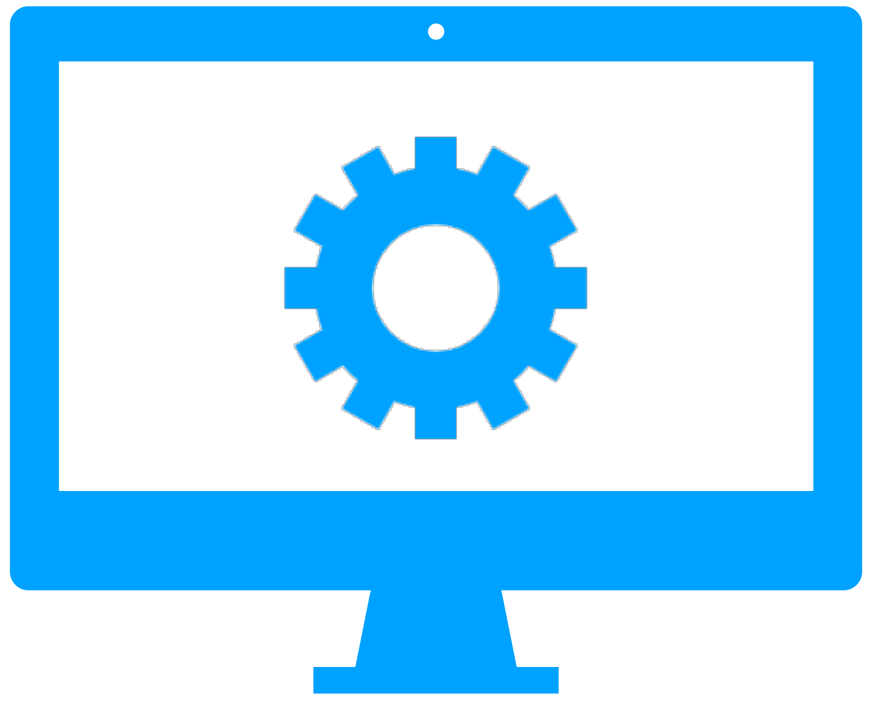
$T_{out1} \subset T_{out2}$ & $T_{out2} \subset T_{out1}$



Unsatisfying Scalability

Symbolic Reasoning

"Check whether q_1, q_2 are equivalent on ALL inputs within size k"



Solver

(1) Target queries

q_1, q_2 → *Grouping & aggregation*

" Select f(val)
From T
Group By id "

(2) Search space

tables with at most k rows → *Exponential ways to partition the table*

"Small Model"
A smaller search space to achieve same reasoning guarantee

(3) Property

$T_{out1} \neq T_{out2}$ → *Computationally expensive*

$T_{out1} \subset T_{out2}$ & $T_{out2} \subset T_{out1}$

Space Refinement

"Check whether q_1, q_2 are equivalent on ALL inputs within size k"

q_1, q_2
(queries)

$T_{out1} \neq T_{out2}$
(property)

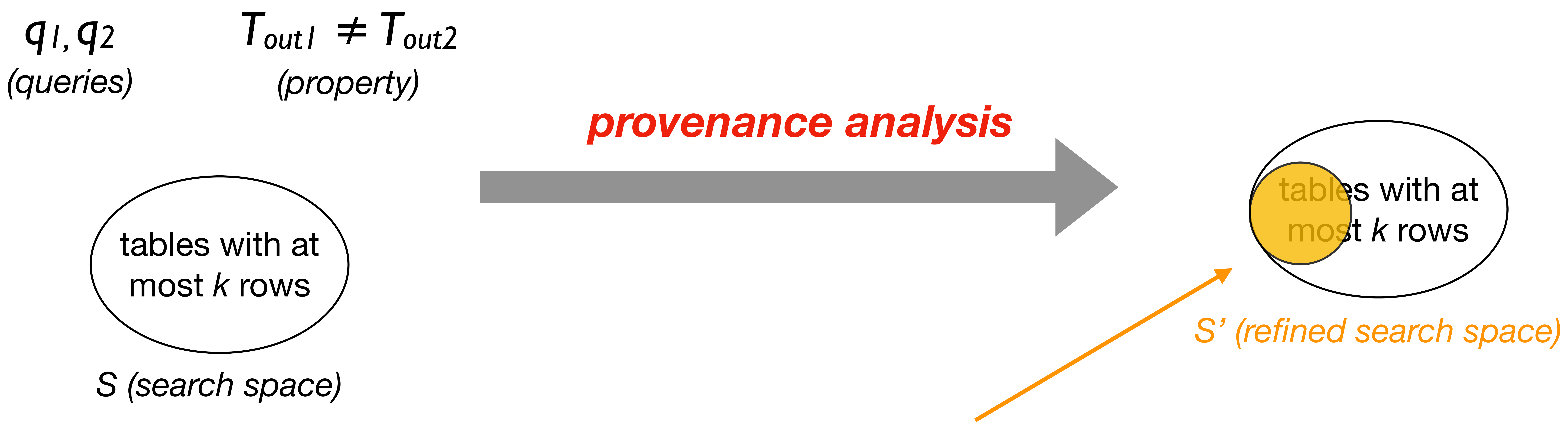
"Small Model"

(1) If exists T ∊ S satisfying the property, we can find one in the S' too.

(2) If none of tables in S' satisfying the property, then no T exists in S too.

Space Refinement

“Check whether q_1, q_2 are equivalent on ALL inputs within size k”



“Small Model”

(1) If exists T ∊ S satisfying the property, we can find one in the S’ too.

(2) If none of tables in S’ satisfying the property, then no T exists in S too.

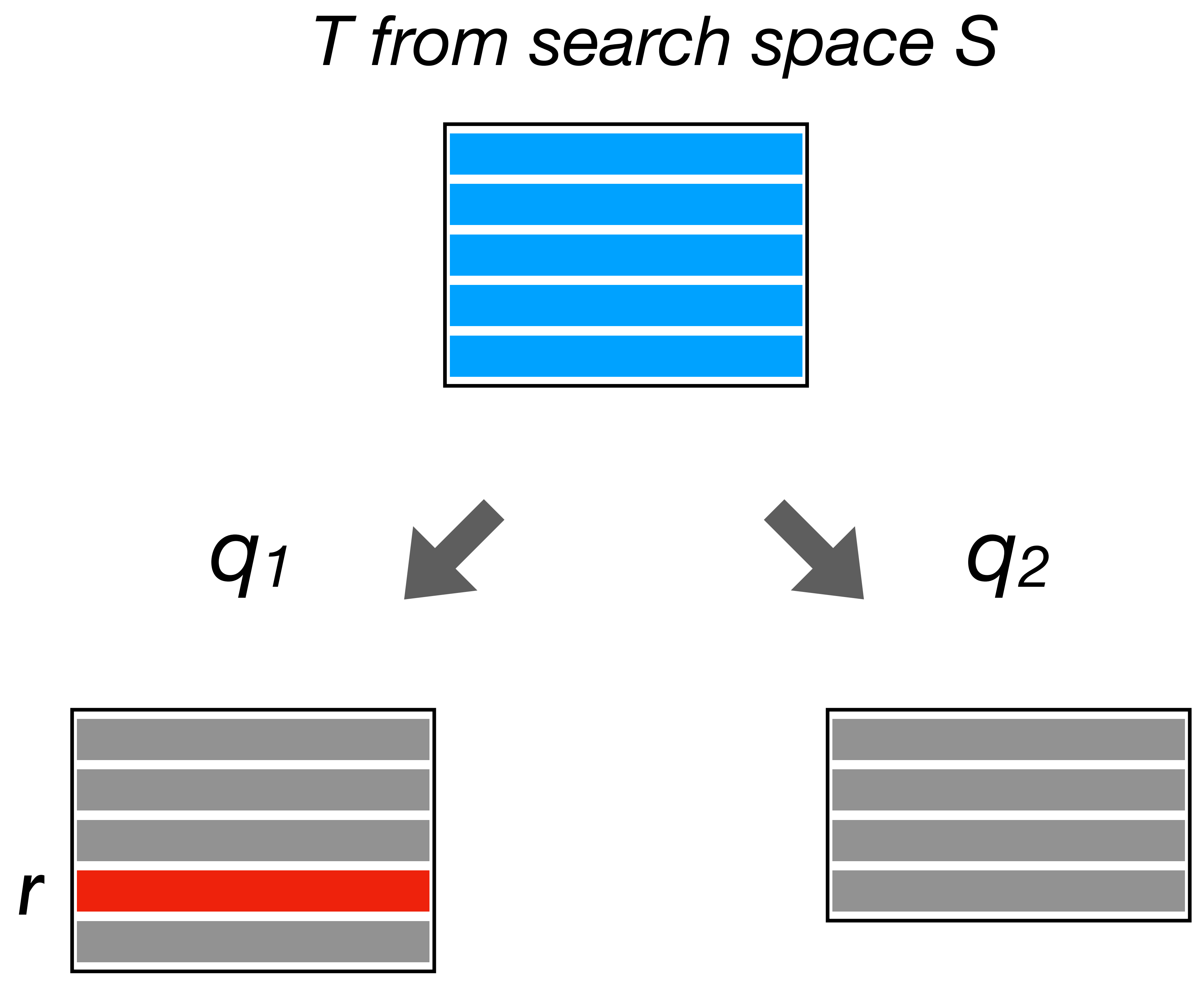
Insight from Property

- Many properties requires only ***one*** tuple in the output to invalidate.

"Check whether q_1, q_2 are equivalent"

$$q_1(T) \neq q_2(T)$$

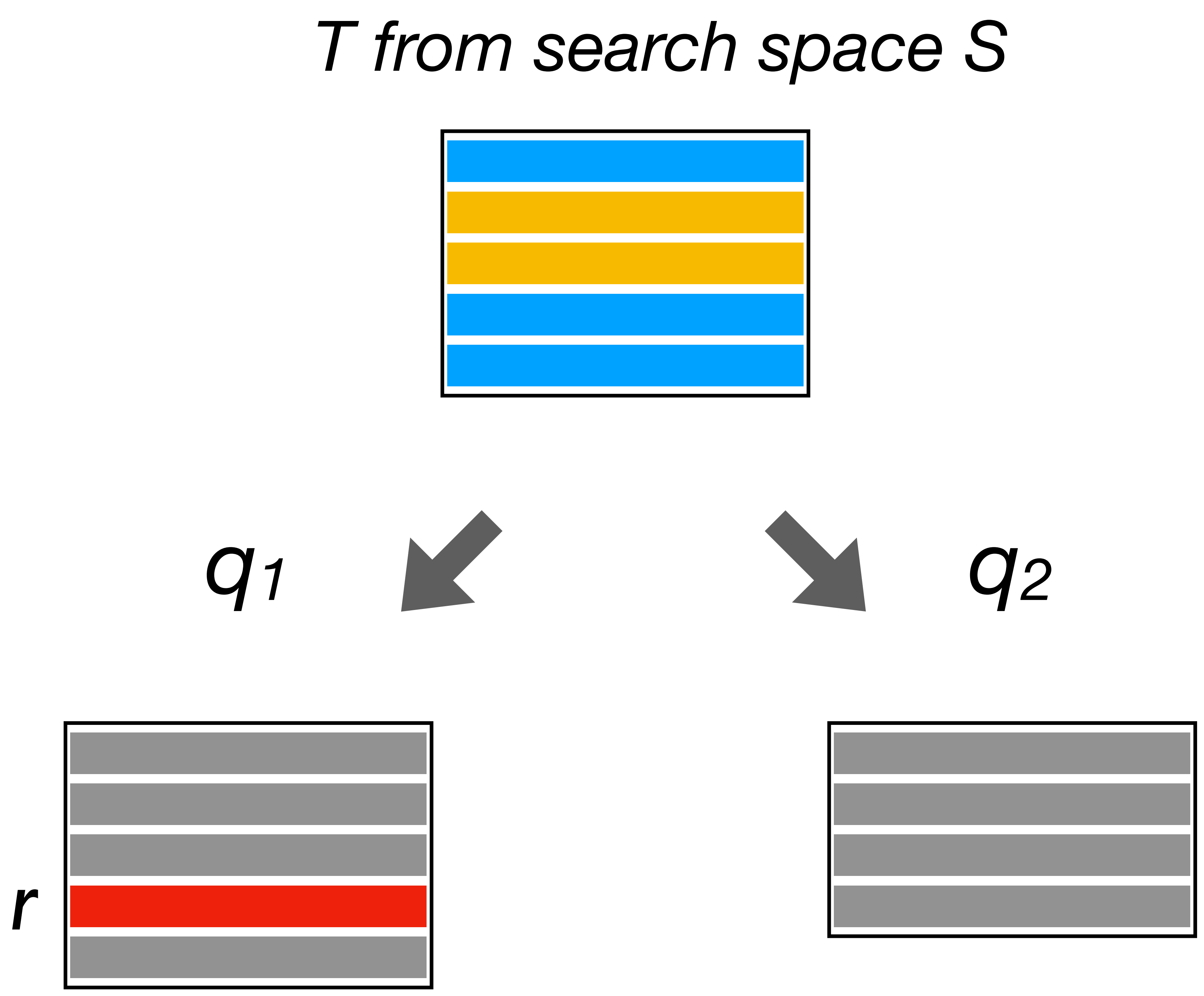
Exists a row r with different multiplicities in T_{out1} and T_{out2}



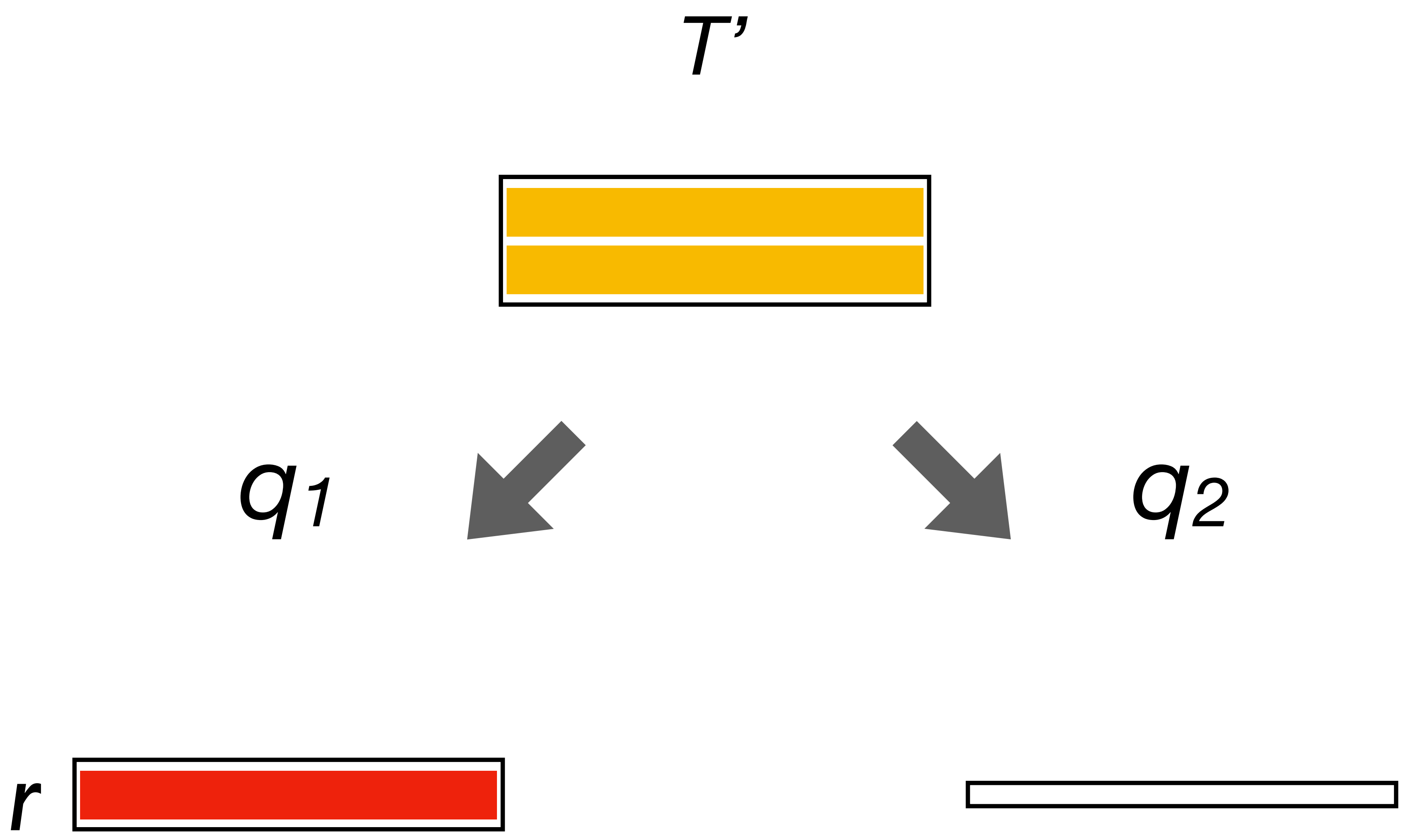
$$r \in T_{out1},\ r \notin T_{out2} \rightarrow q_1(T) \neq q_2(T)$$

Insight from the Property

- Many important properties requires only ***one*** tuple in the output to be invalidated.



$r \in T_{out1}, r \notin T_{out2}$



$r \in T_{out1}, r \notin T_{out2}$

T' can also distinguish q1 from q2!

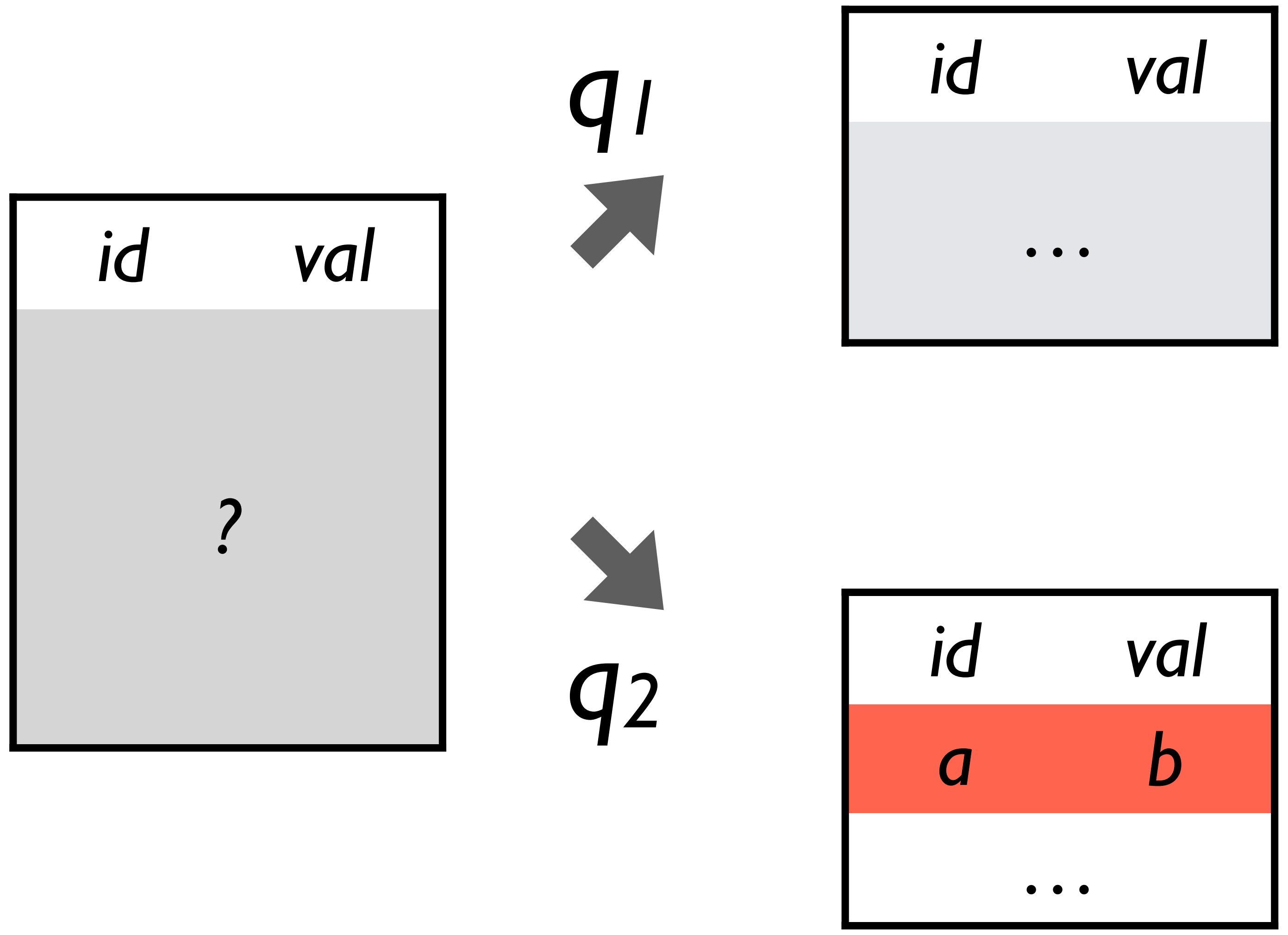
Provenance Analysis

Assume r=(a, b) is the output tuple showing the difference between two queries

$r \in T_{out1}, r \notin T_{out2}$

q_1: Select id, max(val)
From T
Group By id

q_2: Select id, min(val)
From T
Group By id



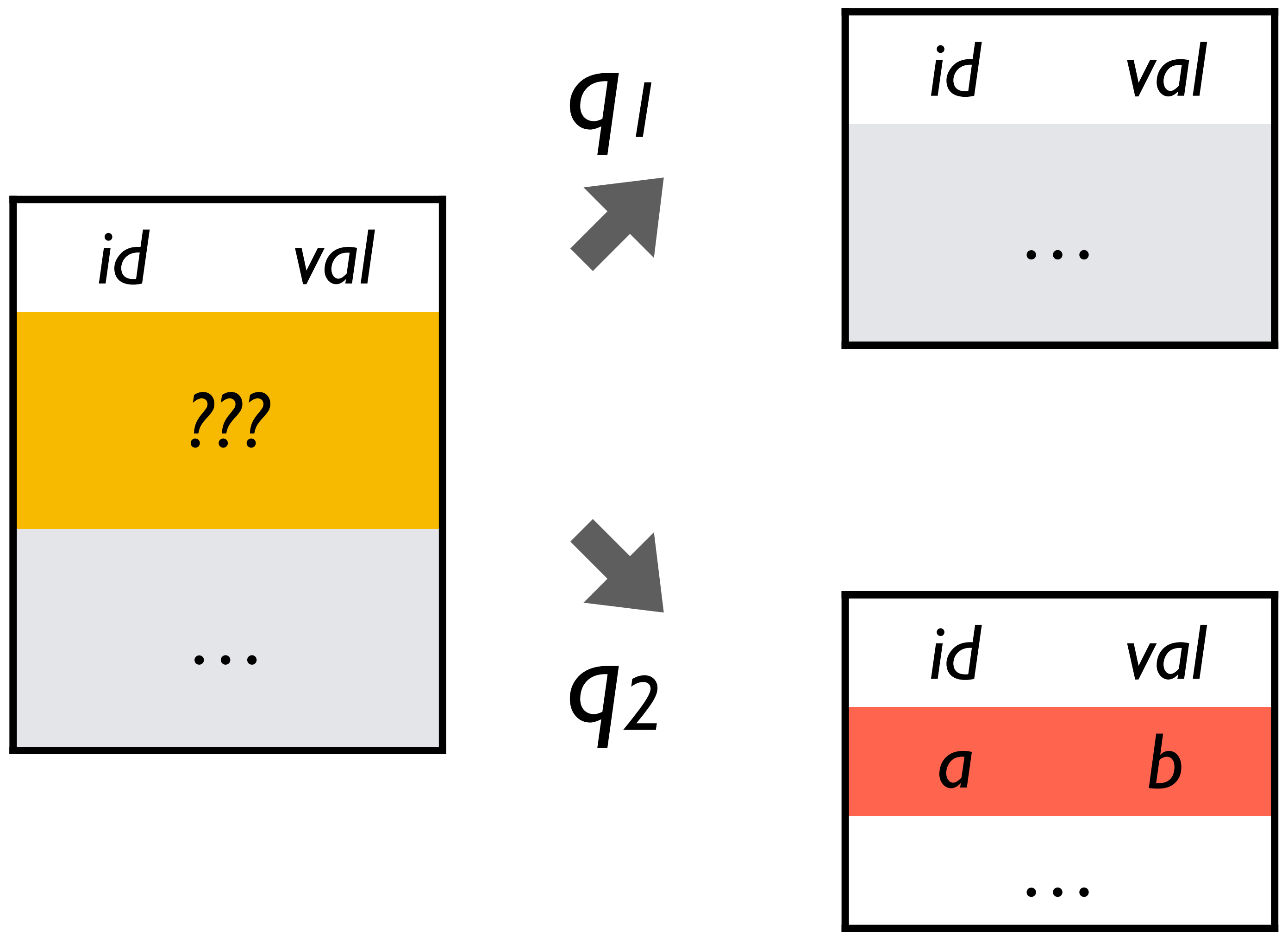
Provenance Analysis

Assume r=(a, b) is the output tuple showing the difference between two queries

$r \in T_{out1}, r \notin T_{out2}$

q_1:
```
Select id, max(val)
From T
Group By id
```

q_2:
```
Select id, min(val)
From T
Group By id
```



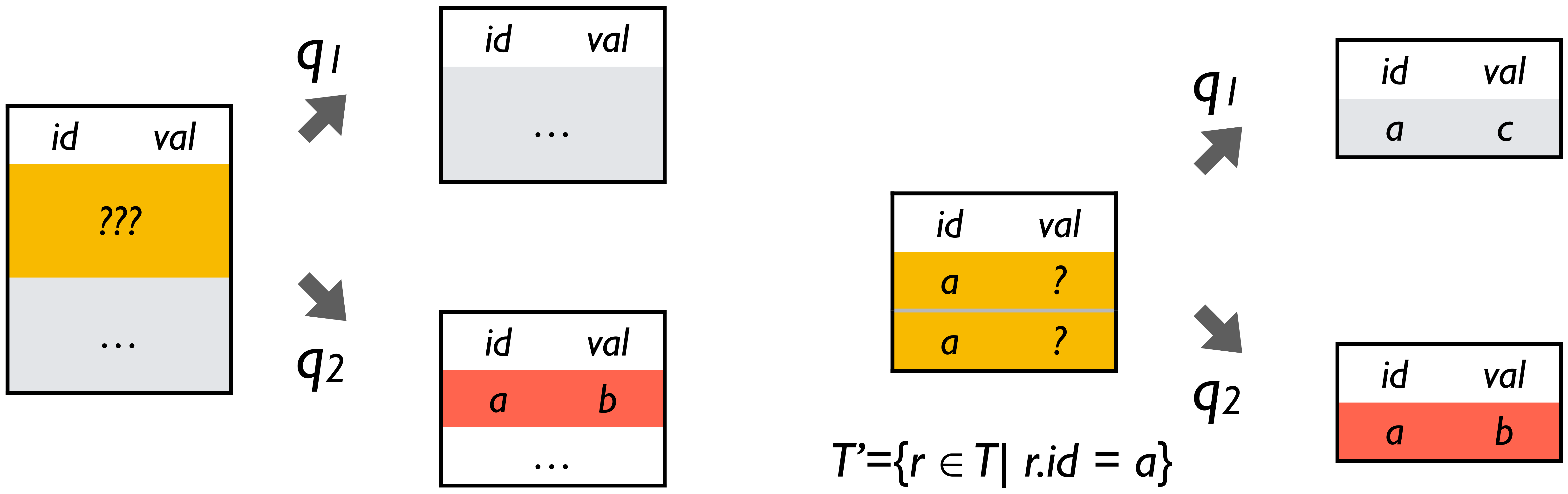
Provenance Analysis

Assume $r=(a, b)$ *is the output tuple showing the difference between two queries*

$r \in T_{out1}, r \notin T_{out2}$

q_1: Select id, max(val)
From T
Group By id

q_2: Select id, min(val)
From T
Group By id



Provenance Analysis

Assume r=(a, b) is the output tuple showing the difference between two queries

$r \in T_{out1}, r \notin T_{out2}$

q_1: Select id, max(val)
From T
Group By id

q_2: Select id, min(val)
From T
Group By id

id	val
???	
…	

q_1

id	val
…	

q_2

id	val
a	b
…	

id	val
a	?
a	?

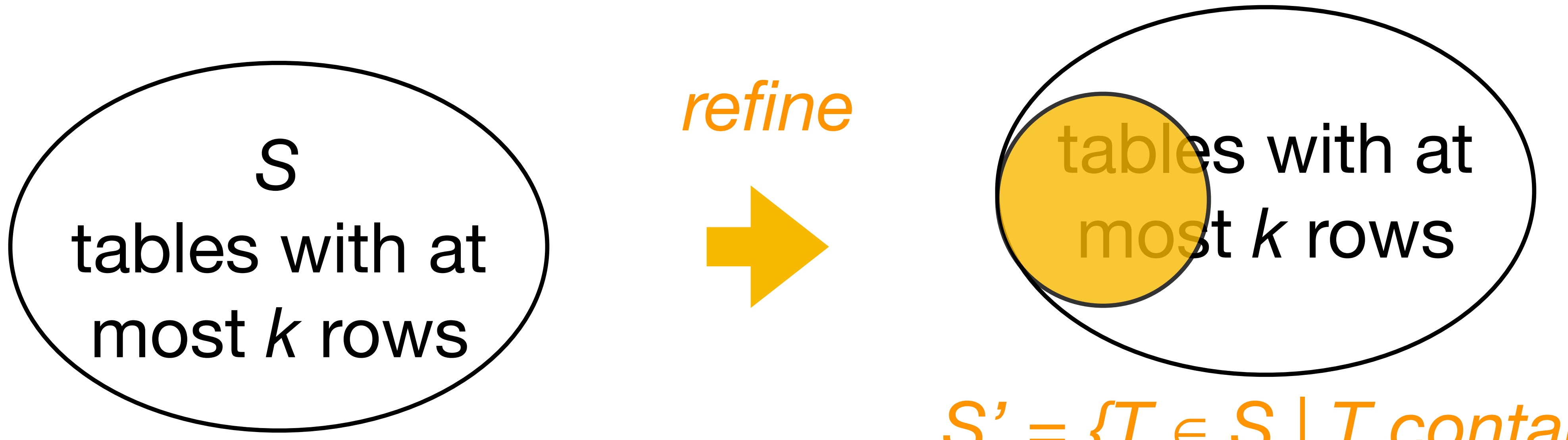
T'={r ∈ T| r.id = a}

q_1

id	val
a	c

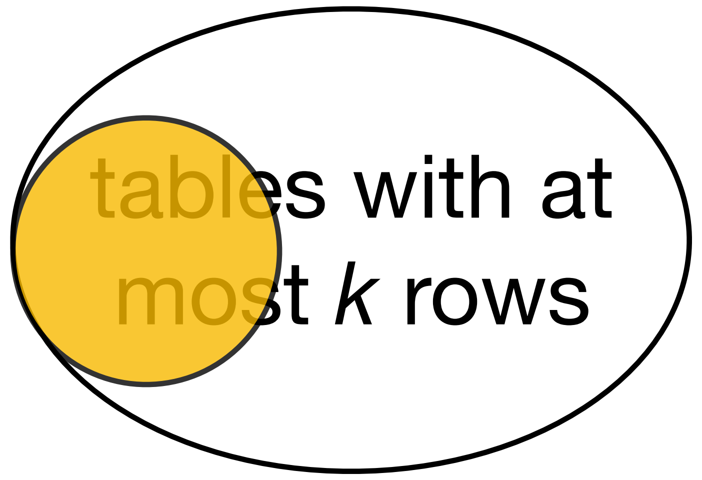
q_2

id	val
a	b



S' = {T ∈ S | T contain only one group}
(the group with id "a")

Space Refinement



$S' = \{T \in S \mid T \text{ contain only one group}\}$

q_1: Select id, max(val)
From T
Group By id

q_2: Select id, min(val)
From T
Group By id

id	val
0	1
0	2
1	3
1	3

q_1

id	val
0	2
1	3

q_2

id	val
0	1
1	3

id	val
0	1
0	2

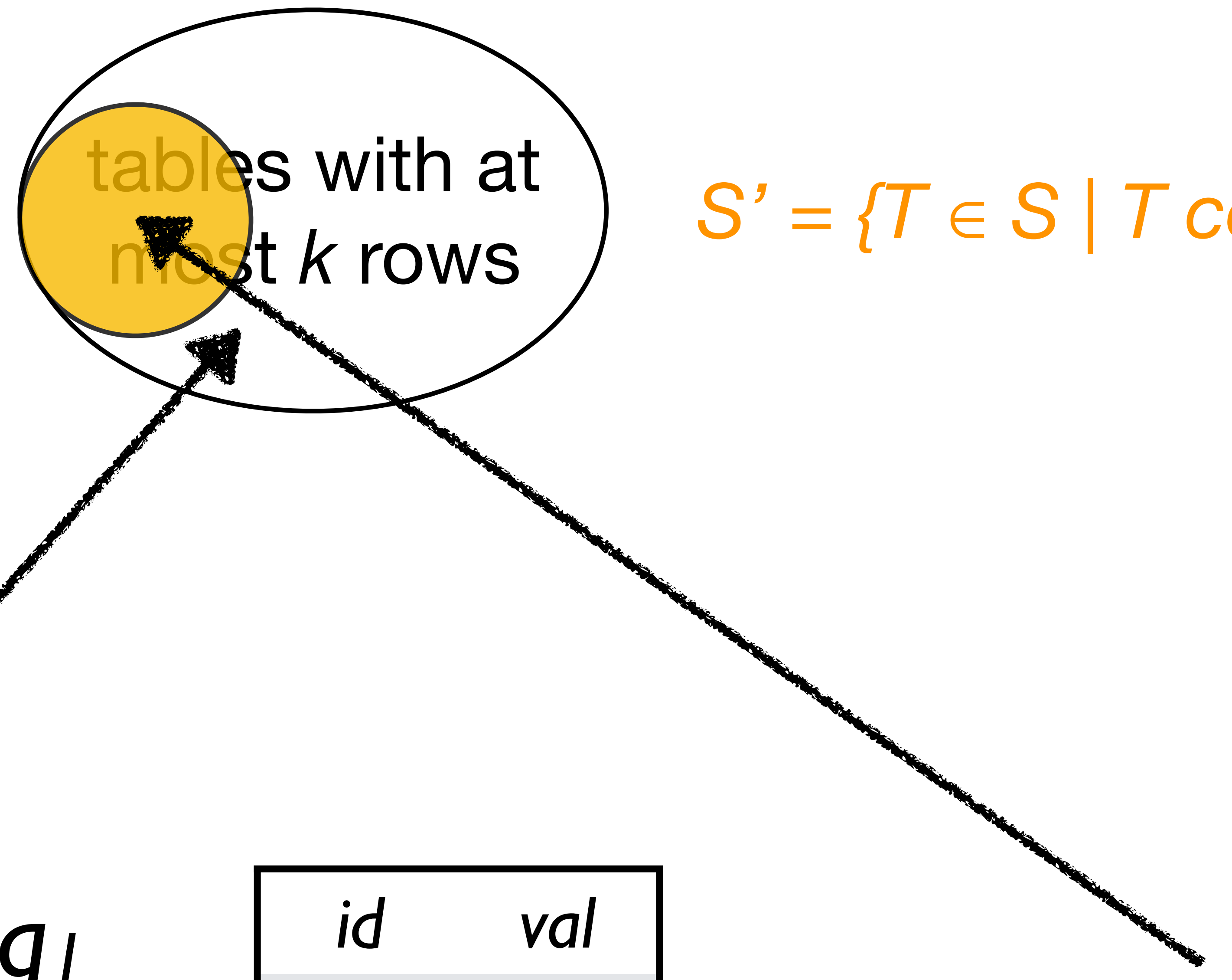
q_1

id	val
0	2

q_2

id	val
0	1

Space Refinement



S' = {T ∈ S | T contain only one group}

q_1: *Select id, max(val)*
From T
Group By id

q_2: *Select id, min(val)*
From T
Group By id

id	val
0	1
0	2
1	3
1	3

q_1 ↗

id	val
0	2
1	3

q_2 ↘

id	val
0	1
1	3

id	val
0	1
0	2

q_1 ↗

id	val
0	2

q_2 ↘

id	val
0	1

Symbolic Provenance Analysis

$$\boxed{(q \sim \phi) \rightsquigarrow (q_1 \sim \phi_1) \land \ldots}$$

$$\frac{q = T}{(q \sim \phi) \rightsquigarrow (T \sim \phi)}\ \text{(Table)} \qquad \frac{q = \mathsf{Select}(q_1, f) \quad \mathsf{schema}(q_1) = \bar{c}}{(q \sim \phi) \rightsquigarrow \left(q_1 \sim \phi \land \left[c_i \mapsto t.i\right] f\right)}\ \text{(Select)}$$

$$\frac{q = \mathsf{Distinct}(q_1)}{(q \sim \phi) \rightsquigarrow (q_1 \sim \phi)}\ \text{(Distinct)} \qquad \frac{q = \mathsf{Proj}(\bar{v}, q_1) \quad \mathsf{schema}(q_1) = \bar{c}}{(q \sim \phi) \rightsquigarrow \left(q_1 \sim \left[t.i \mapsto \left(\left[c_j \mapsto t.j\right] v_i\right)\right] \phi\right)}$$

$$\frac{\begin{array}{c} q = \mathsf{Join}(q_1, q_2) \quad \phi = \phi_1 \land \phi_2 \land \phi_3 \quad |\mathsf{schema}(q_1)| = n_1 \\ \mathsf{colRef}(\phi_i) \cap \mathsf{schema}(q_i) = \emptyset\ _{(i=1,2)} \end{array}}{(q \sim \phi) \rightsquigarrow (q_1 \sim \phi_1) \land (q_2 \sim [t.i \mapsto t.(i - n_1)]\phi_2)}\ \text{(Join)}$$

$$\frac{q = \mathsf{Union}(q_1, q_2)}{(q \sim \phi) \rightsquigarrow (q_1 \sim \phi) \land (q_2 \sim \phi)}\ \text{(Union)} \qquad \frac{q = \mathsf{Rename}(q_1, name, \bar{c})}{(q \sim \phi) \rightsquigarrow (q_1 \sim \phi_1)}\ \text{(Rename)}$$

$$\frac{q = \mathsf{LeftJoin}(q_1, q_2, f) \quad \phi = \phi_1 \land \phi_2 \land \phi_3 \quad \mathsf{colRef}(\phi_i) \cap \mathsf{schema}(q_i) = \emptyset\ _{(i=1,2)}}{(q \sim \phi) \rightsquigarrow (q_1 \sim \phi_1) \land (q_2 \sim \mathsf{true})}\ \text{(LeftJoin)}$$

$$\frac{\begin{array}{c} q = \mathsf{Aggr}(\bar{c}, \bar{\alpha}, \bar{c}_t, q_1) \\ \phi = \phi_1 \land \phi_2 \quad \mathsf{colRef}(\phi_1) \subseteq \{\bar{c}\} \end{array}}{(q \sim \phi) \rightsquigarrow (q_1 \sim \phi_1)}\ \text{(Aggr)} \qquad \frac{(q_i \sim \phi_i) \rightsquigarrow \bigwedge_k \left(q'_{ik} \sim \phi'_{ik}\right)}{\bigwedge_i (q_i \sim \phi_i) \rightsquigarrow \bigwedge_{i,k} \left(q'_{ik} \sim \phi'_{ik}\right)}\ \text{(Step)}$$

- *Inductively define the analysis rules for different operators*
- *How to combine provenance from multiple queries*

S → S’

If exists T ∊ S satisfying the property, we can find one in the S’ too.

Analysis complexity: linear to the query size.

Experiment

Bounded Verification

“Verify two queries are equivalent on ALL inputs.”

$$q_1(T) \equiv q_2(T)$$

Benchmarks: 46 rules from Apache Calcite

Test generation

“Can the query return empty output?”

$$q(T) = empty$$

“Find a distinguishing input between queries.”

$$q_1(T) \neq q_2(T)$$

Benchmarks: 15 student submissions & prior work

Experiment

Bounded Verification

"Verify two queries are equivalent on ALL inputs."

$$q_1(T) \equiv q_2(T)$$

Benchmarks: 46 rules from Apache Calcite

Test generation

"Can the query return empty output?"

$$q(T) = empty$$

"Find a distinguishing input between queries."

$$q_1(T) \neq q_2(T)$$

Benchmarks: 15 student submissions & prior work

Process

Measure solving speed with and without space refinement

1. Increase search space size until hitting 10 minutes limit without refinement

2. Re-run the same search space with space refinement

Experiment

Bounded Verification

"Verify two queries are equivalent on ALL inputs."

$$q_1(T) \equiv q_2(T)$$

Benchmarks: 46 rules from Apache Calcite

Test generation

"Can the query return empty output?"

$$q(T) = empty$$

"Find a distinguishing input between queries."

$$q_1(T) \neq q_2(T)$$

Benchmarks: 15 student submissions & prior work

Process

Measure solving speed with and without space refinement

1. Increase search space size until hitting 10 minutes limit without refinement

2. Re-run the same search space with space refinement

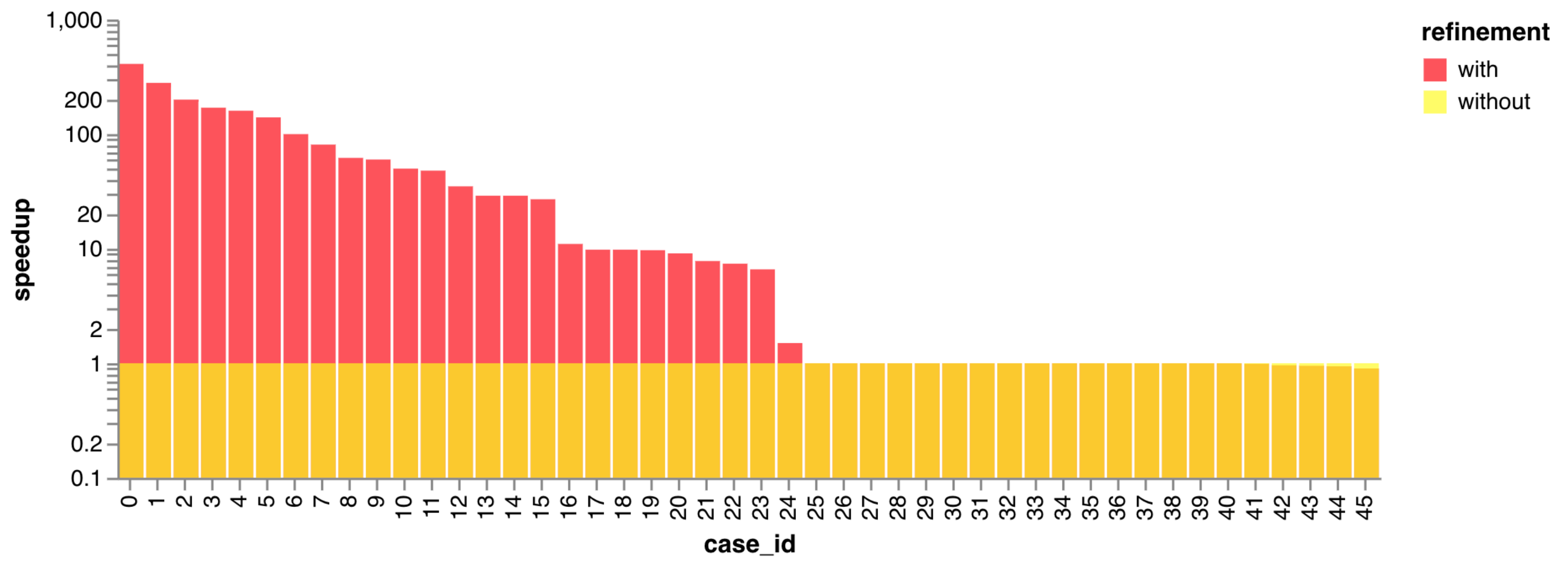
Cosette SQL Solver
Qex SQL Solver

Experiment — Verification

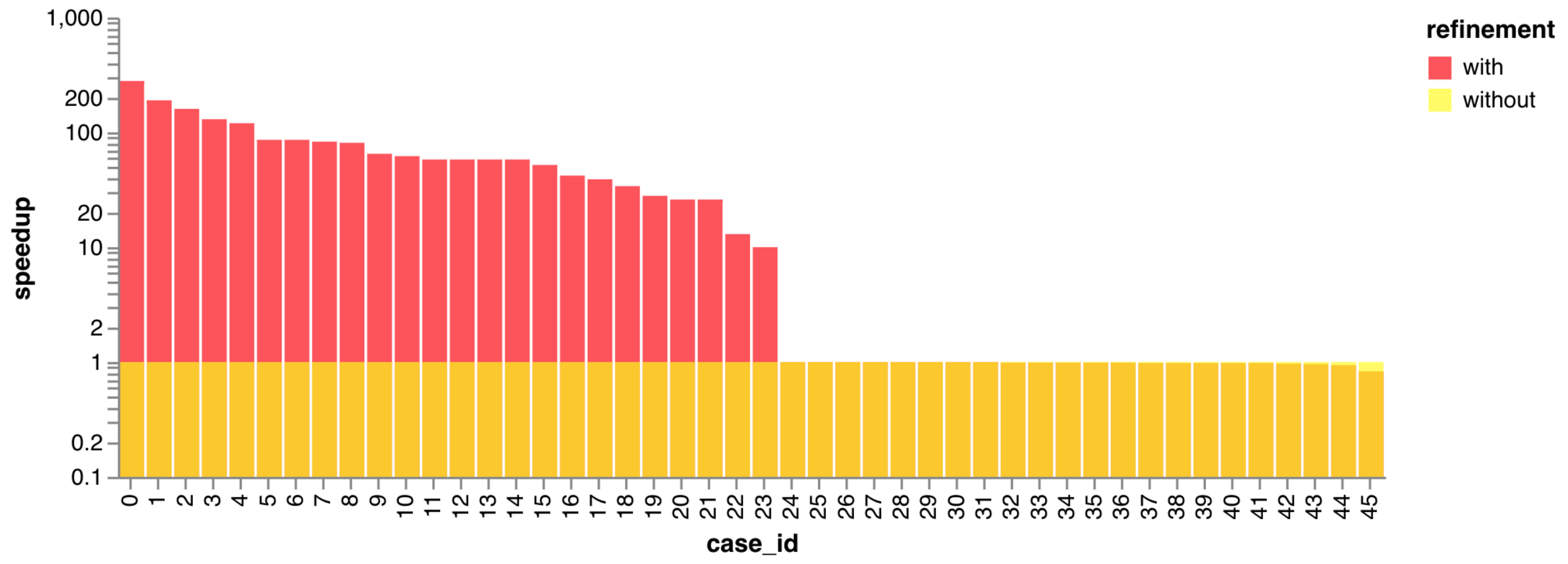
Bounded Verification

"Verify that two queries are equivalent on ALL inputs with no more than k tuples."

Result



Cosette *with and without refinement*



Qex *with and without refinement*

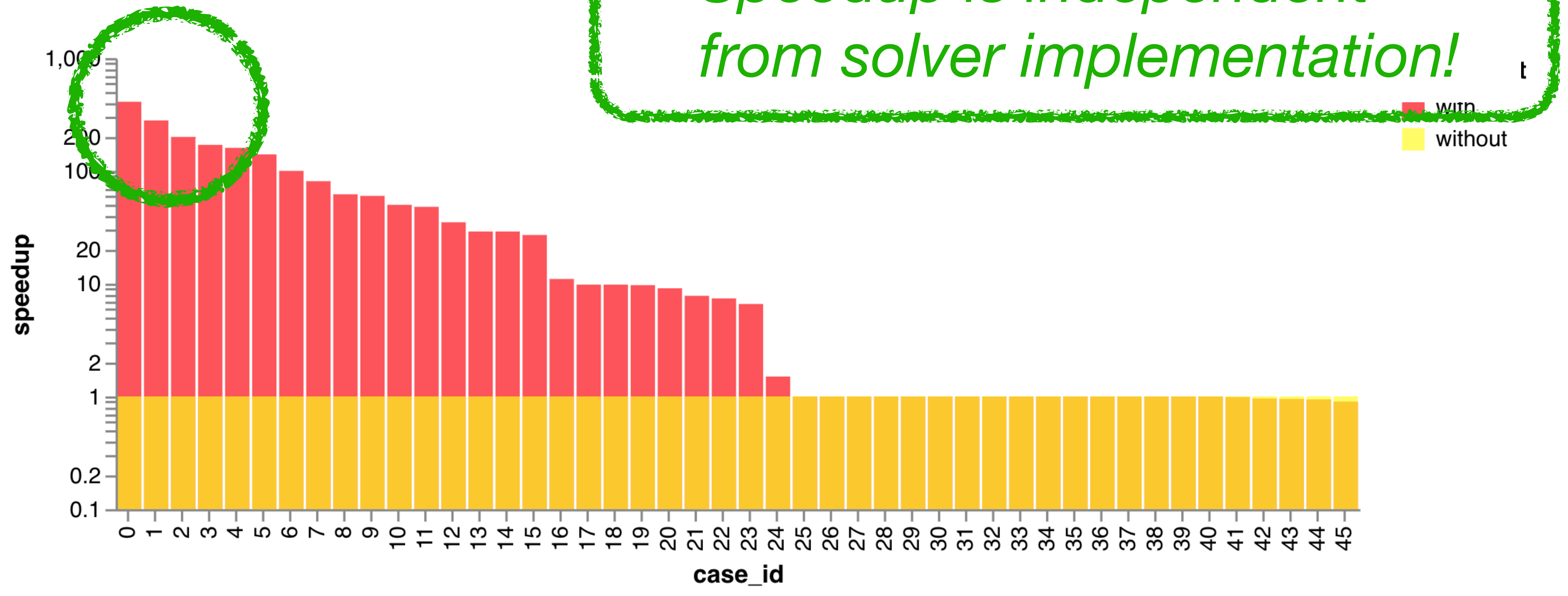
Experiment — Verification

Bounded Verification

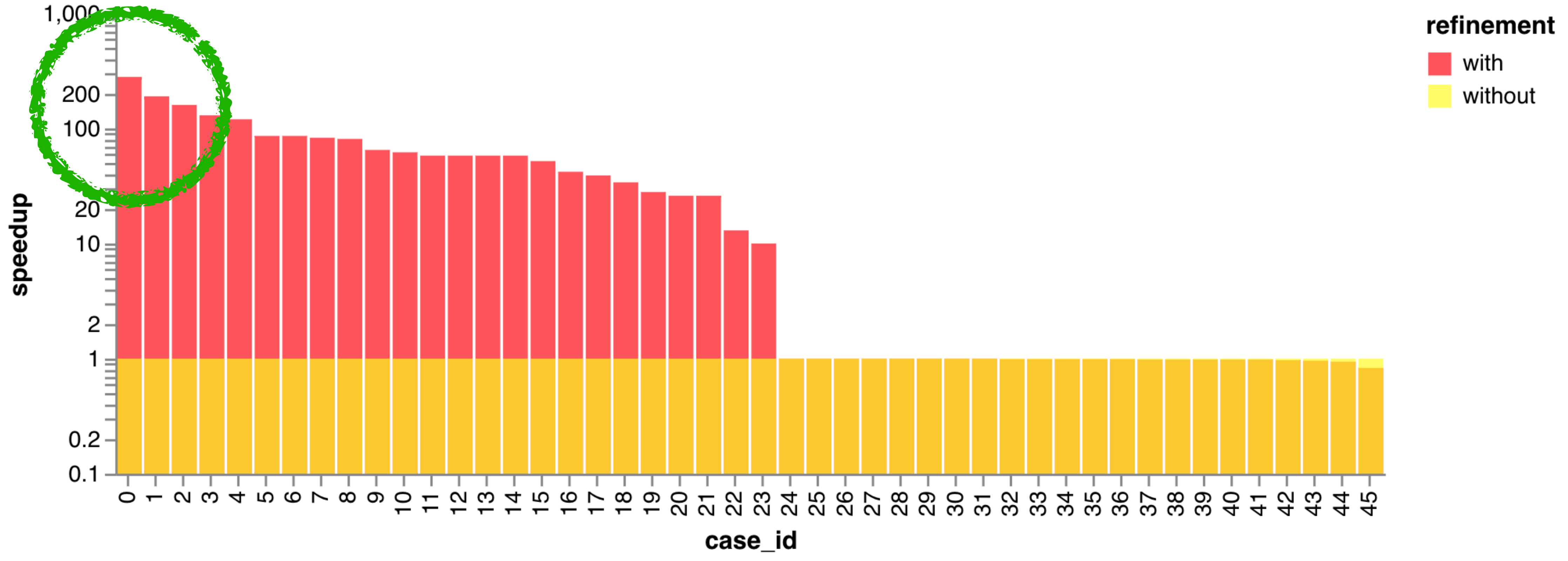
“Verify that two queries are equivalent on ALL inputs with no more than k tuples.”

Result

- *Up to 400x speed up & little overhead.*
- *Speedup is independent from solver implementation!*



***Cosette** with and without refinement*



***Qex** with and without refinement*

Experiment — Verification

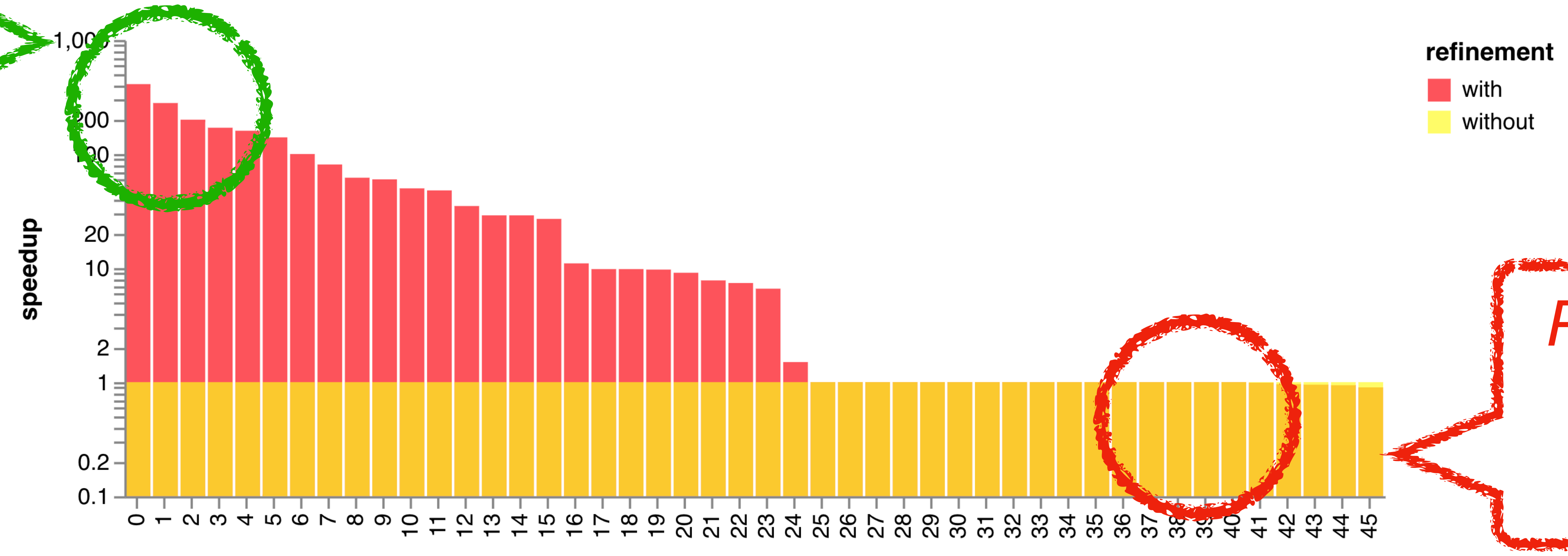
Bounded Verification

“Verify that two queries are equivalent on ALL inputs with no more than k tuples.”

```
SELECT 1
FROM (SELECT * FROM emp
      WHERE emp.deptno > 7) AS t
INNER JOIN emp AS EMP0
ON t.deptno = EMP0.deptno
INNER JOIN emp AS EMP1
ON EMP0.deptno = EMP1.deptno;
```

Result

Benefit from exponential reduction of the number of groups



Provenance is not able to refine search space

```
Select t2.name, t2.deptno, Count(*)
From   (Select name, deptno
        From Dept) As t2
Where  t2.name = 'Charlie'
Group By t2.name, t2.deptno;
```

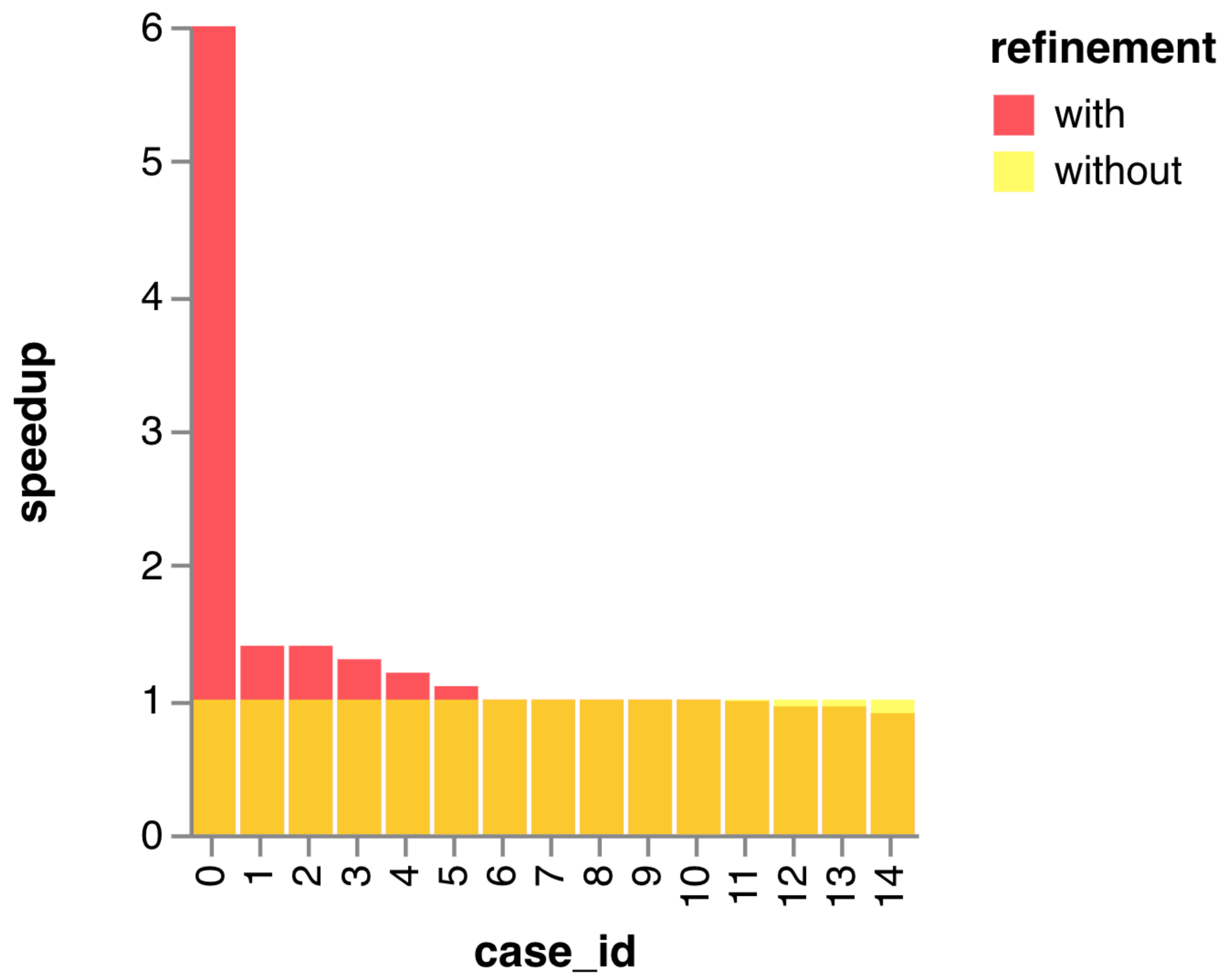
Experiment - Test Generation

Test generation

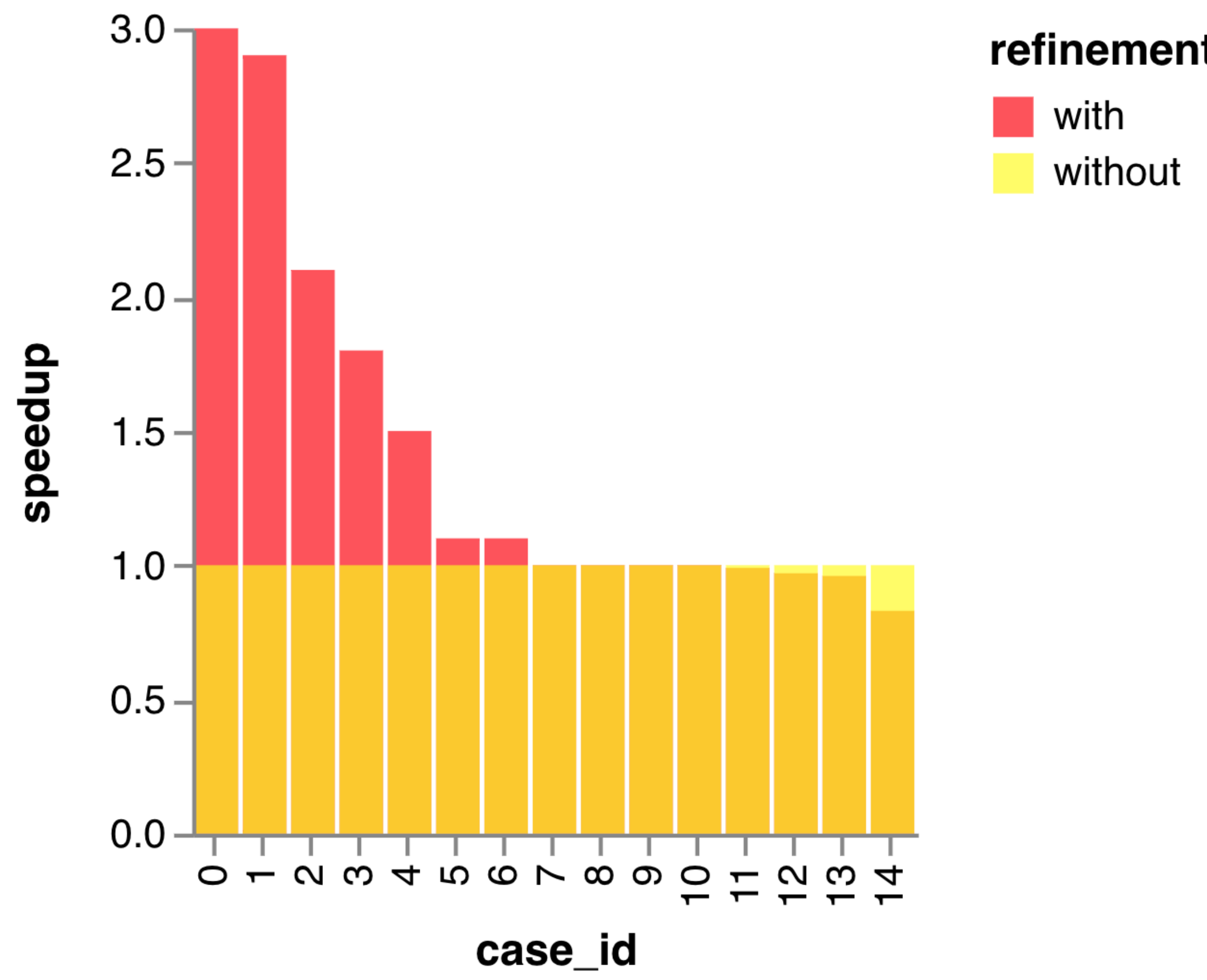
“Can the query return empty output on SOME input?”

“Find a distinguishing input between queries.”

Result



cosette with and without refinement



qex with and without refinement

Experiment - Test Generation

Test generation

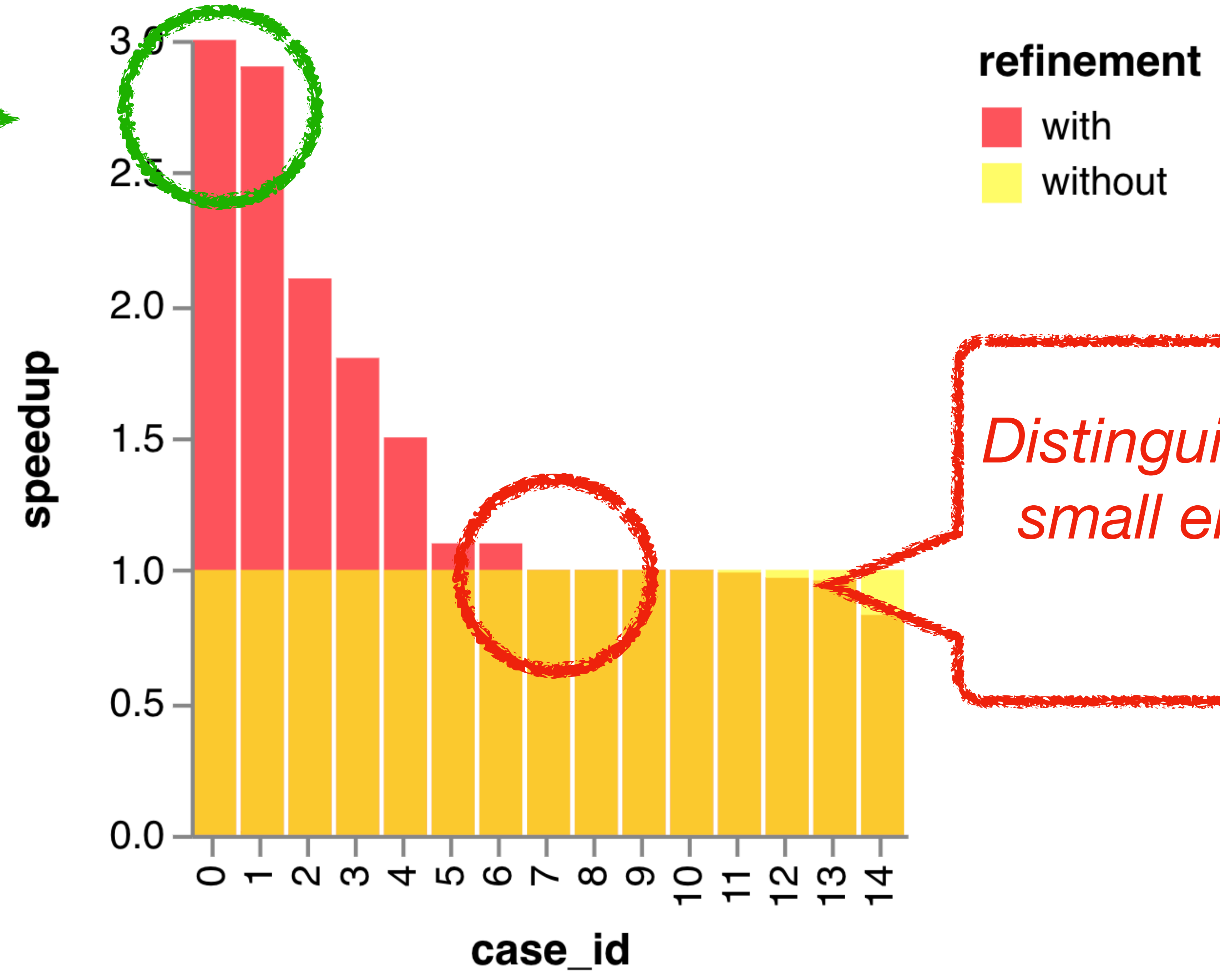
“Can the query return empty output on SOME input?”

“Find a distinguishing input between queries.”

Result

Reduction of the number of groups

```sql
SELECT Distinct dept_name
FROM course
WHERE credits =
 (SELECT MAX(credits)
  FROM course NATURAL JOIN department
  WHERE title="CS"
  GROUP BY dept
  HAVING COUNT(DISTINCT course id)>4)
```



Distinguishing input size is small enough, benefit is marginal

Limitations

- Property supported are those can be invalidate by ***one*** tuple in the output.

$q_1(T) \neq q_2(T)$

$q_1(T) \neq empty$

exists r, $\phi(q(T))$ ✓

"$q_1(T)$ contains exactly 5 tuples"

"every tuple in $q(T)$ has same multiplicity" ✗

Generalize to arbitrary property

- Improving provenance analysis precision.

Summary

Scaling Up Symbolic Reasoning for Relational Queries

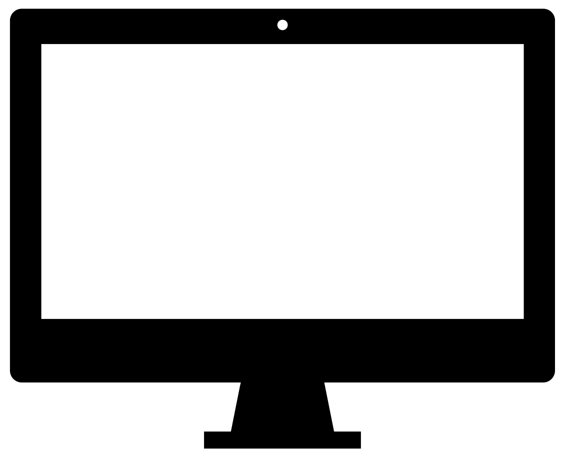
(1) Symbolic Reasoning

tables with at most *k* rows

q_1, q_2 *(queries)*

(search space)

assert $q_1 \neq q_2$ *(property)*



unsatisfiable (proved $q_1 = q_2$)

found T, $q_1(T) \neq q_2(T)$

(2) Scaling Up

q *(query)*

tables with at most *k* rows

(search space)

ϕ *(property)*

provenance analysis

tables with at most *k* rows

(refined search space)

(3) Result

tables with at most *k* rows

(1) smaller search space & easier to traverse

(2) equivalent for reasoning

Low analysis overhead & over **100x** speed up in

(1) bounded verification

(2) test generation

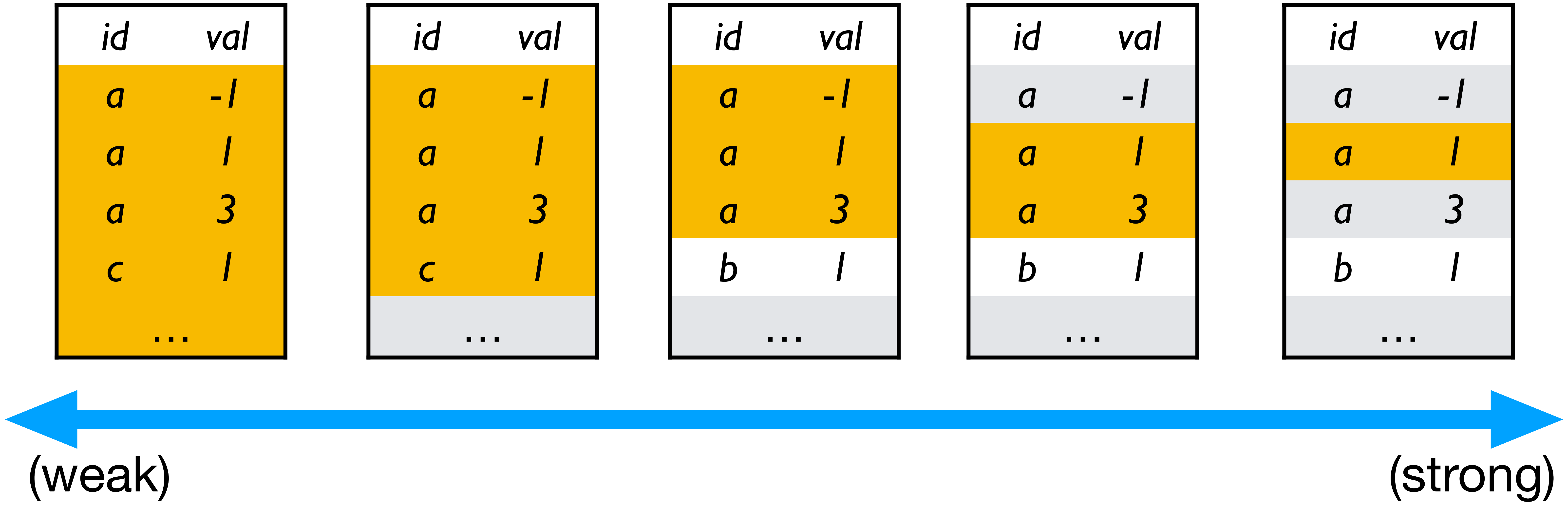
Hidden Slides!

Symbolic Provenance Analysis

Multiple provenance exists for t_{out} for a query q

id	val
a	1
…	

```
Select id, min(val)
From T
Where val > 0
Group By id
```



Choice of abstraction trades between the analysis overhead and pruning power