

# Generating Targeted Queries for Database Testing

Chaitanya Mishra, Nick Koudas

University of Toronto

Calisto Zuzarte

IBM Toronto



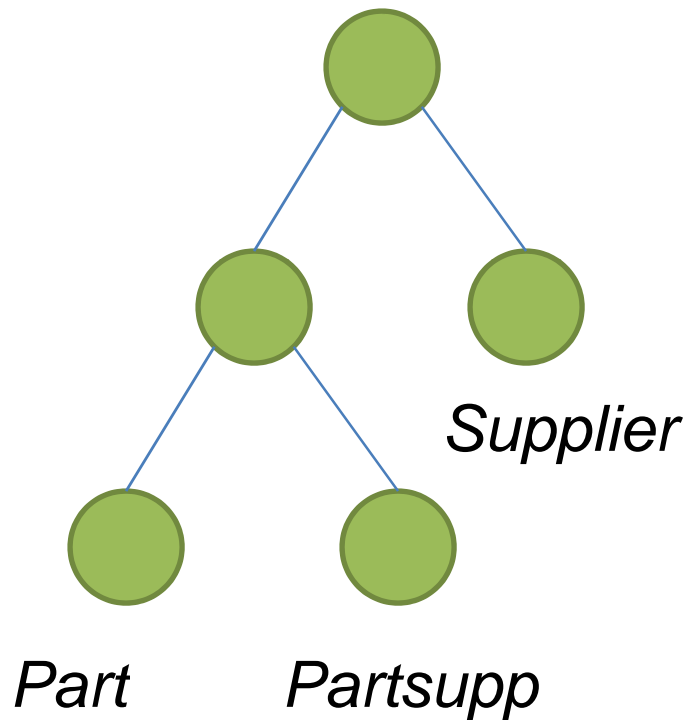
# Performance Testing

- Standard Benchmarks (e.g. TPC)
  - May not test the new feature across a wide range of conditions.
- Microbenchmarks
  - Carefully designed tests to evaluate the performance of the new feature with controlled parameters.

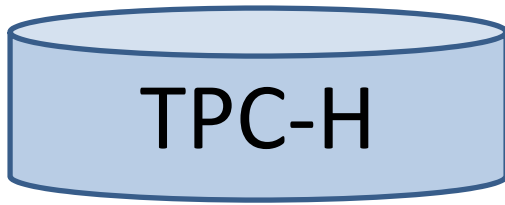
# Performance Testing: Examples

- New cardinality estimation technique:
  - Test across queries with varying selectivities.
- Modified join algorithm:
  - Test across queries with varying join input sizes and selectivities.
  - To test pipelining, vary intermediate relation sizes.
- New memory manager:
  - Vary sizes of intermediate blocking points.

# Designing a Test Query

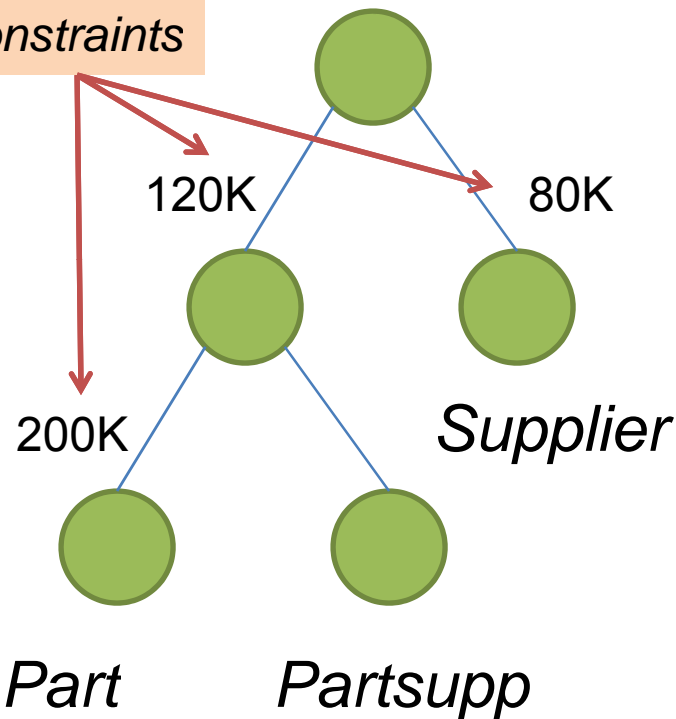


Select \* from Part P, Supplier  
S, Partsupp PS  
Where p\_partkey = ps\_partkey  
and s\_suppkey = ps\_suppkey

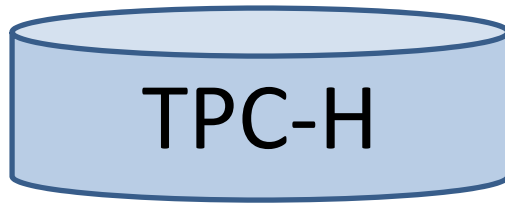


# Designing a Test Query

Constraints



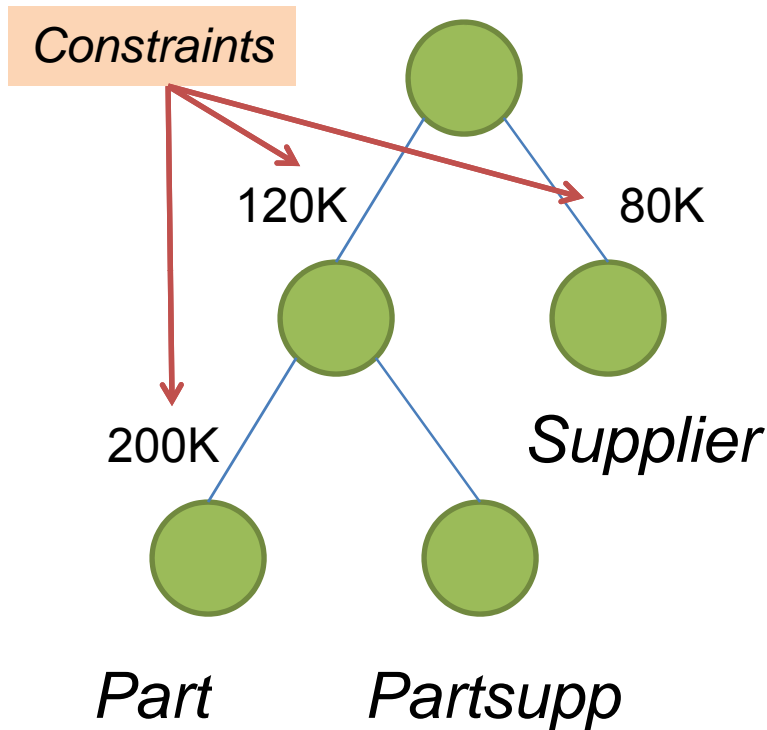
Select \* from Part P, Supplier S, Partsupp PS  
Where p\_partkey = ps\_partkey  
and s\_suppkey = ps\_suppkey



Introduction

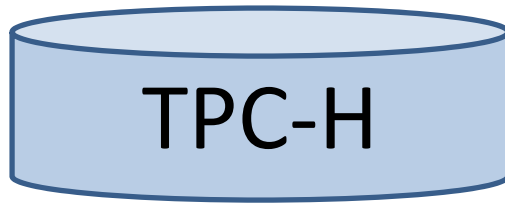
Cardinality constraints  
specified on intermediate  
relations.

# Designing a Test Query



```
Select * from Part P, Supplier S, Partsupp PS
Where p_partkey = ps_partkey
and s_suppkey = ps_suppkey
and s_acctbal < ?
and p_retailprice < ?
and ps_availqty < ?
and ps_supplycost < ?
```

**Predicates**

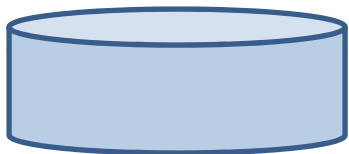


Introduction

Cardinality constraints specified on intermediate relations.

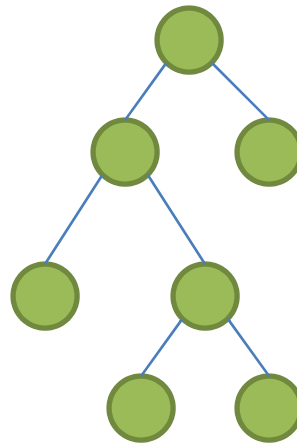
# Targeted Query Generation (TQG)

**Given:**



$D$

+



$Q$

+

$(Q_1, N_1) \dots (Q_m, N_m)$   
 $|Q_i| = N_i$

$m$  Constraints

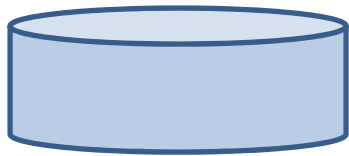
$x_1 < c_1 \dots x_d < c_d$   
 $d$  range predicates

**Generate:**

$Q^r : x_1 < c_1^r \dots x_d < c_d^r$

s.t  $Q^r$  satisfies the constraints when executed on  $D$ .

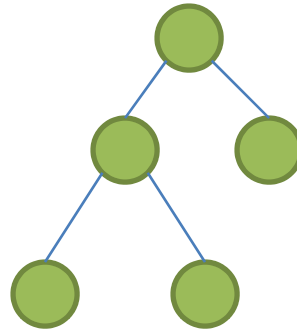
# Targeted Query Generation (TQG)



*TPCH*

*Database*

+



$P \bowtie PS \bowtie S$

+

$|P| = 200K$

$|P \bowtie PS| = 120K$

$|S| = 80K$

$s\_acctbal < 6000$

$p\_retailprice < 2500$

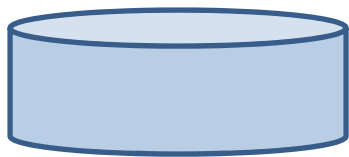
$ps\_availqty < 50$

$ps\_supplycost < 1000$

*Constraints*

*Predicates*

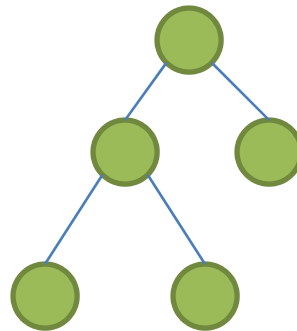
# Targeted Query Generation (TQG)



*TPCH*

*Database*

+



$P \bowtie PS \bowtie S$

+

$|P| = 200K$

$|P \bowtie PS| = 120K$

$|S| = 80K$

$s\_acctbal < 9000$

$p\_retailprice < 1500$

$ps\_availqty < 30$

$ps\_supplcost < 2000$

*Constraints*

*Predicates*

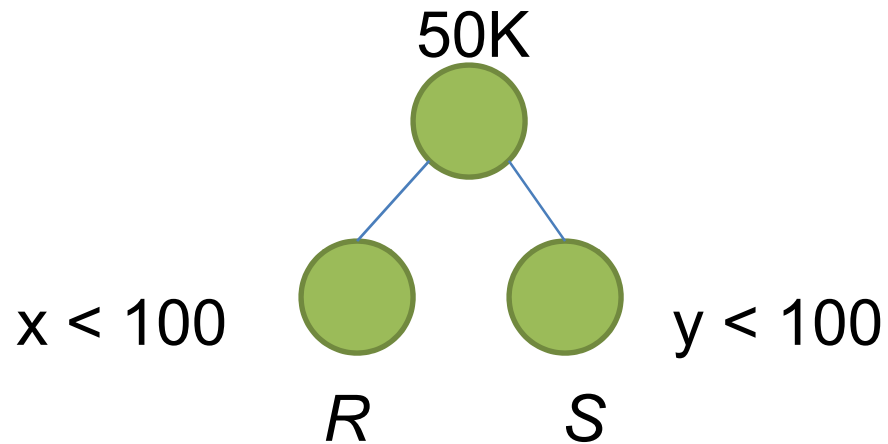
# Outline

- Base Cases
- TQGen Algorithm
- Cardinality Estimation
- Implementation and Experiments

# Outline

- Base Cases
  - Single Constraint
  - Multiple Constraints with Independence
- TQGen Algorithm
- Cardinality Estimation
- Implementation and Experiments

# Single Constraint



- Query  $Q$ , Database  $D$ , Constraint  $(Q, N)$ .
- [BCT06]: NP-complete.

# Lowerbound for Exact Solution

# distinct values

# predicates

- [BCT06]: Lowerbound of  $n+d-2 C_{d-1}$  calls to the cardinality estimation component to satisfy constraint (Q,N) exactly.
- There may be no query that satisfies the constraint exactly!

# Lowerbound for *Approx.* Solution

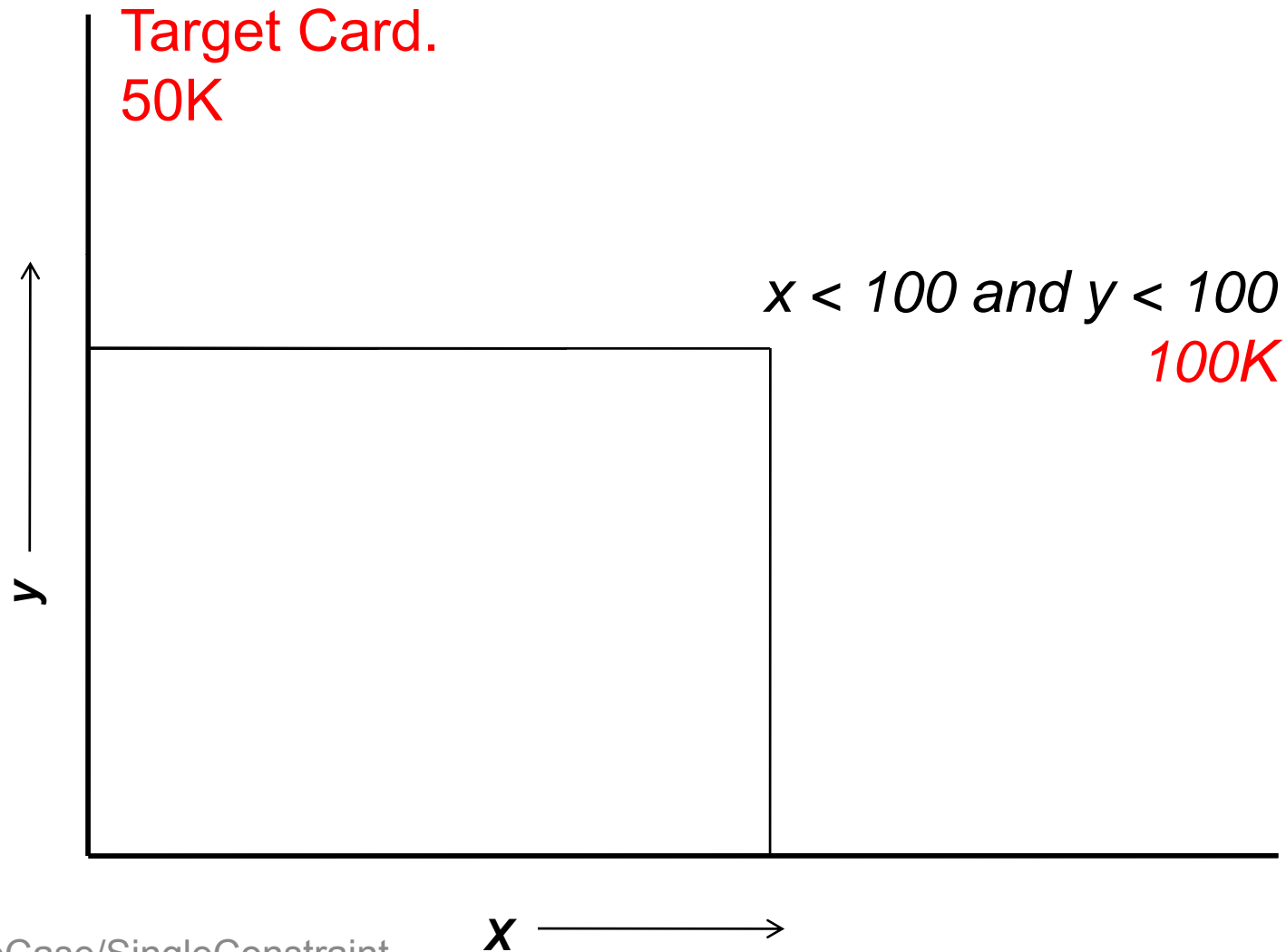
- Lowerbound of  ${}^{n+d-2}C_{d-1}$  calls to the cardinality estimation component to satisfy constraint (Q,N) to within  $[N-E, N+E]$  or  $[N(1-\epsilon), N(1+\epsilon)]$ .



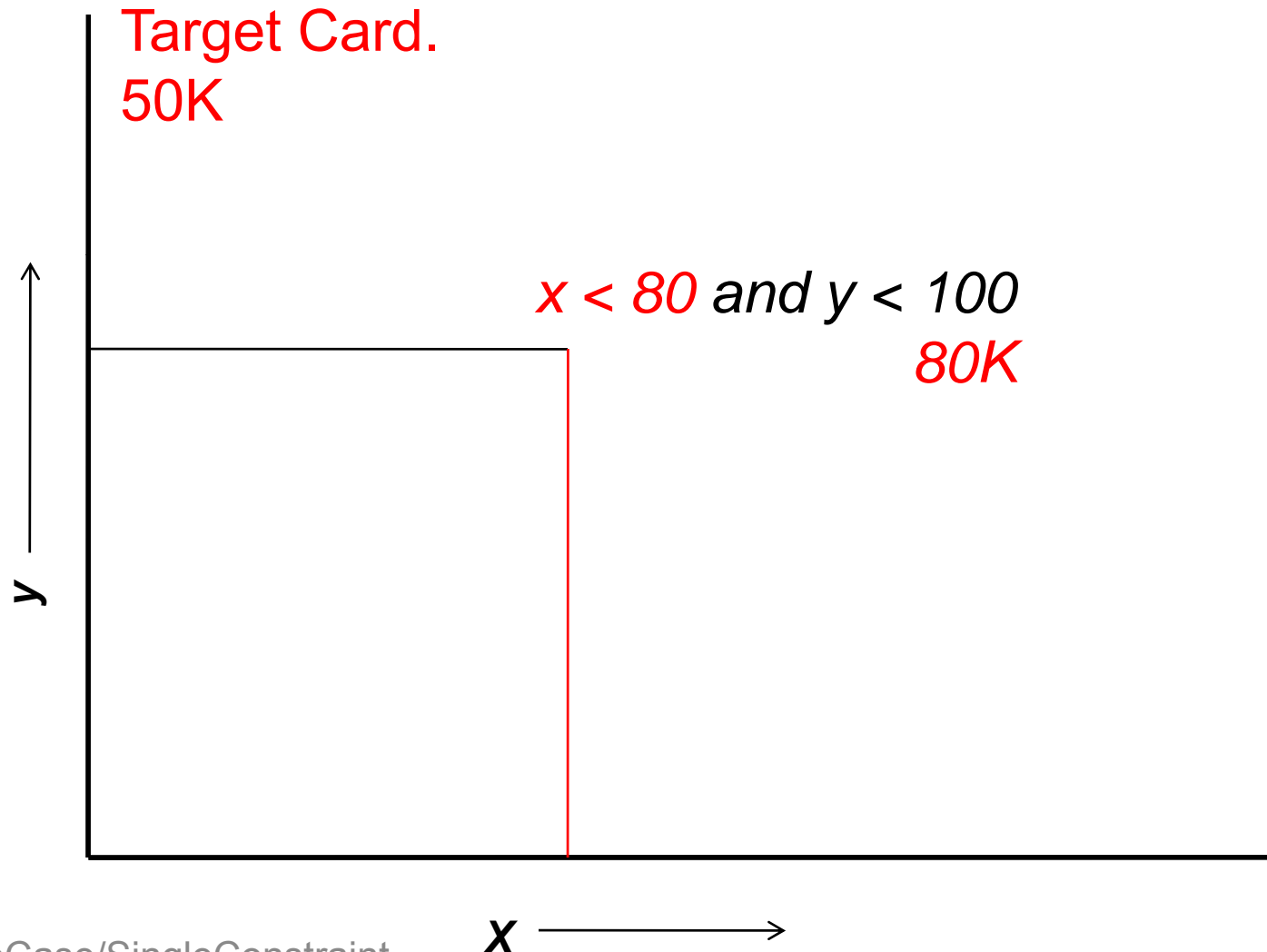
Constants

- Approximating to within an arbitrary absolute or relative error is hard!

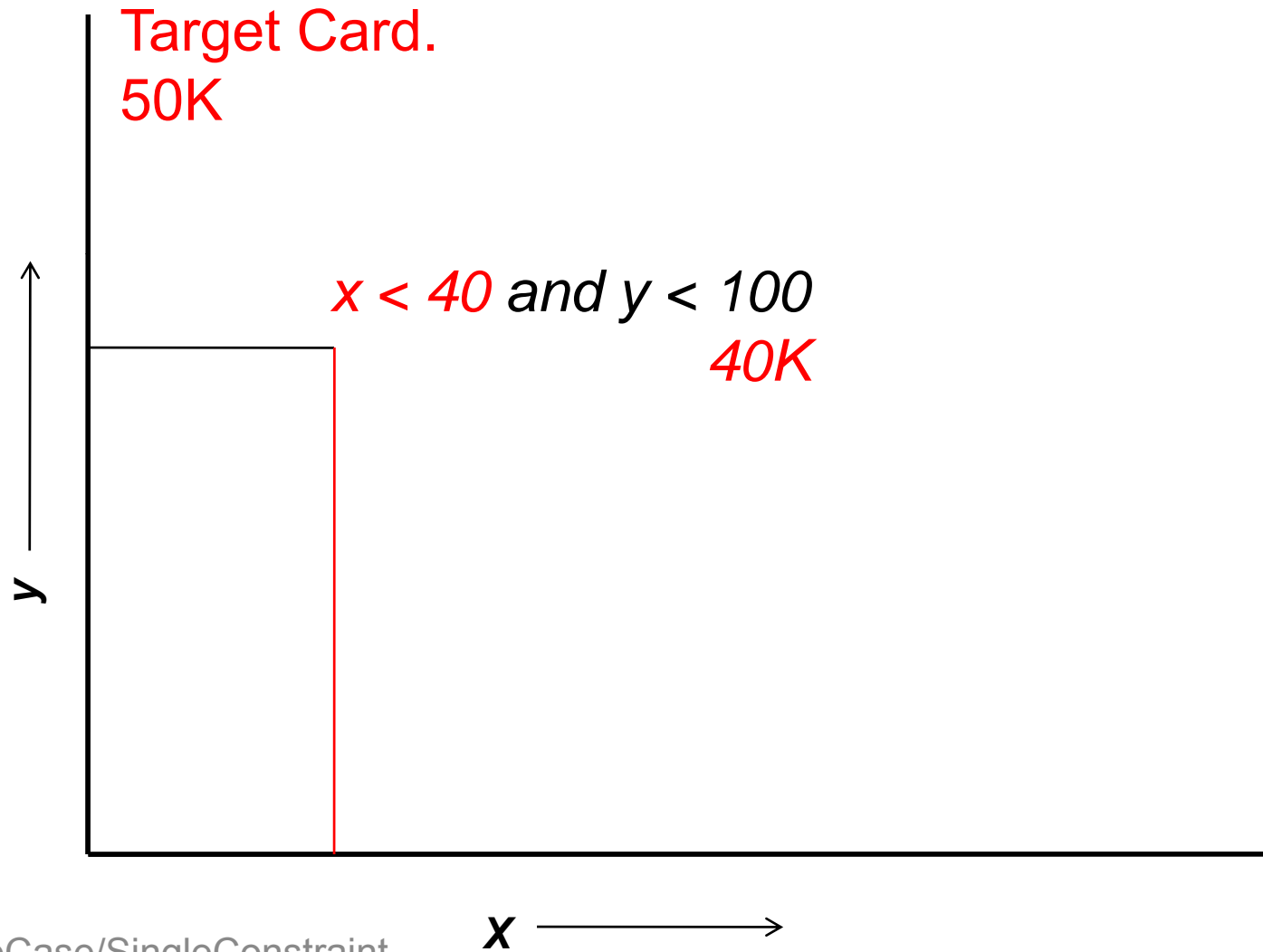
# Binary Search



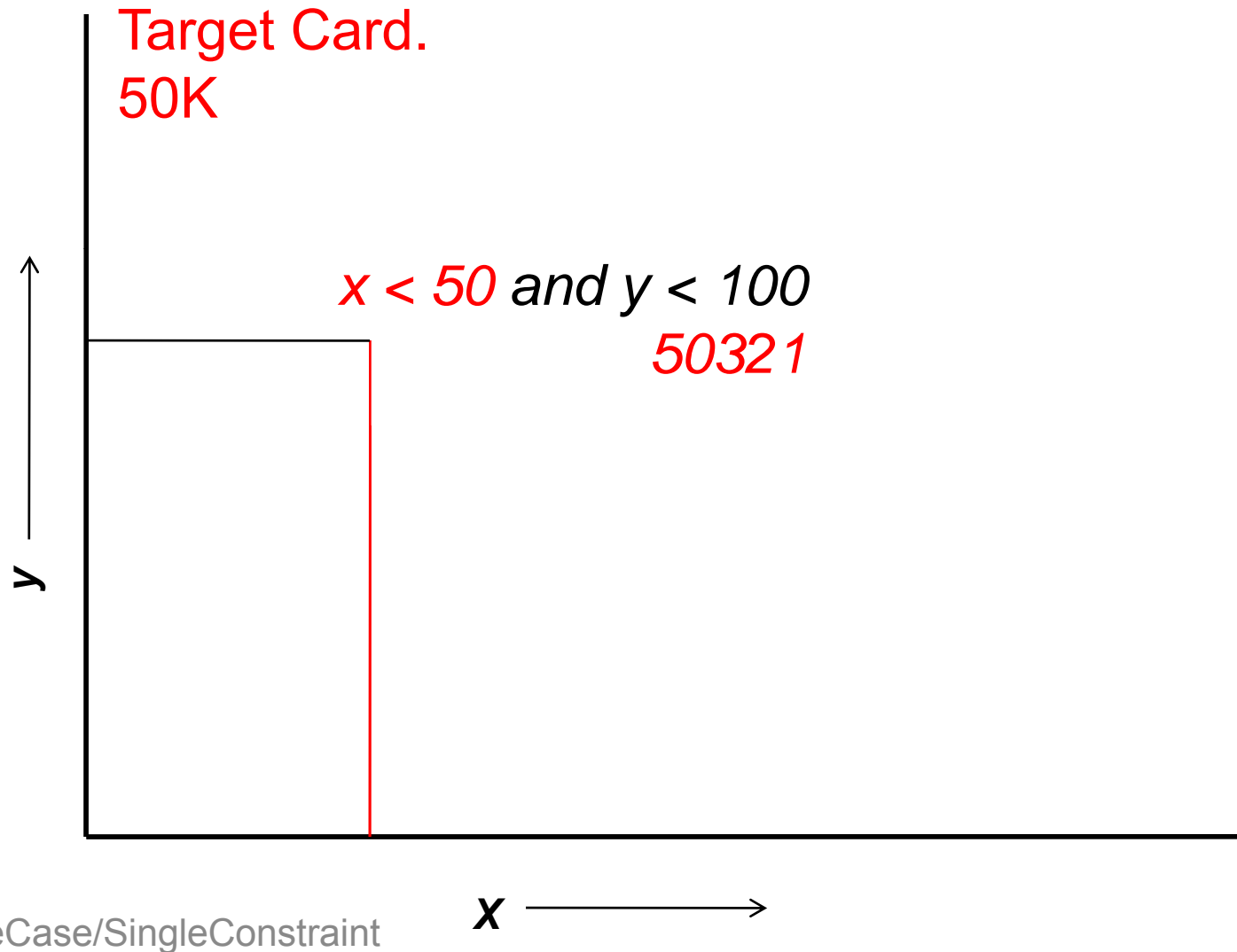
# Binary Search



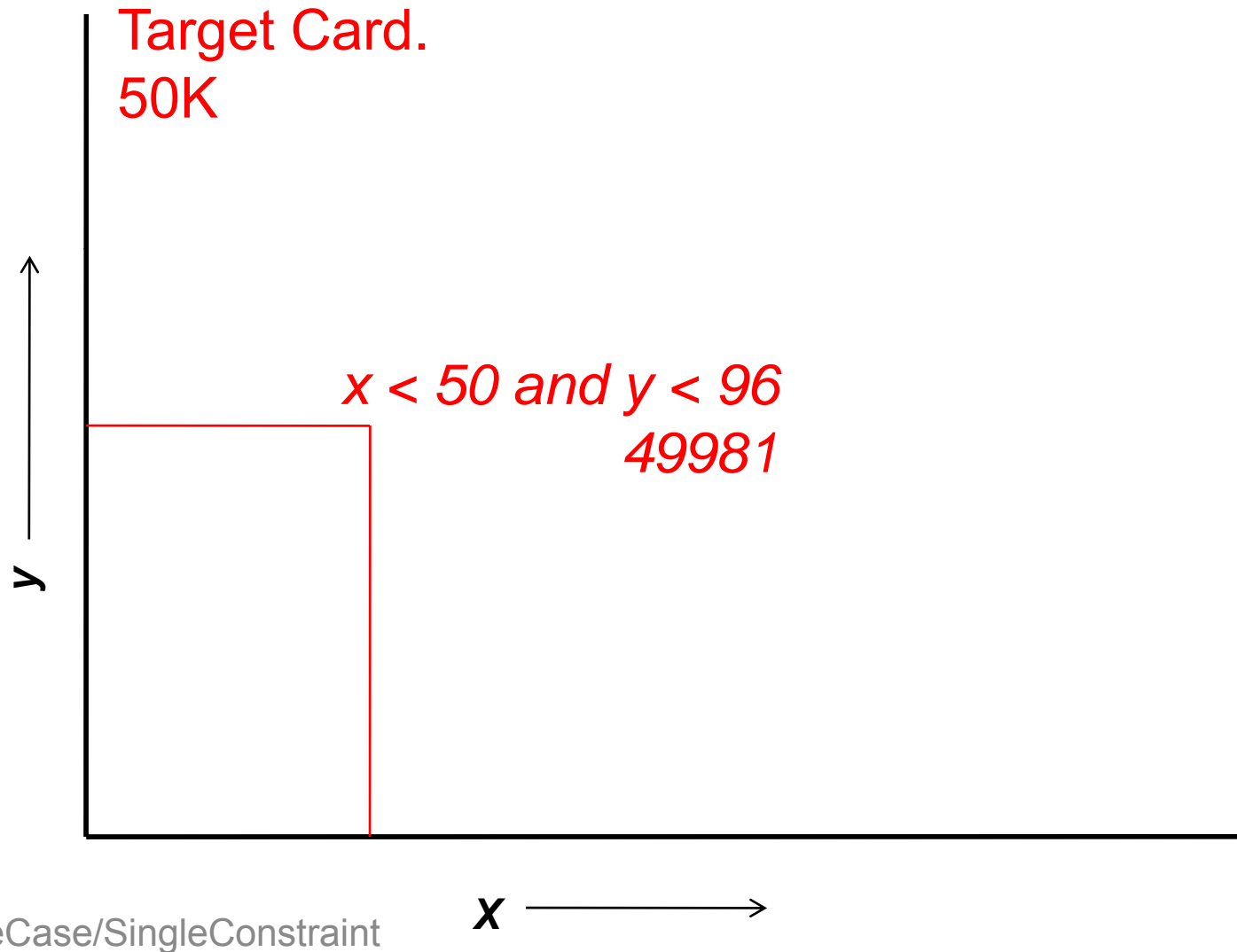
# Binary Search



# Binary Search



# Binary Search



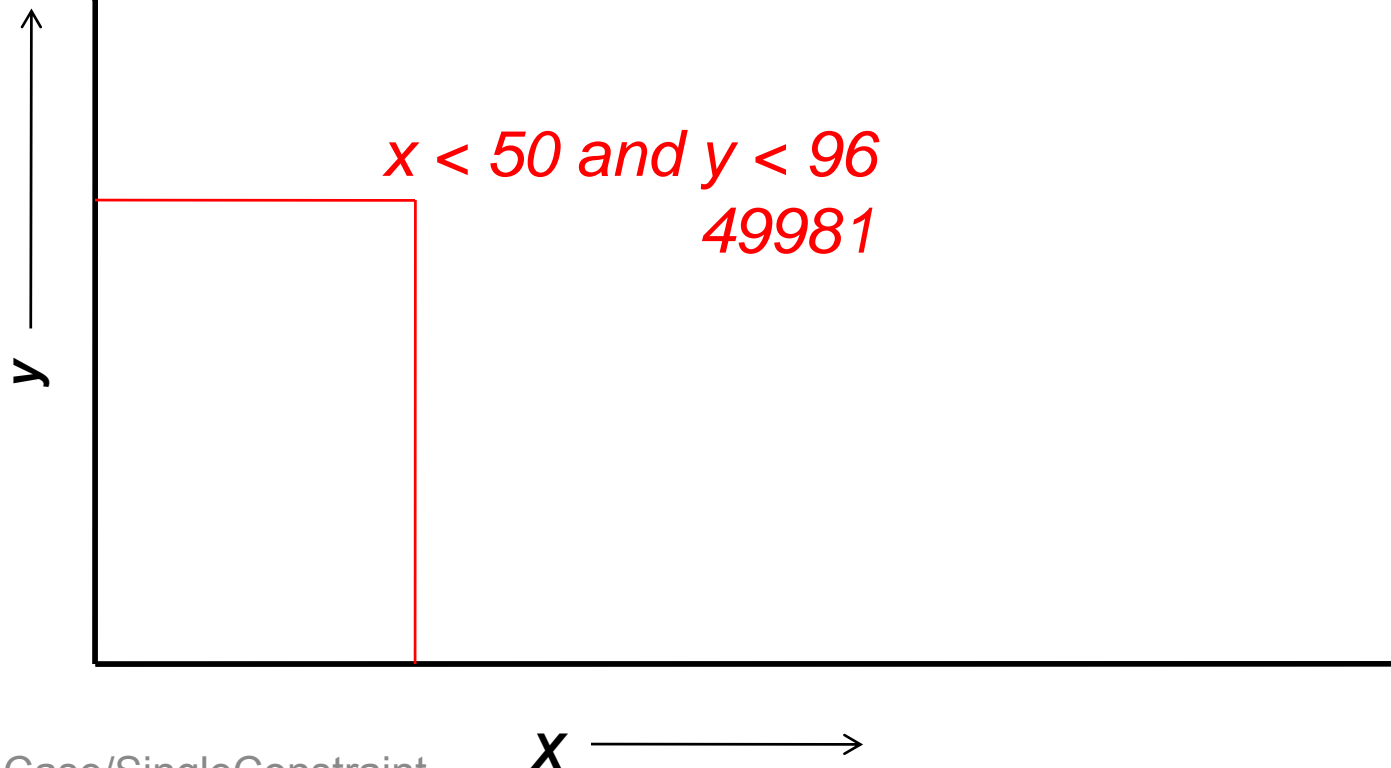
# Binary Search

# predicates

# distinct values

Target Card.  
50K

- Cost:  $O(d \log n)$
- Error =  $[ \min_i( \maxfreq(i) ) ] / 2$



# Binary Search

# predicates

# distinct values

Target Card.  
50K

- Cost:  $O(d \log n)$
- Error =  $\lceil \min_i(\maxfreq(i)) \rceil / 2$

*Discreteness makes the problem hard.*

$x < 50$  and  $y < 96$   
49981

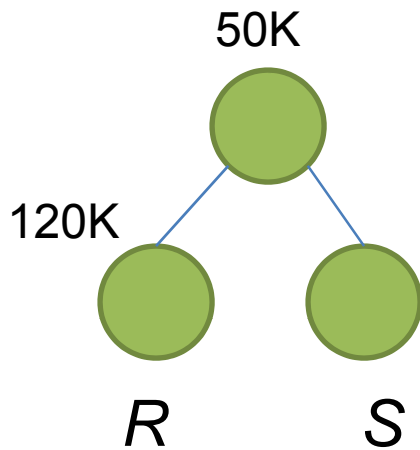
y

x

# Outline

- Base Cases
  - Single Constraint
  - Multiple Constraints with Independence
- TQGen Algorithm
- Cardinality Estimation
- Implementation and Experiments

# Multiple Constraints



Select \* from R, S

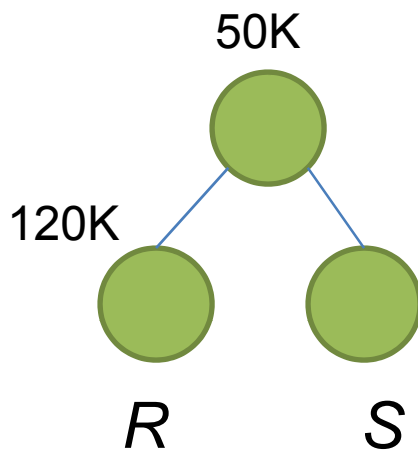
Where R.id = S.id

And R.x <  $C_x$

And R.y <  $C_y$

And S.z <  $C_z$

# Multiple Constraints: Equations



$$X = \log(\text{sel}(R.x < C_x))$$

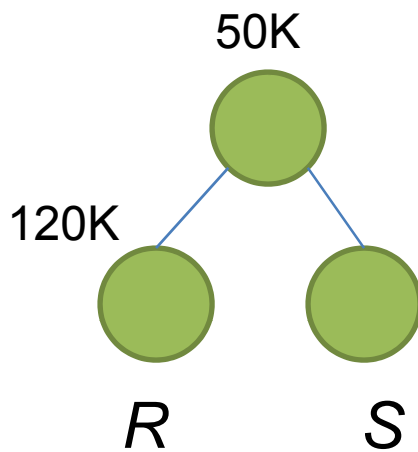
$$X + Y = \log(120K) - \log(|R|)$$

$$X + Y + Z = \log(50K) - \log(|R \bowtie S|)$$

$$X, Y, Z \leq 0$$

- Assuming independence between attributes, express relationship between predicate selectivities and cardinality constraints as a system of linear equations.

# Multiple Constraints: Equations



$$X = \log(\text{sel}( R.x < C_x ))$$

$$X + Y = \log(120K) - \log(|R|)$$

$$X + Y + Z = \log(50K) - \log(|R \bowtie S|)$$

$$X, Y, Z \leq 0$$

- Solve for X, Y, Z.
- Use Single Constraint procedure to find  $C_x C_y C_z$

*Independence reduces the multiple constraint problem to the single constraint version.*

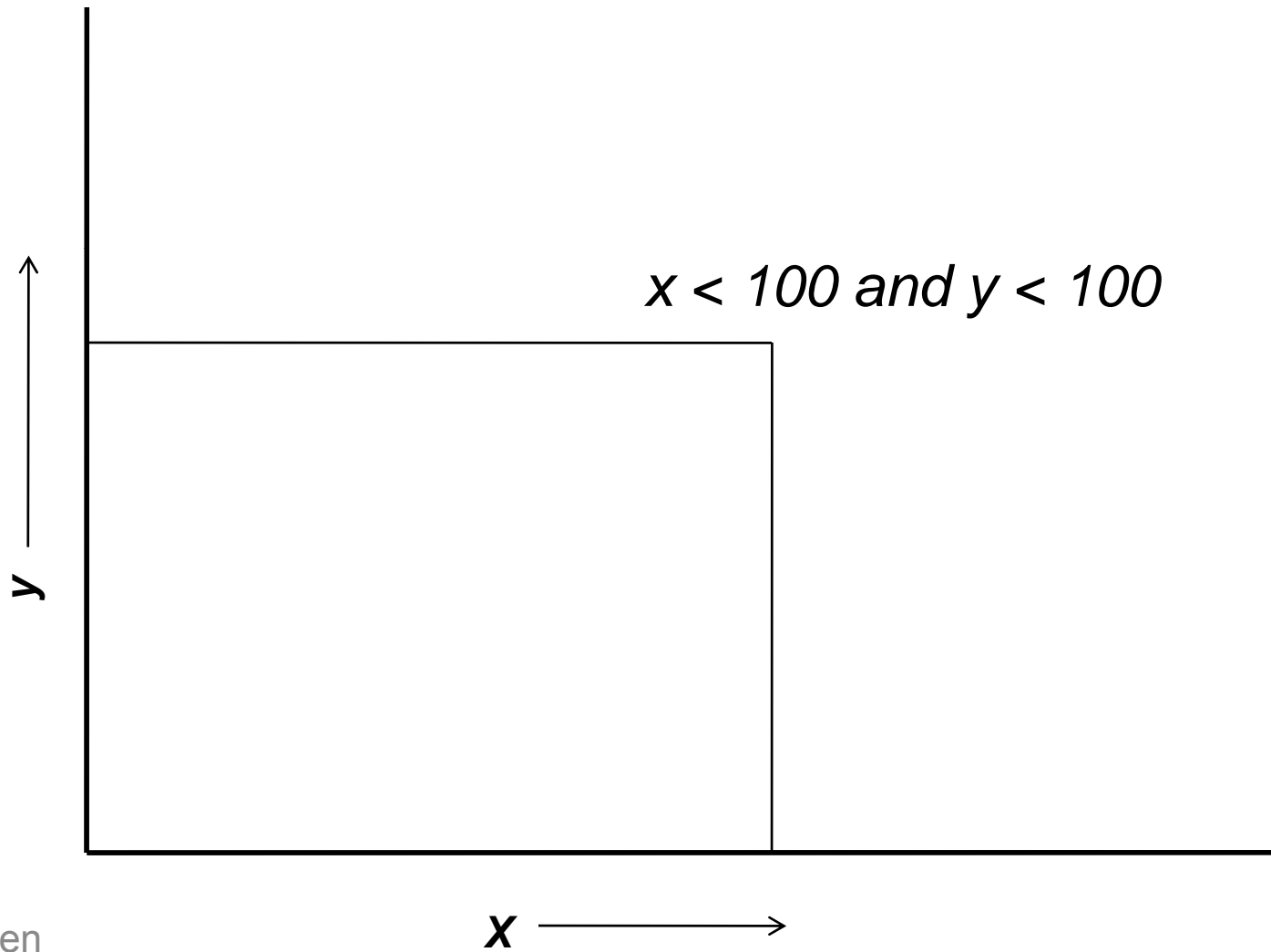
# Outline

- Base Cases
- TQGen Algorithm
- Cardinality Estimation
- Implementation and Experiments

# TQGen

Search procedure for targeted query generation  
given an arbitrary number of constraints  
*without making distributional assumptions.*

# TQGen



# TQGen

*Bounding*

$x < 150$  and  $y < 160$

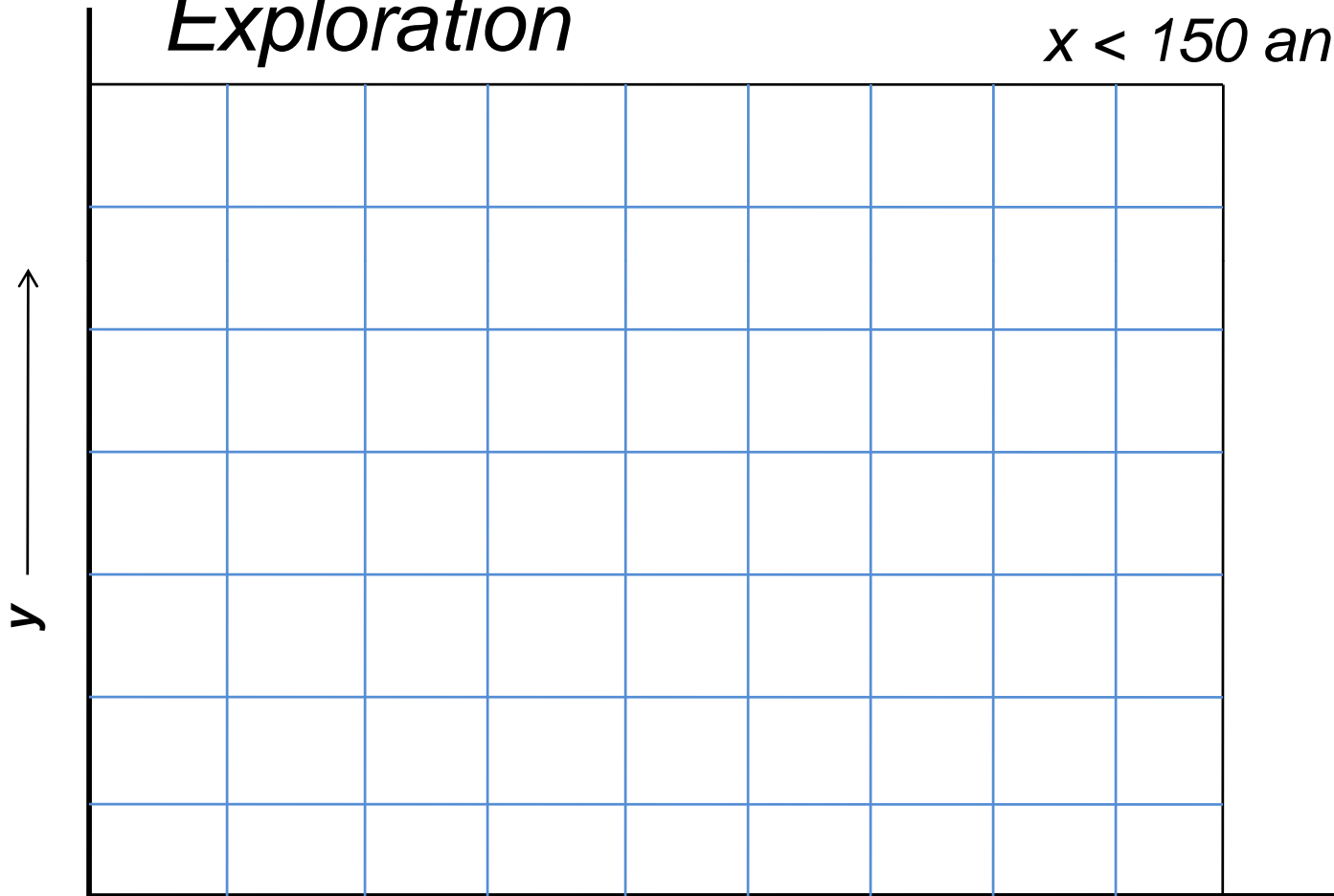
$y$  ↑

$x$  →

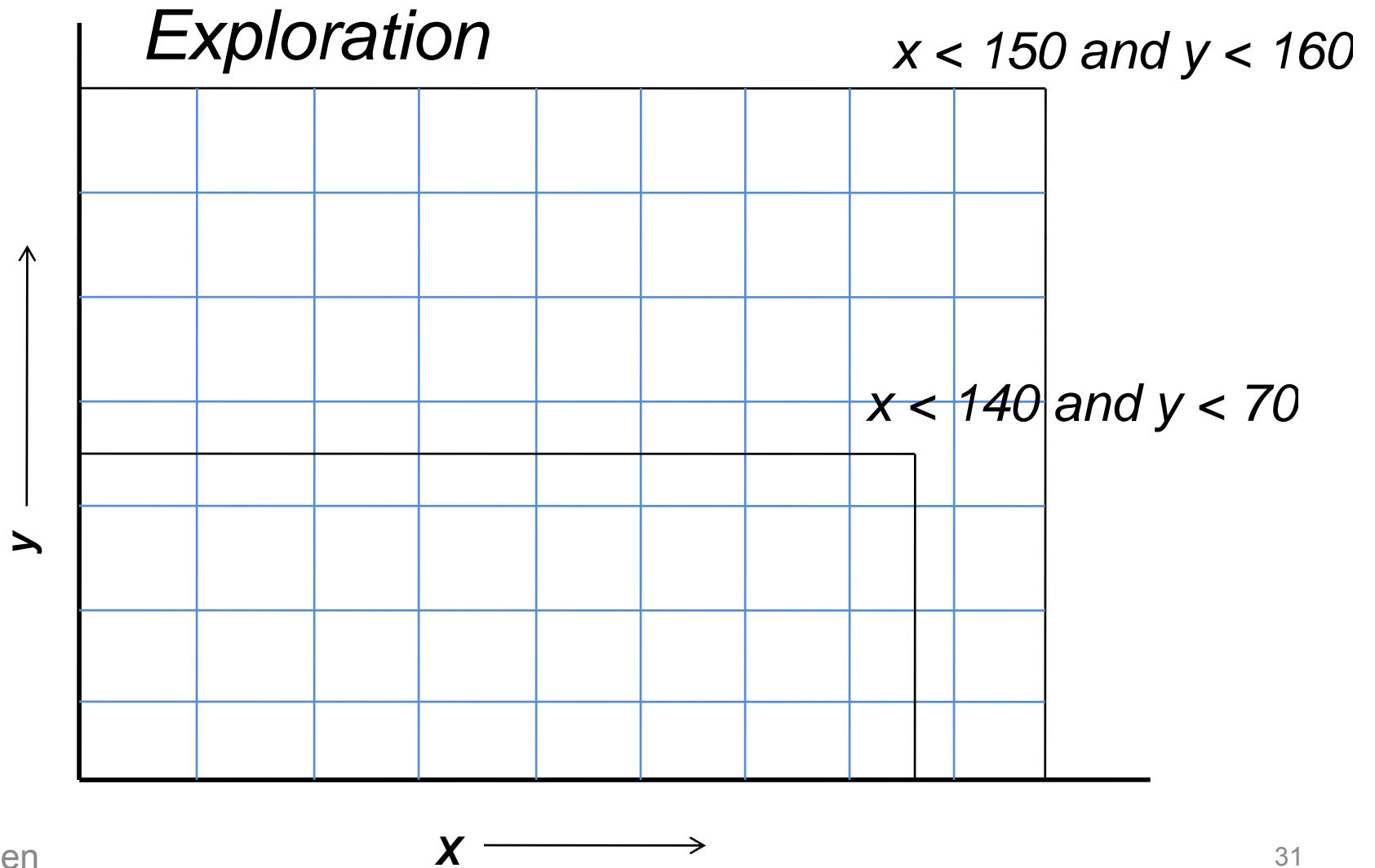
# TQGen

*Exploration*

$x < 150$  and  $y < 160$



# TQGen

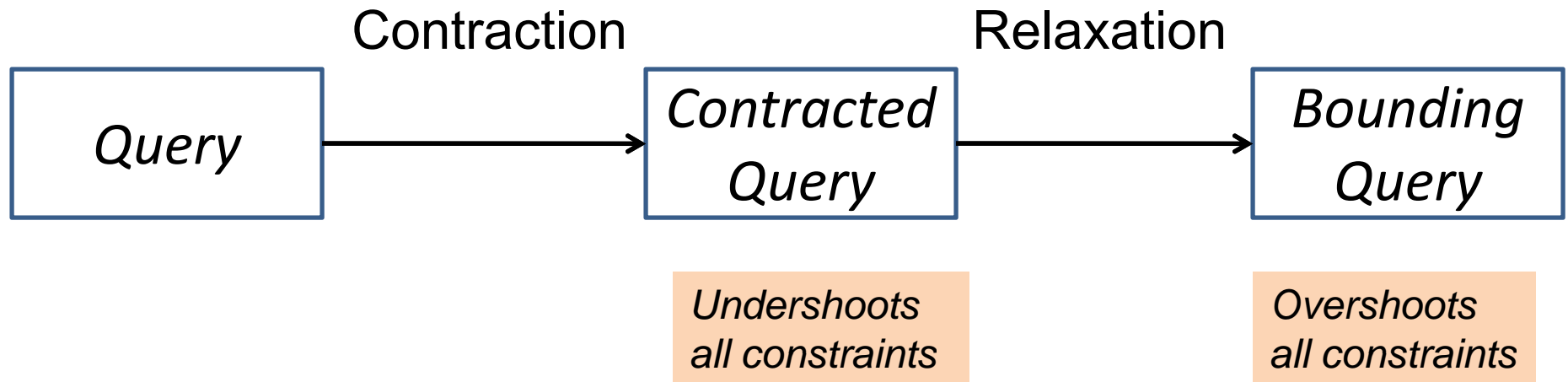


# Outline

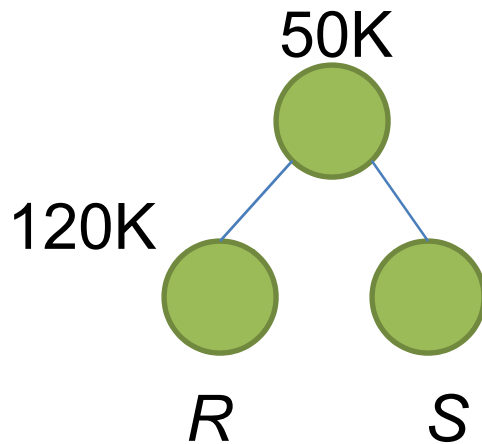
- Base Cases
- TQGen Algorithm
  - Bounding
  - Exploration
  - Scoring and Pruning
- Cardinality Estimation
- Implementation and Experiments

# Bounding

- Restrict the search space for solutions.
- Two phase bounding



# Bounding



Select \* from R, S

Where R.id = S.id

And R.x < 100

And R.y < 100

And S.z < 100

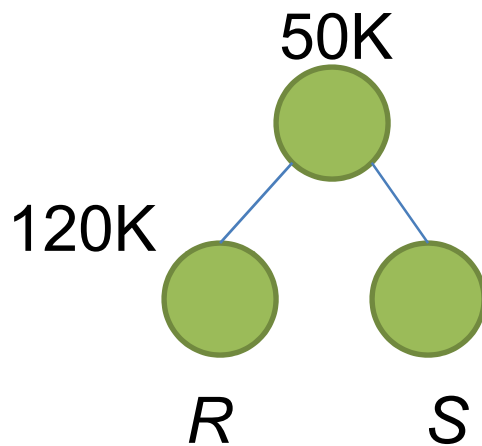
TQGen/Bounding

Original Cardinality:

$|R \bowtie S| = 70K$

$|R| = 80K$

# Bounding: Contraction

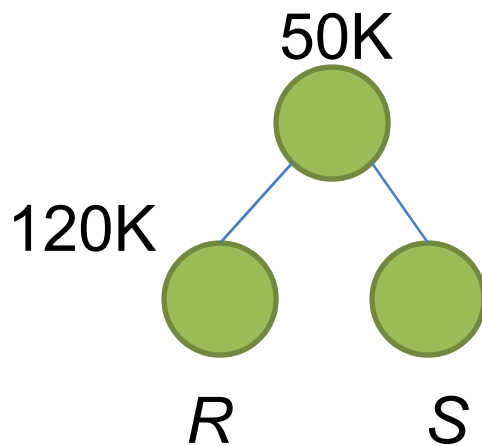


Independently Contract each predicate till you undershoot all constraints affected by it.

Select \* from R, S  
Where R.id = S.id  
And R.x < 100  
And R.y < 100  
And S.z < 100

Original Cardinality:  
 $|R \bowtie S| = 70K$   
 $|R| = 80K$

# Bounding: Contraction

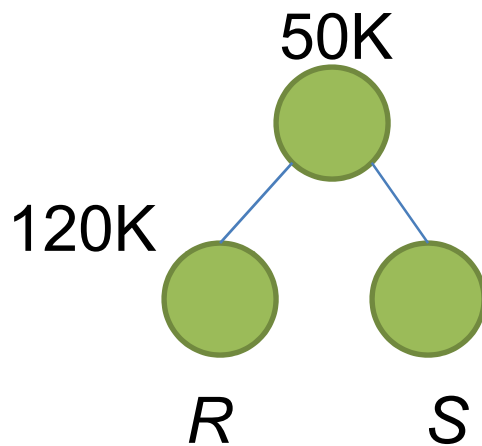


Independently Contract each predicate till you undershoot all constraints affected by it.

Select \* from R, S  
Where R.id = S.id  
And R.x < 50  
And R.y < 100  
And S.z < 100

Current Cardinality:  
 $|R \bowtie S| = 50K$   
 $|R| = 65K$

# Bounding: Contraction

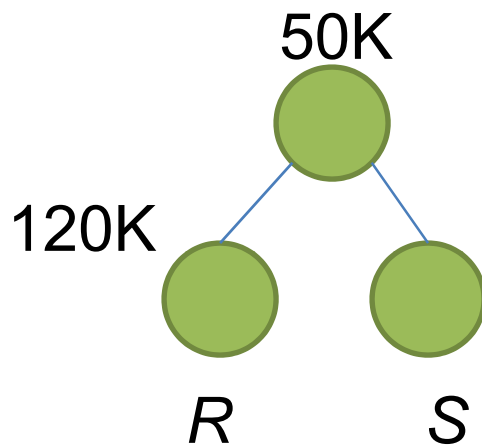


Independently Contract each predicate till you undershoot all constraints affected by it.

Select \* from R, S  
Where R.id = S.id  
And R.x < 100  
And R.y < 40  
And S.z < 100

Current Cardinality:  
 $|R \bowtie S| = 50K$   
 $|R| = 70K$

# Bounding: Contraction

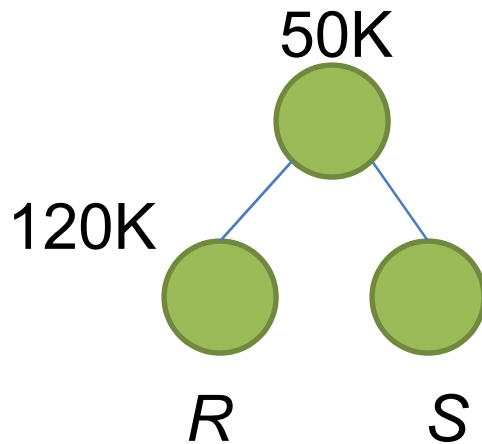


Independently Contract each predicate till you undershoot all constraints affected by it.

Select \* from R, S  
Where R.id = S.id  
And R.x < 100  
And R.y < 100  
And S.z < 80

Current Cardinality:  
 $|R \bowtie S| = 50K$   
 $|R| = 80K$

# Contracted Query



Select \* from R, S

Where R.id = S.id

And R.x < 50

And R.y < 40

And S.z < 80

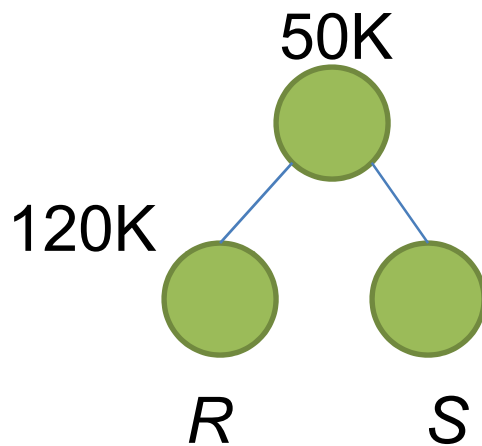
TQGen/Bounding

Contracted Query Cardinality:

$|R \bowtie S| = 20K$

$|R| = 55K$

# Bounding: Relaxation



Independently Relax each predicate till you overshoot all constraints affected by it.

Select \* from R, S

Where R.id = S.id

And R.x < 50

And R.y < 40

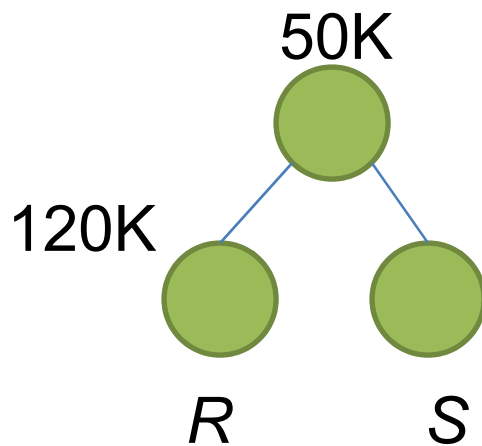
And S.z < 80

Contracted Query Cardinality:

$|R \bowtie S| = 20K$

$|R| = 55K$

# Bounding: Relaxation

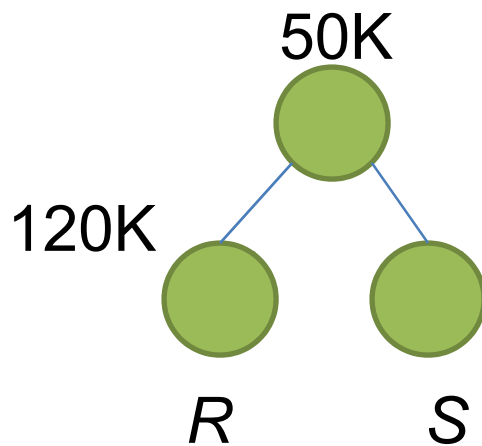


Independently Relax each predicate till you overshoot all constraints affected by it.

Select \* from R, S  
Where R.id = S.id  
And R.x < 150  
And R.y < 40  
And S.z < 80

Current Cardinality:  
 $|R \bowtie S| = 80K$   
 $|R| = 120K$

# Bounding: Relaxation



Independently Relax each predicate till you overshoot all constraints affected by it.

Select \* from R, S

Where R.id = S.id

And R.x < 50

And R.y < 160

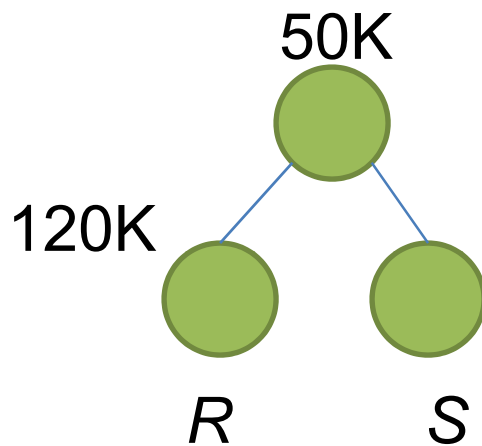
And S.z < 80

Current Cardinality:

$|R \bowtie S| = 70K$

$|R| = 120K$

# Bounding: Relaxation



Independently Relax each predicate till you overshoot all constraints affected by it.

Select \* from R, S

Where R.id = S.id

And R.x < 50

And R.y < 40

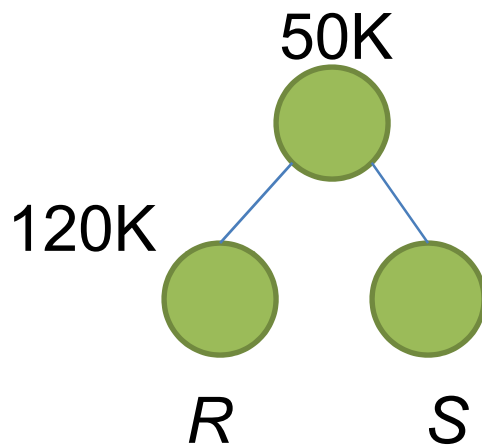
And S.z < 190

Current Cardinality:

$|R \bowtie S| = 50K$

$|R| = 55K$

# Bounding: Relaxation



Two phase bounding ensures that we consider a sufficiently large range of values for each predicate.

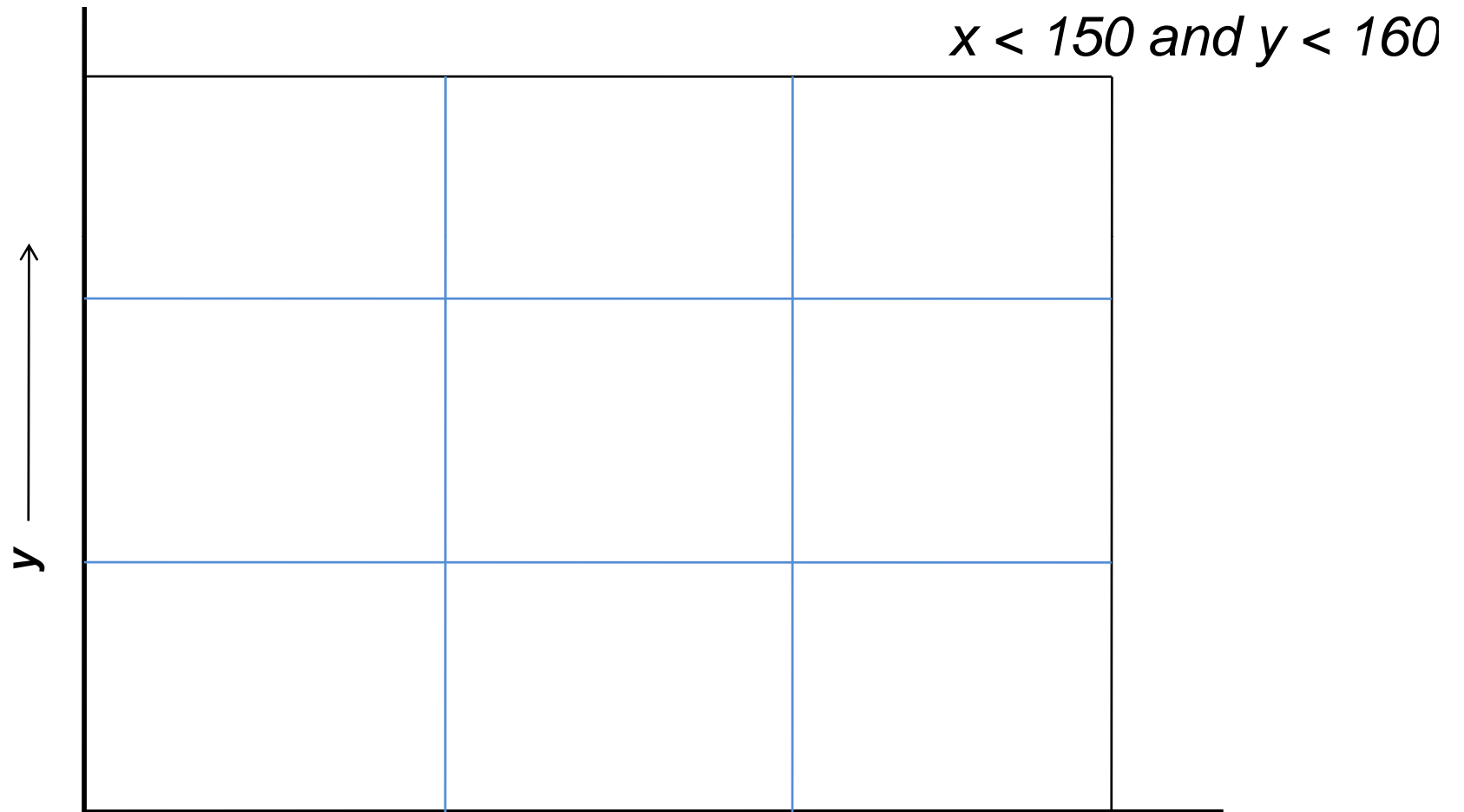
Select \* from R, S  
Where R.id = S.id  
And R.x < 150  
And R.y < 160  
And S.z < 190

Bounding Query Cardinality:  
 $|R \bowtie S| = 150K$   
 $|R| = 200K$

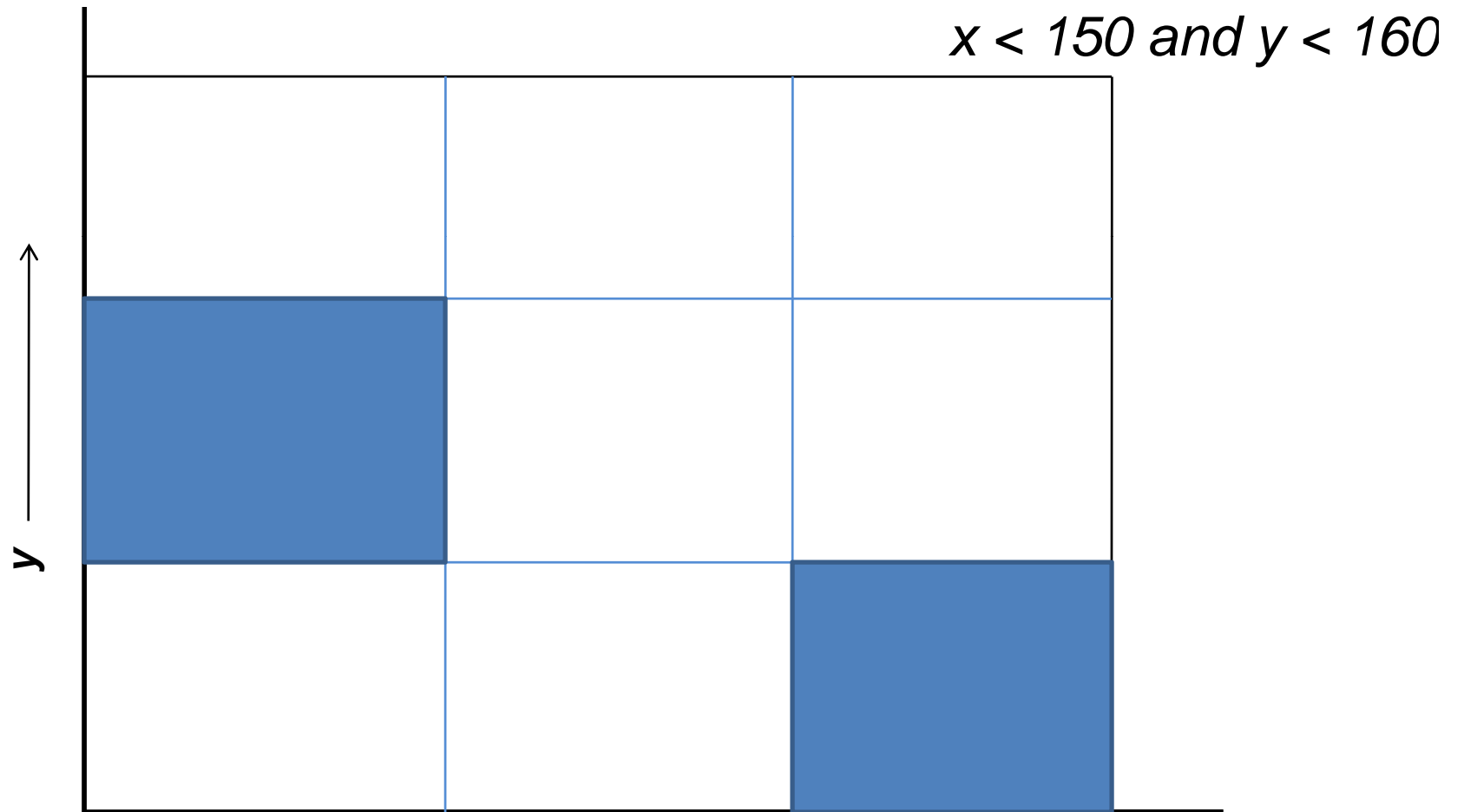
# Outline

- Base Cases
- TQGen Algorithm
  - Bounding
  - Exploration
  - Scoring and Pruning
- Cardinality Estimation
- Implementation and Experiments

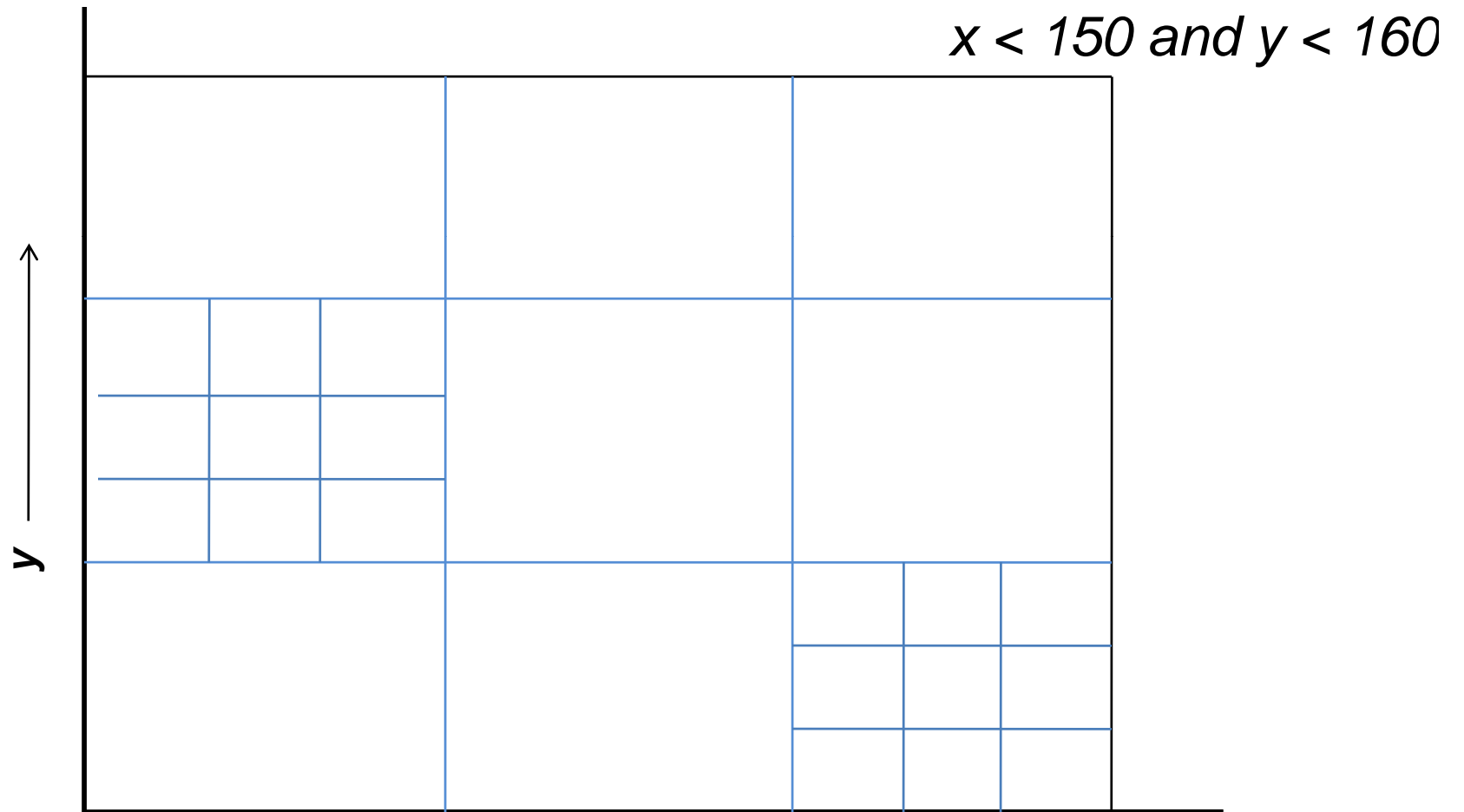
# Exploration



# Exploration



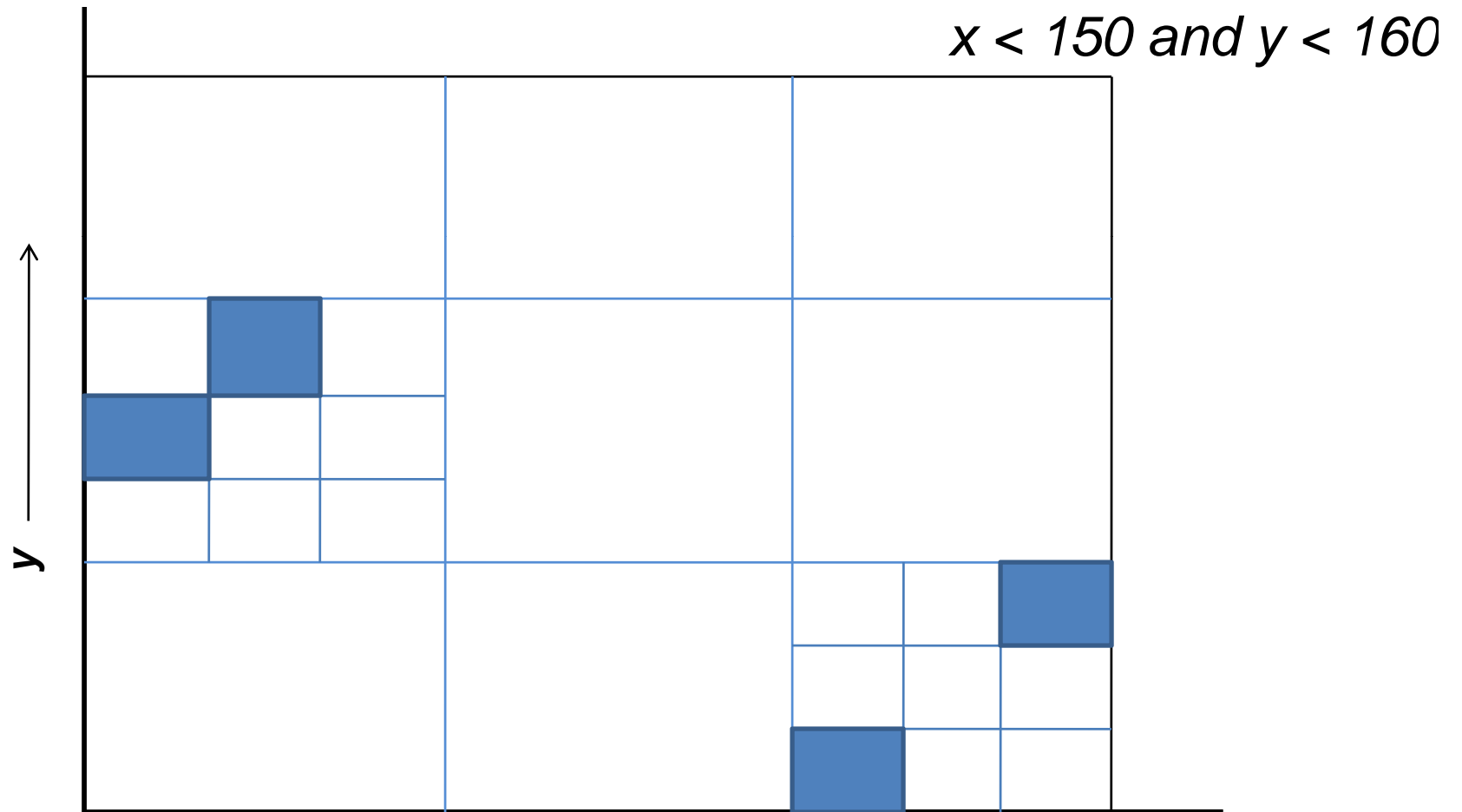
# Exploration



TQGen/Exploration

$x$  →

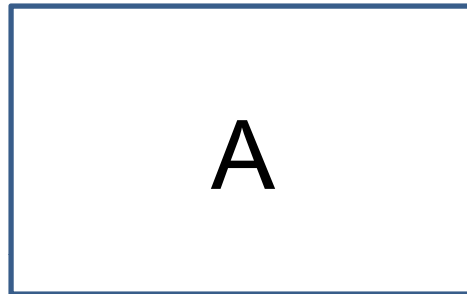
# Exploration



# Outline

- Base Cases
- TQGen Algorithm
  - Bounding
  - Exploration
  - Scoring
- Cardinality Estimation
- Implementation and Experiments

# Scoring



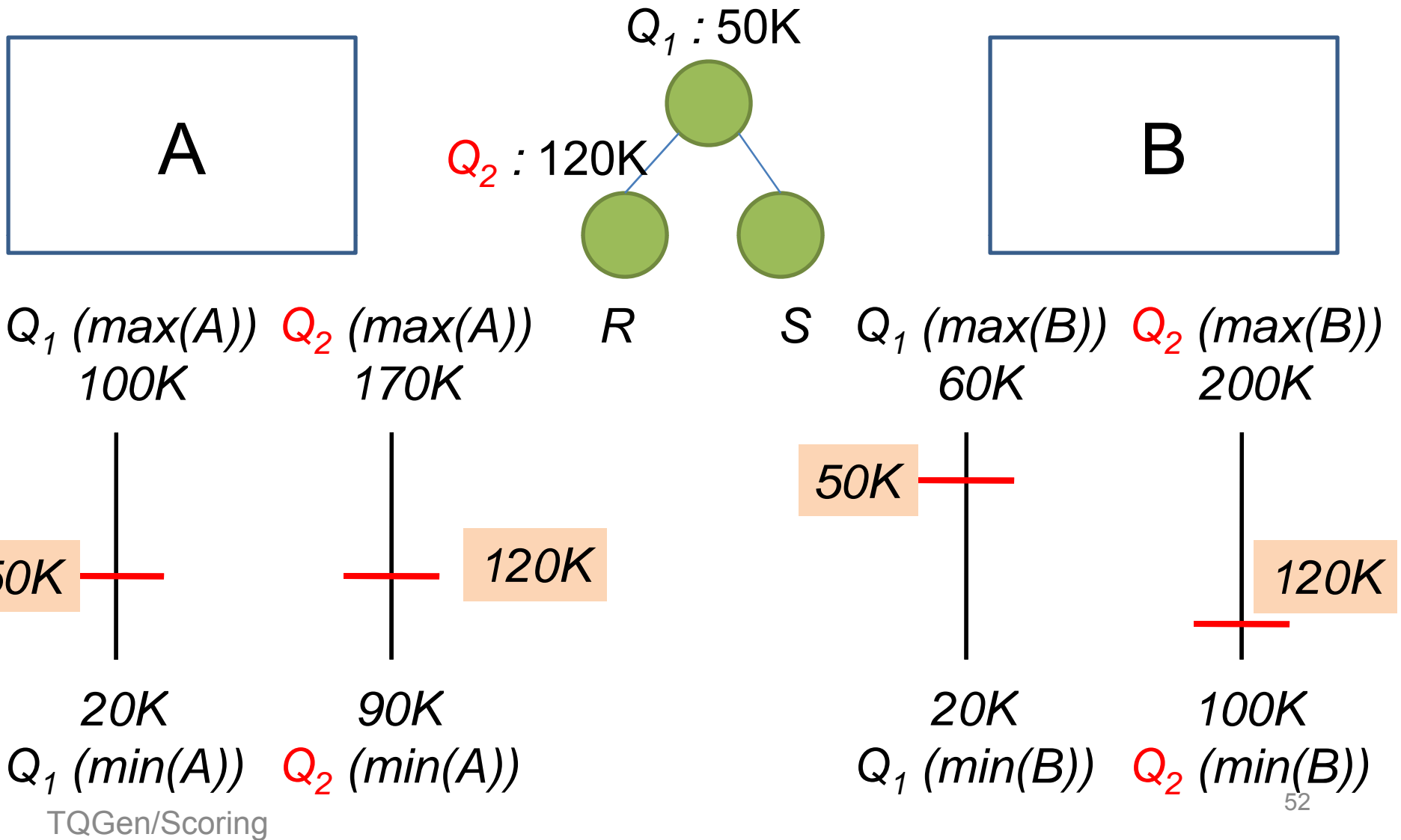
$max(A): x_h, y_h$

$min(A): x_l, y_l$

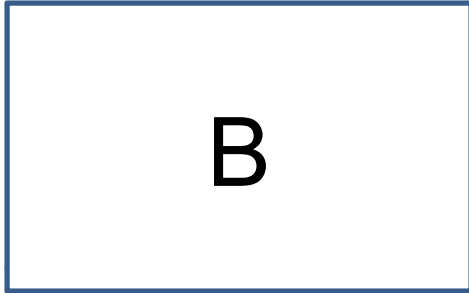
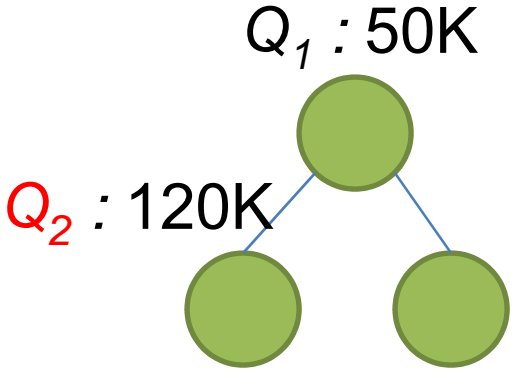
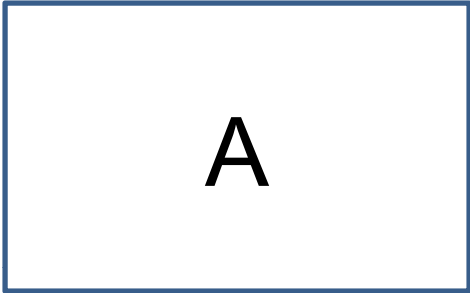
Count number of constraints  $(Q_i, N_i)$  that are bounded.

$$|Q_i(max(A))| \geq N_i \geq |Q_i(min(A))|$$

# Scoring: Breaking Ties



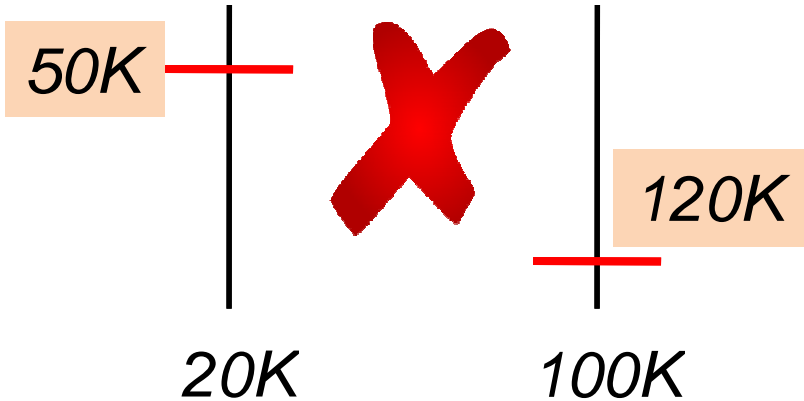
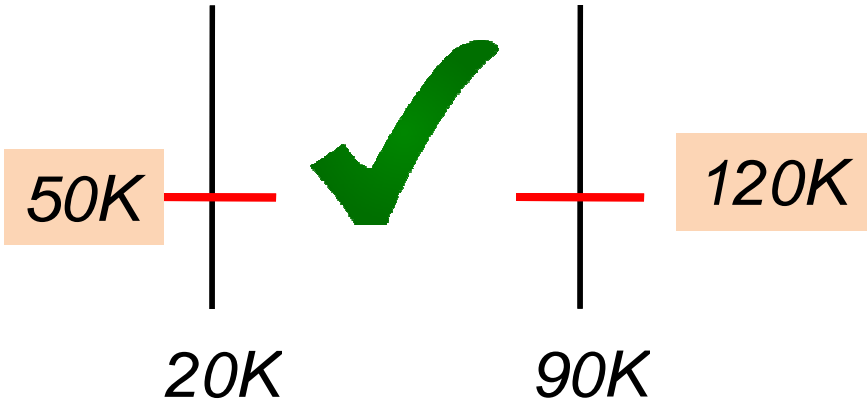
# Scoring: Breaking Ties



$Q_1 (max(A))$   $Q_2 (max(A))$   
 100K 170K

$R$   $S$

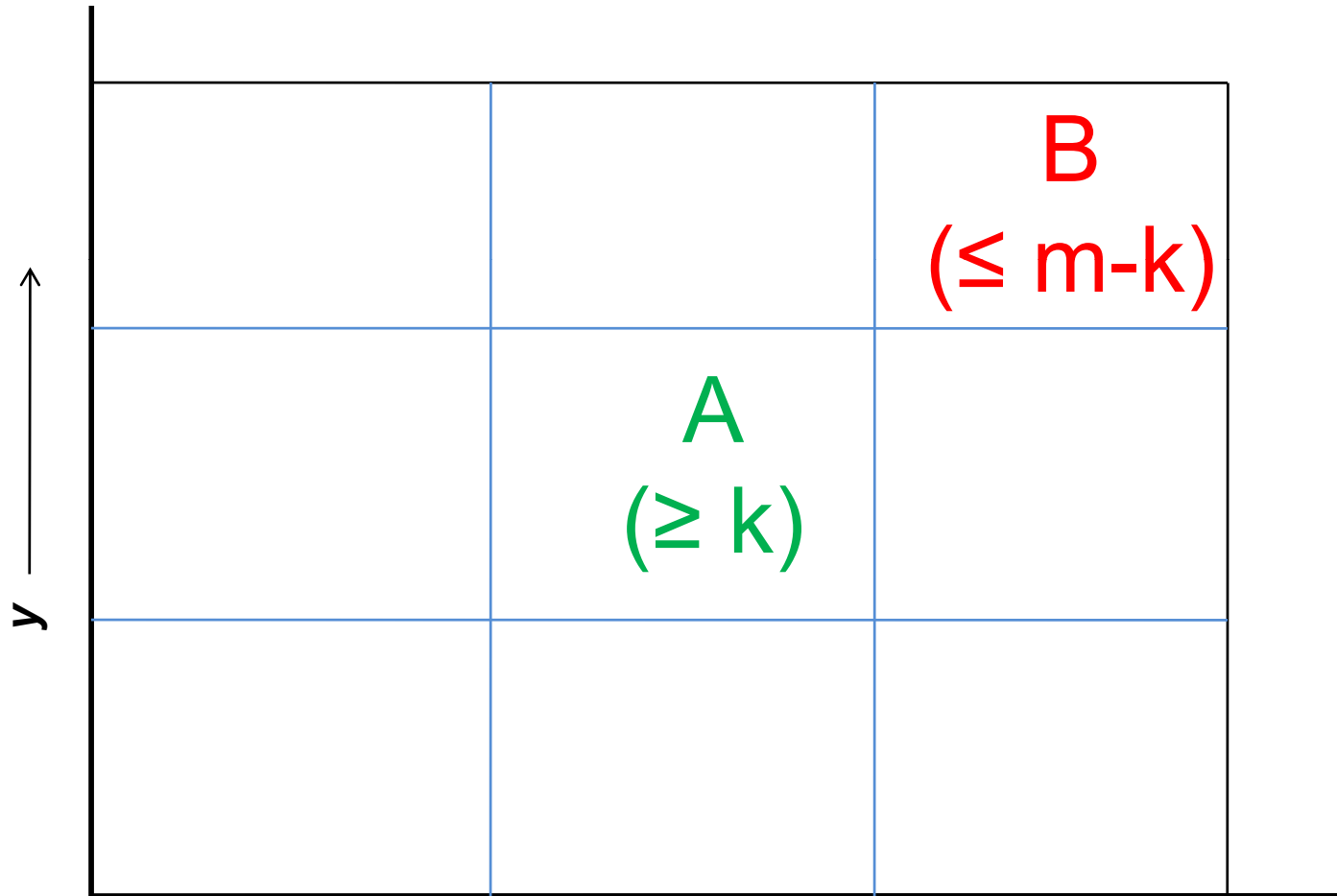
$Q_1 (max(B))$   $Q_2 (max(B))$   
 60K 200K



$Q_1 (min(A))$   $Q_2 (min(A))$

$Q_1 (min(B))$   $Q_2 (min(B))$

# Pruning



# TQGen

- Space Bounding.
- Grid based exploration.
- Scoring functions to guide search, with pruning techniques for efficiency.

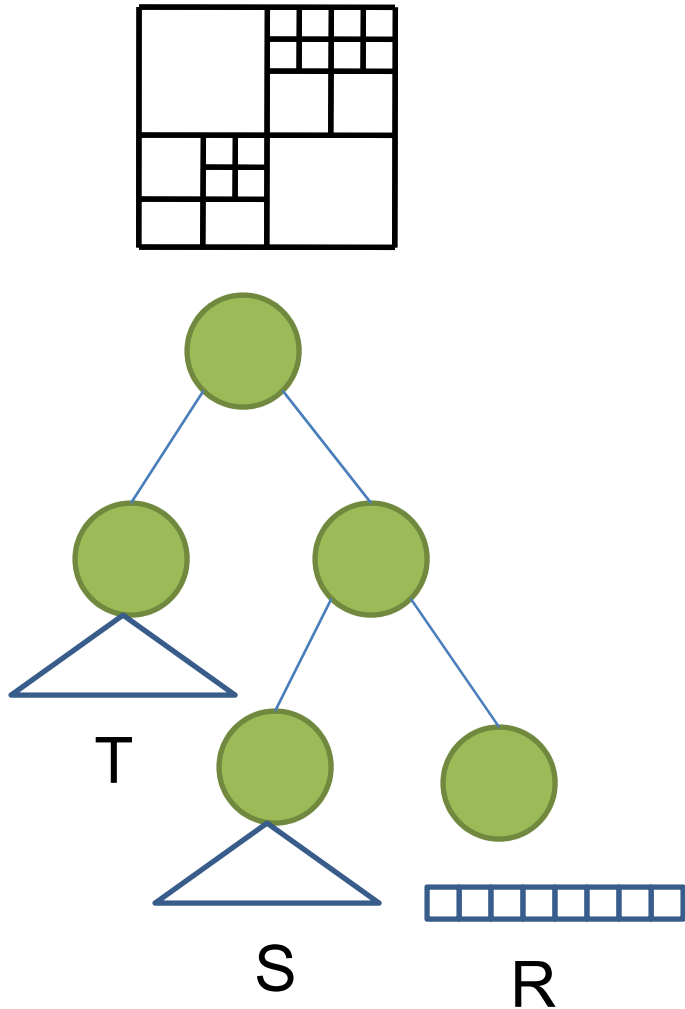
# Outline

- Base Cases
- TQGen Algorithm
- **Cardinality Estimation**
- Implementation and Experiments

# Cardinality Estimation

- Actual query execution can be expensive.
- Optimizer cardinality estimates are cheaper, but potentially incorrect.
- *Sampling based cardinality estimation*

# Sampling

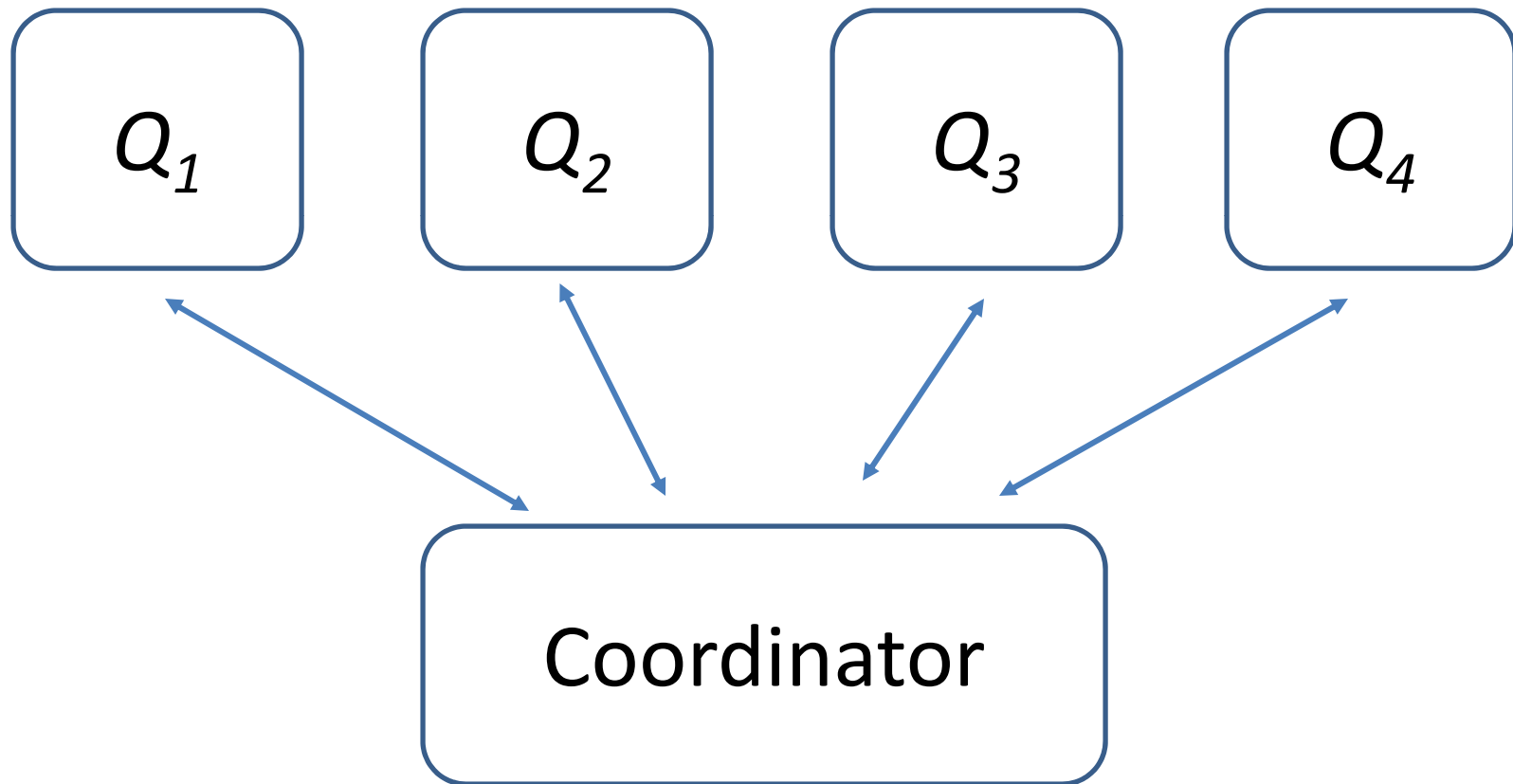


- Index Assisted Sampling.
  - Sampling multiple times may be expensive
- Sample over a superset
  - bounding query
- Store samples in a QuadTree for fast range counting.

# Outline

- Base Cases
- TQGen Algorithm
- Cardinality Estimation
- **Implementation and Experiments**

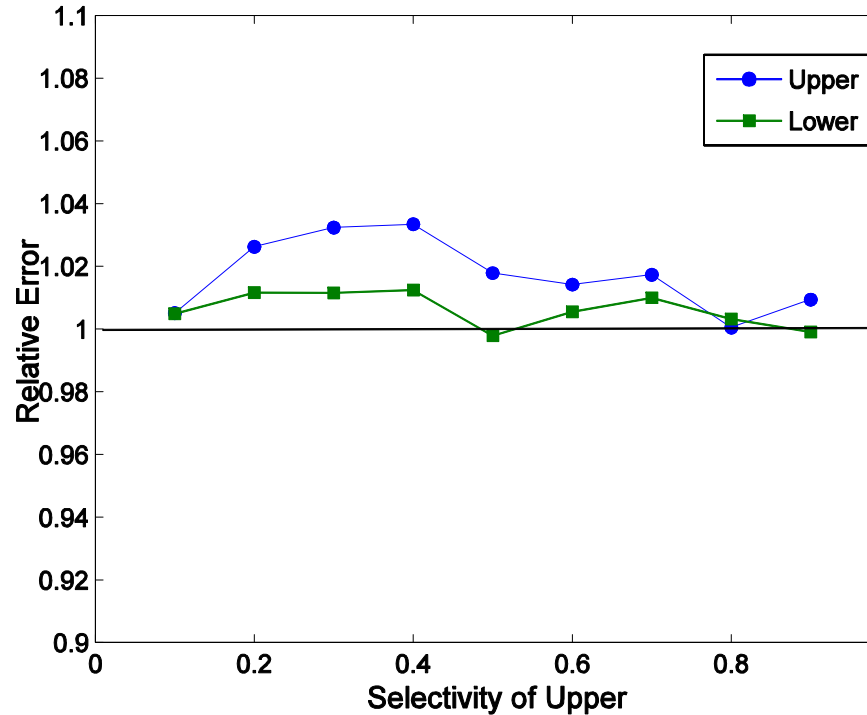
# System Design



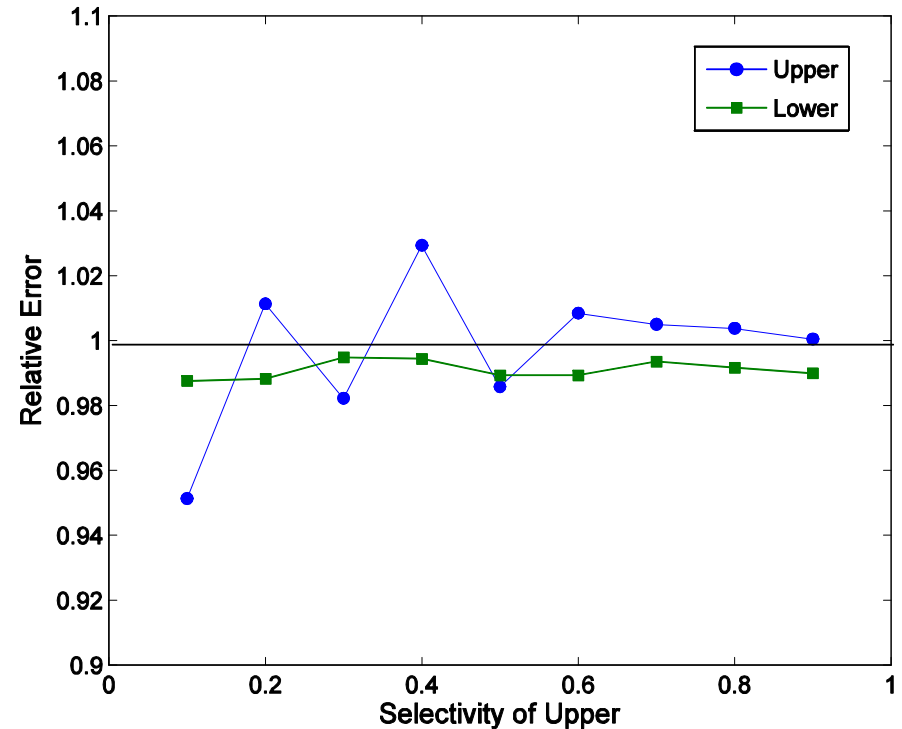
# Experiment Design

- Postgresql + Java.
- 2 Test Databases:
  - TPCH 1 GB Uniform data & Zipfian Skew ( $Z = 1$ )
- Experiments
  - Accuracy
    - Varying target cardinalities
    - Varying #Tables and Constraints
  - Overheads
    - Varying #Constraints and Predicates

# Accuracy: Target Cardinality

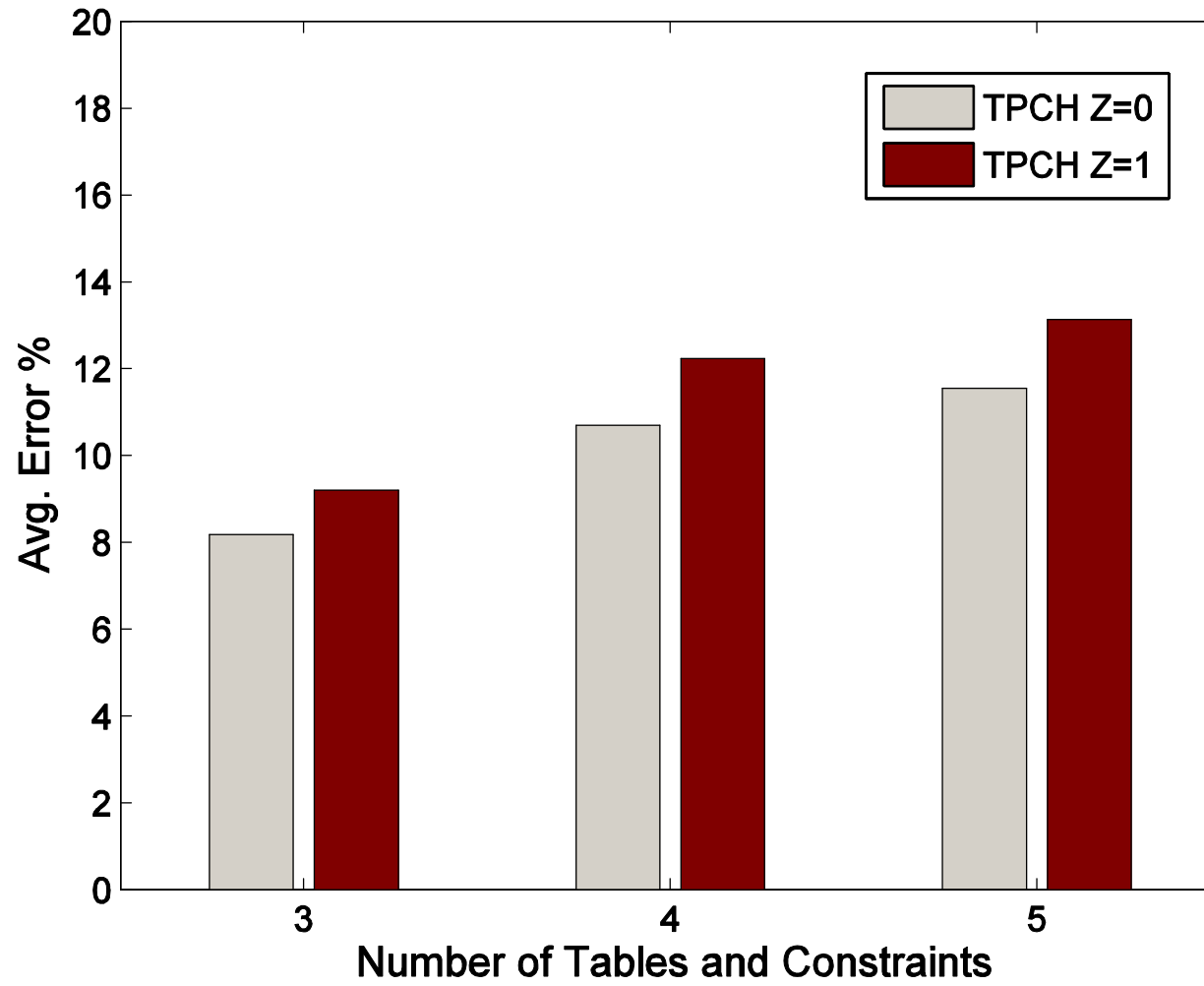


Uniform

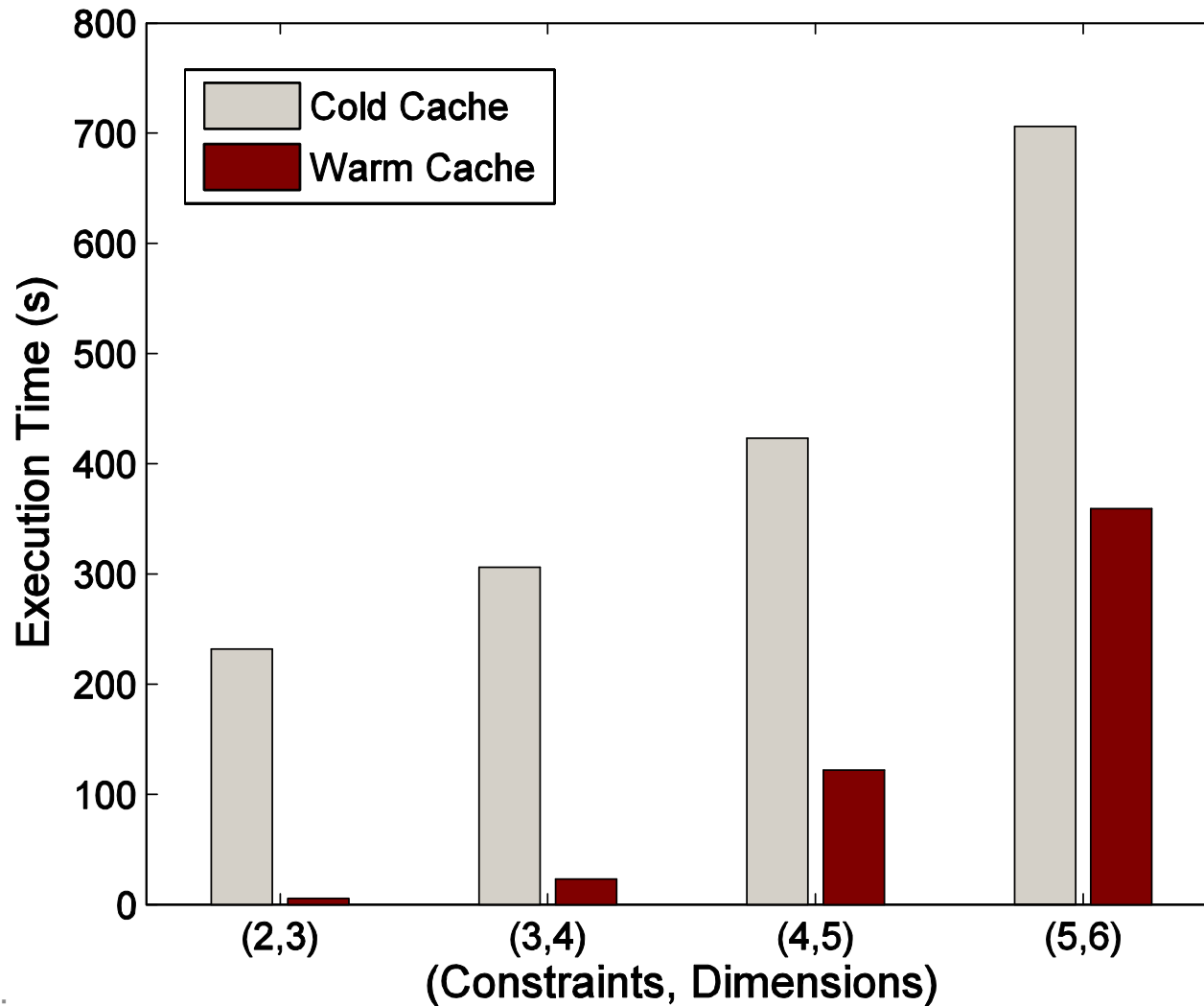


Zipfian  $Z = 1$

# Accuracy: #Tables/Constraints



# Overheads: #Constraints/Predicates



# Summary

- Targeted Query Generation
  - is useful.
  - is computationally hard.
  - is amenable to practical solutions.
  - presents a set of challenging and interesting problems.

**Thank You!**