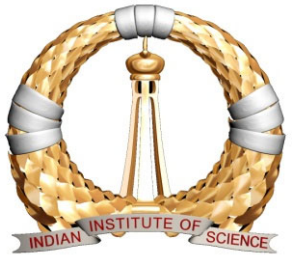


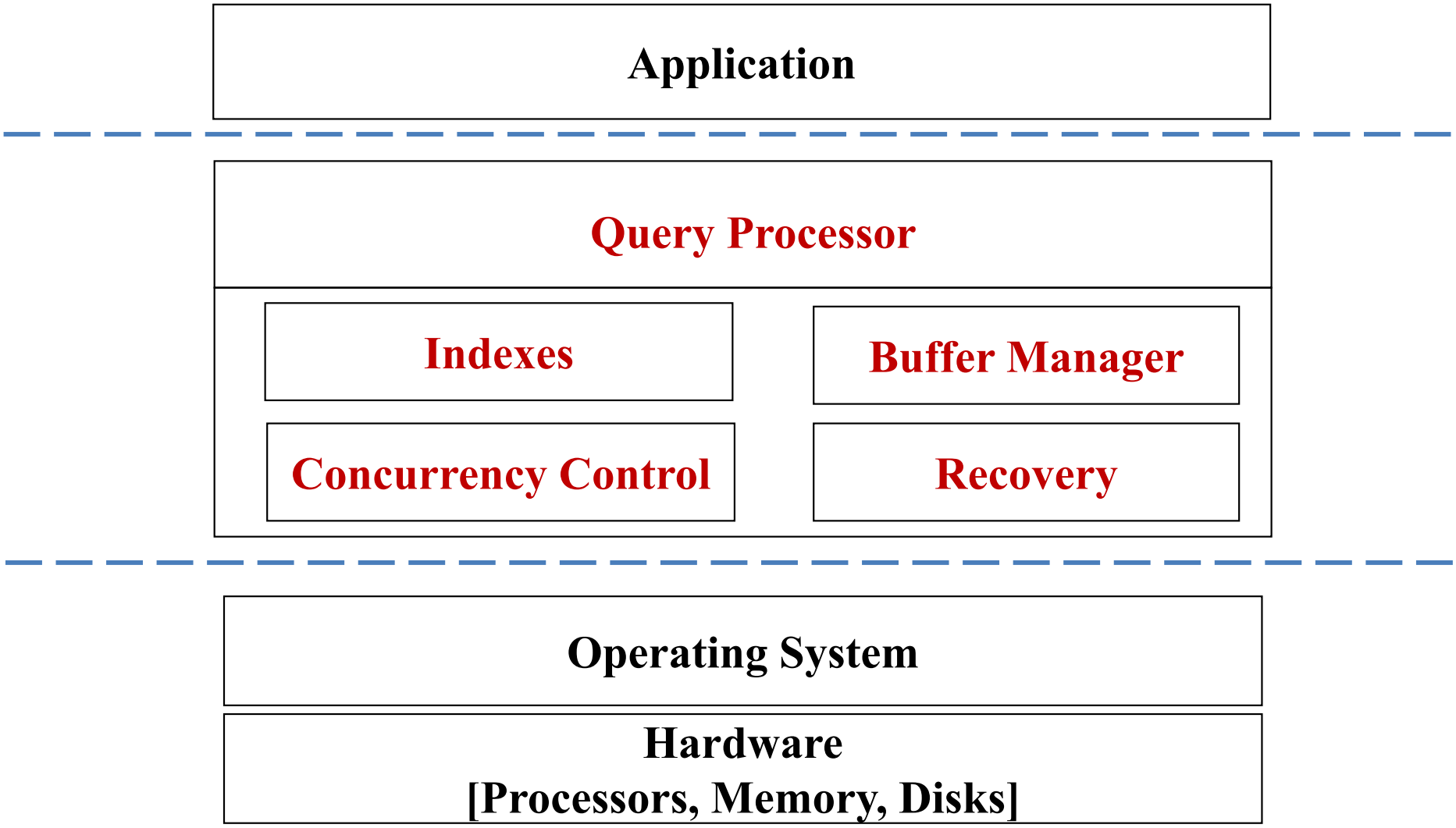
ROBUST QUERY PROCESSING: *Mission Possible!*

Jayant Haritsa
Database Systems Lab
Indian Institute of Science, Bangalore, India

Relational DBMS

- Workhorse of today's Information Industry
  - Commercial
    - IBM DB2, MS SQL Server, Oracle Exadata, HP SQL/MX
  - Public-domain
    - PostgreSQL, MySQL, Berkeley DB
- Extensively researched for over four decades
  - Journals
    - ACM TODS, IEEE TKDE, VLDBJ, …
  - Conferences
    - ACM SIGMOD, IEEE ICDE, VLDB, EDBT, CIKM, …

Typical RDBMS Engine



Design of RDBMS Engines

- Transaction Processing (ACID)
  - WAL/ARIES for Atomicity/Recovery
  - 2PL for Concurrency Control
- Data Access Methods
  - B-trees/Hashing for Large Ordered Domains
  - Bitmaps for Small Categorical Domains
  - R-trees for Geometric Domains
- Memory Management
  - LRU-k (k=2 balances history and responsiveness)
- Query Processing (SQL)
  - "Black Art"

Query Execution Plans

- SQL is a declarative language
  - Specifies ends, not means

```
select   STUDENT.Name, COURSE.Title
from     STUDENT,  COURSE,  REGISTER
where    STUDENT.RollNo = REGISTER.RollNo   and
           REGISTER.CourseNo = COURSE.CourseNo
```

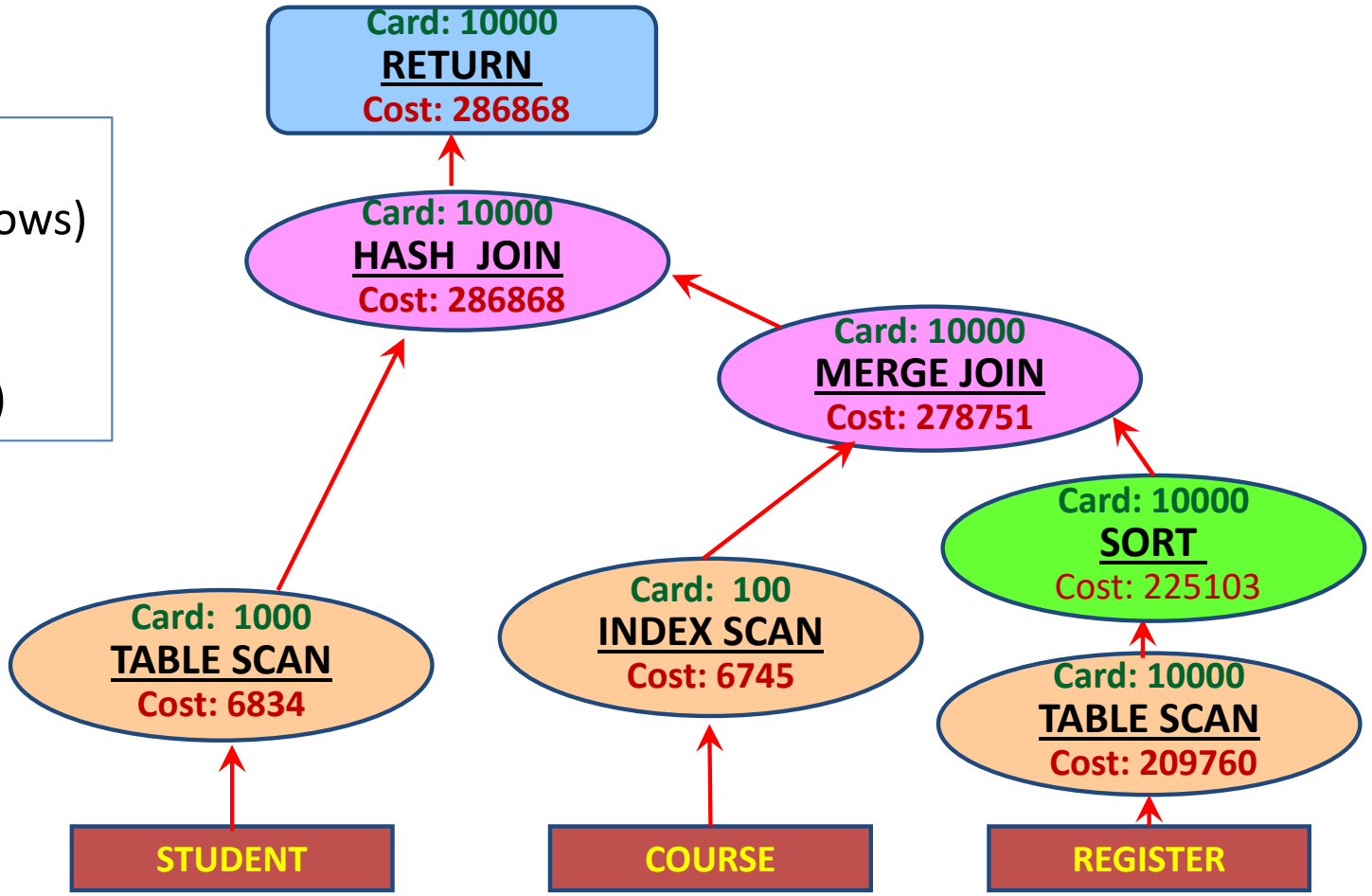
Unspecified: **join order [((S ⋈ R) ⋈ C) or ((R ⋈ C) ⋈ S) ?]**
join technique [Nested-Loops / Sort-Merge / Hash?]

- DBMS query optimizer identifies the optimal evaluation strategy: "query execution plan"

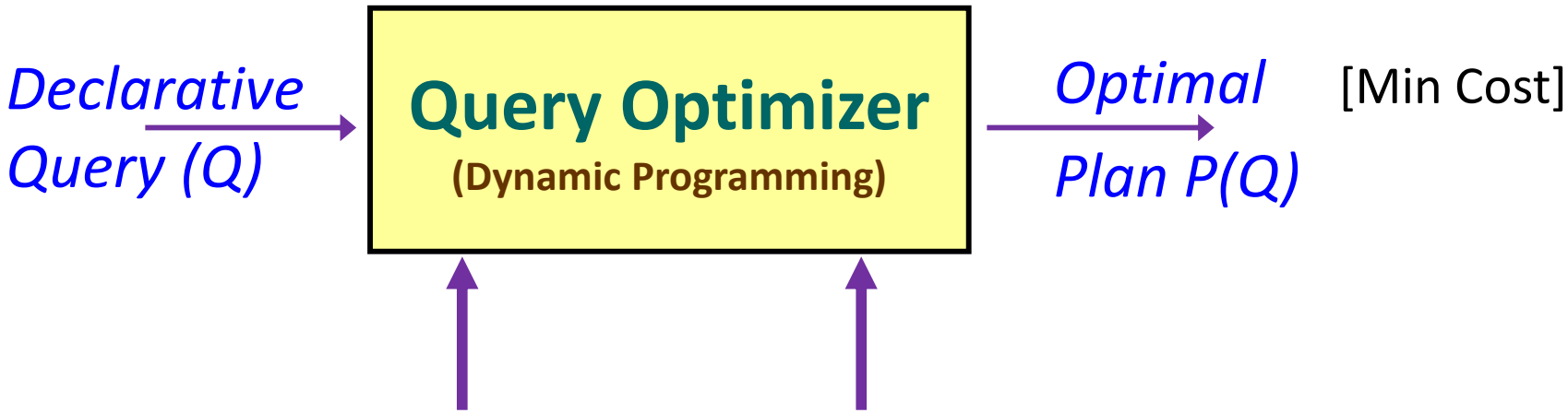
Sample Execution Plan

Card:
Output Cardinality (rows)

Cost:
Execution Cost (time)



Query Optimization Framework



Operator **Execution Cost** Estimation Model

Function of
Hardware and DB Engine

Operator **Output Cardinality** Estimation Model

Function of
Data Distributions and Correlations

Run-time Sub-optimality

The supposedly optimal plan-choice may actually turn out to be highly sub-optimal (e.g. a 1000 times worse!) when the query is executed with this plan. This adverse effect is due to errors in:

(a) cost model

- Reasons: Simple linear models, operator-agnostic features, fixed coefficients, system dynamics …

(b) cardinality model

- Reasons: Coarse statistics, outdated statistics, attribute value independence (AVI) assumption, multiplicative error propagation, query construction, …

What have QP folks been doing all these years?

DB2 Oracle SQL Server

"Elephants"† are highly sensitive animals!

(† Stonebraker-speak for enterprise DBMS)

Cardinality Sensitivity Example

EMPLOYEE

EID	Name	Age
1	Cohen	25
2	Giuliani	25
3	Manafort	25
4	Melania	25
5	Ivanka	25
6	Donald	25
7	Jared	25
….	….	25
10^9	Eric	25

MANAGER

MID	Name	Age
1	Trump	50
2	Pence	50
3	Mnuchin	50
4	Shanahan	50
5	Whitaker	50
6	Bernhardt	50
7	Perdue	50
….	….	50
10^6	Ross	50

EMPLOYEE.AGE ⋈ MANAGER.AGE

- Output cardinality of the join is **ZERO**
- One new employee aged **50** joins the company
- Output cardinality of the join jumps to a **million!**
- No summary mechanism can capture such **"nanoscopic"** changes

Proof by Authority [Guy Lohman, IBM]

Snippet from April 2014 Sigmod blog post on

"Is Query Optimization a "Solved" Problem?"

The root of all evil, the Achilles Heel of query optimization, is the estimation of the size of intermediate results, known as cardinalities. The cardinality model can easily introduce errors of many orders of magnitude! With such errors, the wonder isn't "Why did the optimizer pick a bad plan?" Rather, the wonder is "Why would the optimizer ever pick a decent plan?"

Sound-bites

- Dave DeWitt: Query optimizers do terrible job of producing good plans without a lot of hand tuning.
- Surajit Chaudhuri: Current state is unsatisfactory with known big gaps in the technology.
- Little difference between worst-case and average-case in Query Processing

Prior DB Research (lots!)

- Sophisticated estimation techniques
  - SIGMOD 1999/2010, VLDB 2001/2009/2011, …, CIDR 2019
  - e.g. wavelet histograms, self-tuning histograms, deep learning
- Selection of stable plans
  - SIGMOD 1994/2005/2010, PODS 1999/2002, VLDB 2008, …, VLDB 2017
  - e.g. Variance-aware plan selection
- Runtime re-optimization techniques
  - SIGMOD 1998/2000/2004/2005, …, ICDE 2019 [Stonebraker et al]
  - e.g. POP (progressive optimization) [35], RIO (re-optimizer) [6]

Several novel ideas and formulations,
but are they robust?

Is there any hope?

Over last decade, several promising advances that collectively promise to soon make robustness a contemporary reality – we survey these techniques in the rest of the tutorial …

Thanks

Gratefully acknowledge presentation material provided by

- Renata Borovica-Gajic (U of Melbourne, Australia)
- Goetz Graefe (Google Madison Labs, USA)
- Thomas Neumann (TU Munich, Germany)
- Wolf Roediger (Tableau, Germany)
- Wentao Wu (Microsoft Research, USA)
- Srinivas Karthik (IISc Bangalore, India)
- Andreas Kipf (TU Munich, Germany)
- Zongheng Yang (UC Berkeley, USA)

QP Robustness

Importance of Robustness

- Dagstuhl Seminars
  - 2010 (#10381), 2012 (#12321), 2017 (#17222)
- ICDE 2011 panel on Robust Query Processing
- Immediate relevance to database vendors
- Huge impact on database users and customers
- Critical for Big Data world!

ROBUSTNESS DEFINITION

- Multiple perspectives, no consensus
  - If worst-case performance is improved at the expense of average-case performance, is that acceptable?
  - Is it to be defined on a query instance basis, or “in expectation”?
  - …
- Ultimately, robustness definition is application dependent
- Graceful performance profile – no “cliffs”
- Seamless scaling with workload complexity, database size, distributional skew, join correlations
- Provable guarantees on worst-case performance (relative to an offline ideal that makes all the right decisions)

TUTORIAL OUTLINE

- Stage 1: Robust Operators
- Stage 2: Robust Plans
- Stage 3: Robust Query Execution
- Stage 4: Robust Cost Models
- Stage 5: Machine Learning Approaches
- Stage 6: Future Research Directions

Stage 1: Robust Operators

Approaches

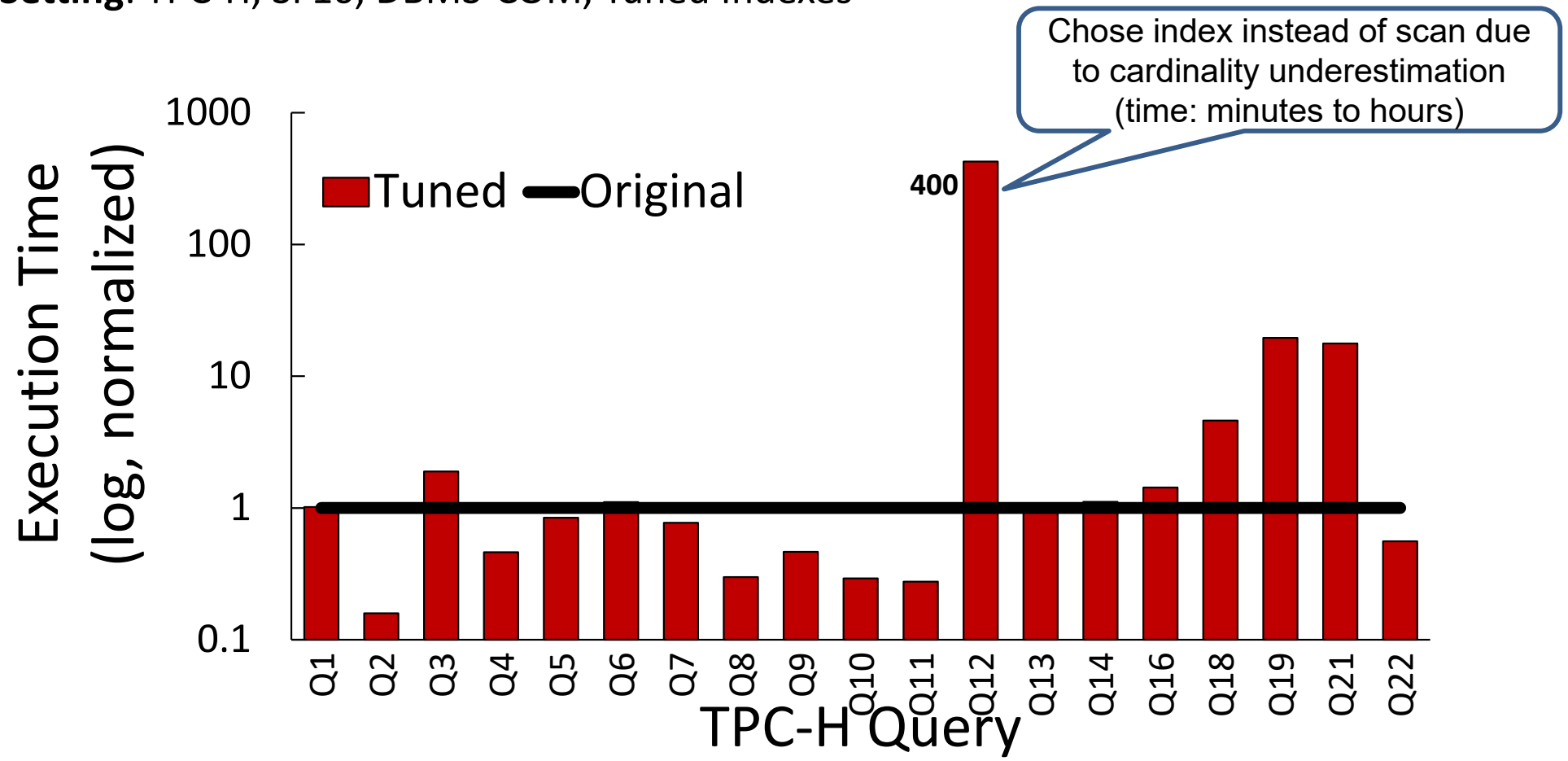
- Unified operators
  - *Basic Idea: If choice is eliminated, cannot make mistakes, by definition! The key challenge is retaining, in the absence of choice, comparable performance to the multi-choice environment.*
  - Smooth Scan (ICDE 2015, VLDBJ 2018 [7])
    - Unifying Table Scan, Index Scan
  - G-join (CS R&D, 2012 [17])
    - Unifying Nested-loops, Sort-merge, Hash-join
- Scaling operators
  - Flow-join (ICDE 2016 [39])
    - Broadcast "heavy hitter" tuples to handle skew in distributed systems

Smooth Scan [7]

(Morph between Sequential Scan **and** Index Scan**)**

Sub-optimal Access Paths: Example

Setting: TPC-H, SF10, DBMS-COM, Tuned Indexes

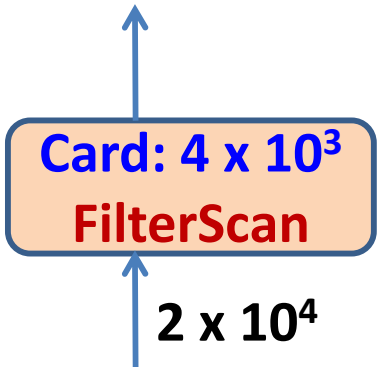


Selectivity

Selectivity = Normalized Cardinality [0 to 100%]

sel = (Output Rows / Max Output Rows) * 100

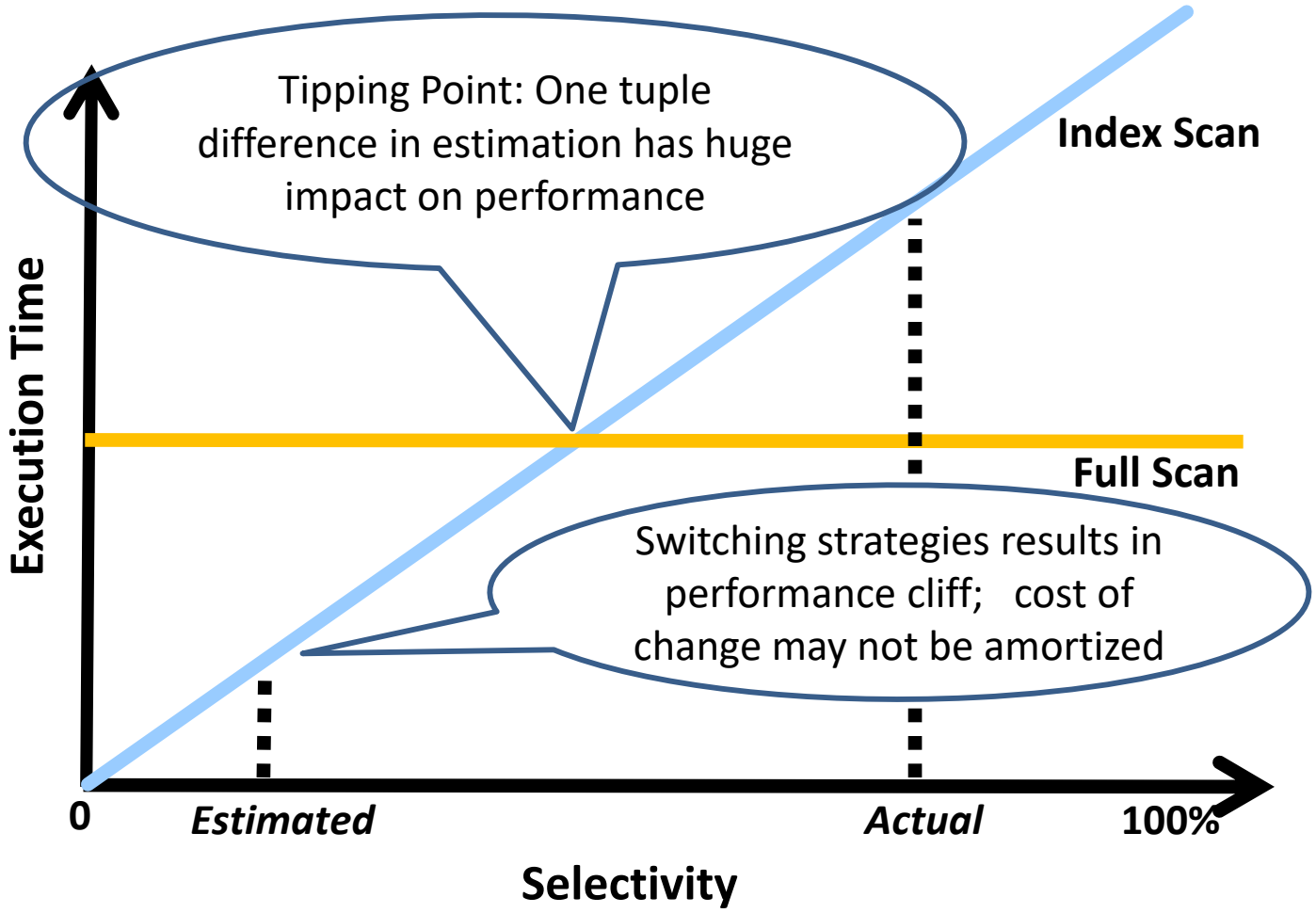
COURSE.fees < 1000



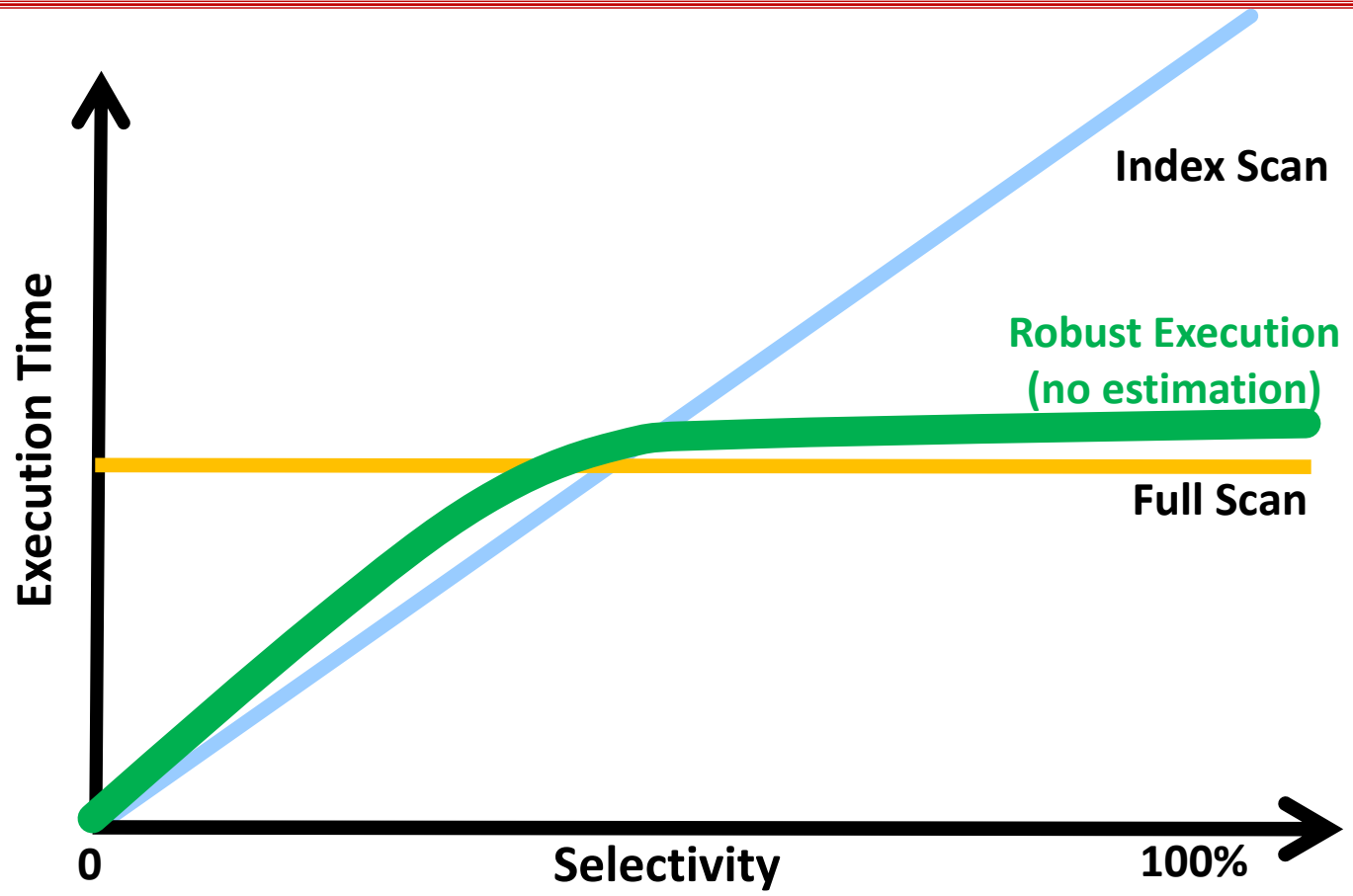
sel = (4×10^3 / 2×10^4) * 100

= 20 %

Access path selection problem



Quest for robust access paths



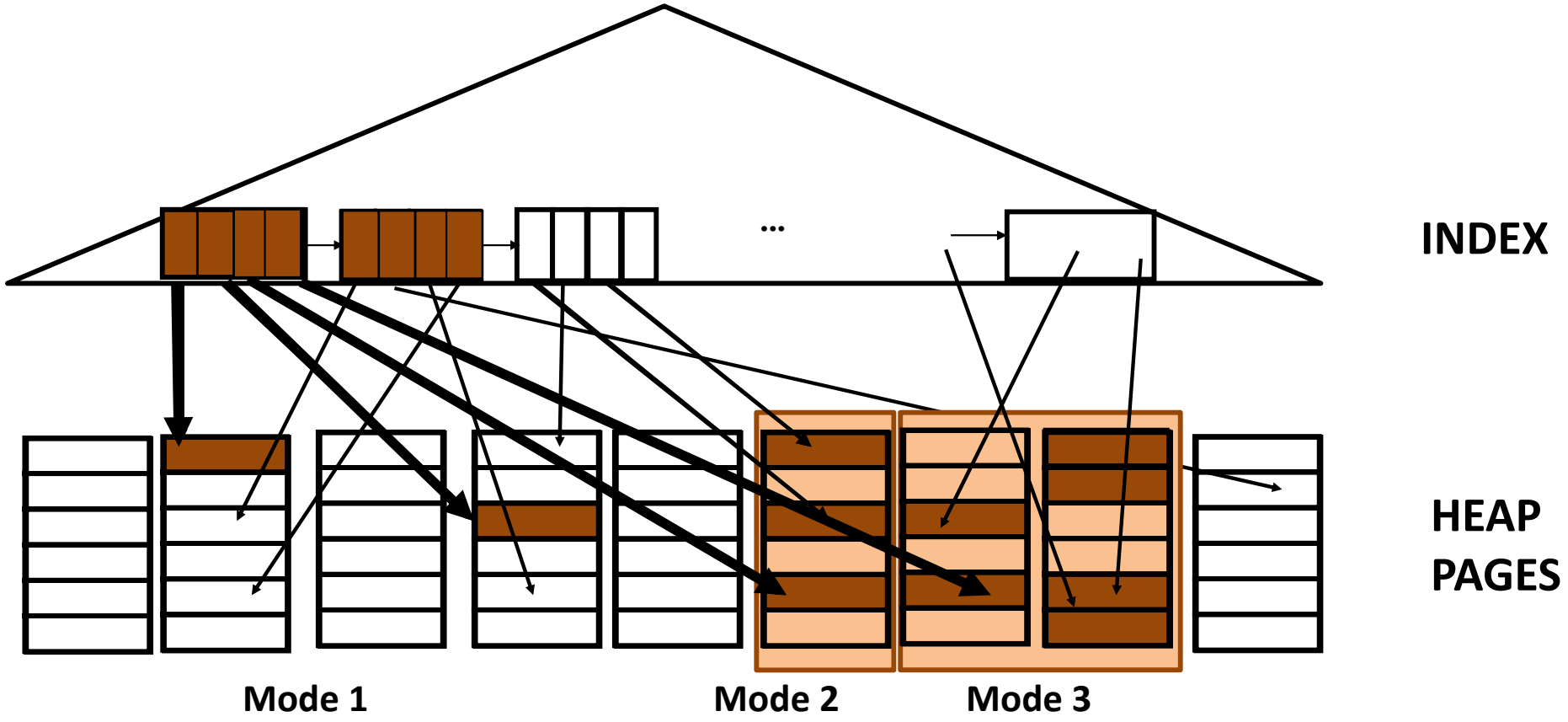
Near-optimal (= min(IS,FS)) throughout entire selectivity range

Smooth Scan in a nutshell

- Statistics-oblivious access path
- Learn result distribution at run-time
- Adapt as you go

Morphing Mechanism Modes

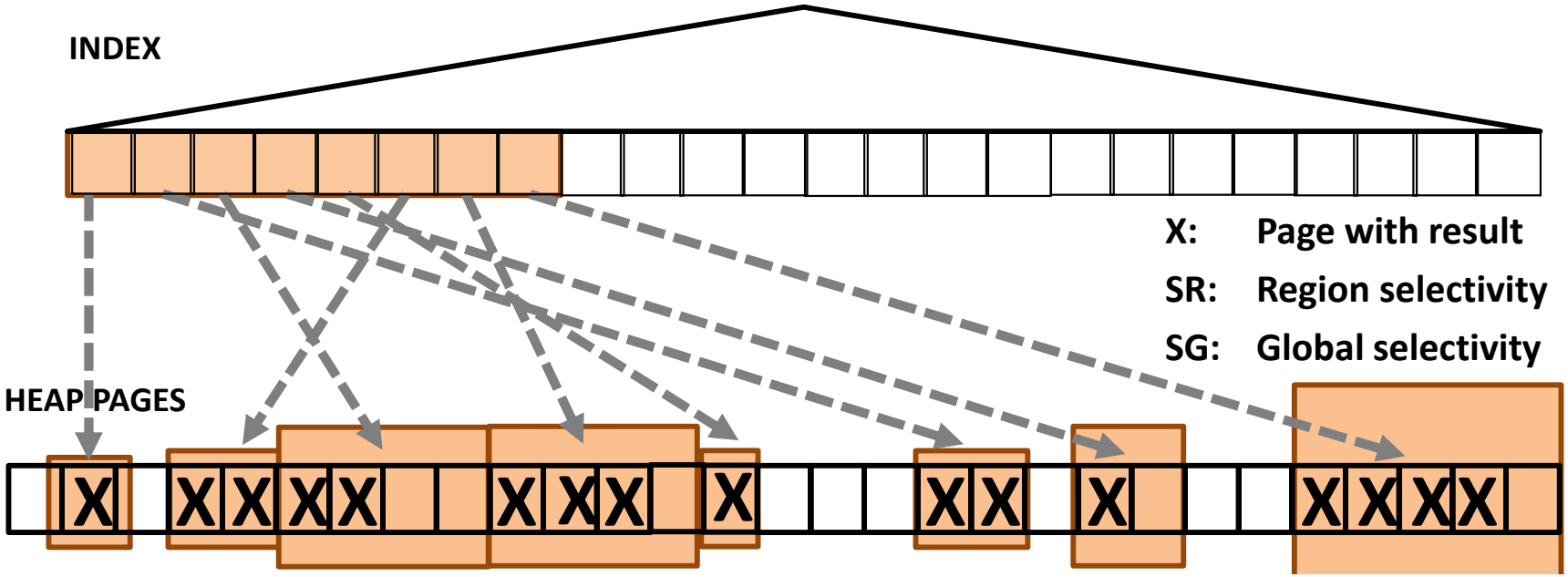
1. **Index Access:** Traditional index access
2. **Entire Page Probe:** Index access probes entire page
3. **Gradual Flattening Access:** Probe adjacent region(s)



Morphing policies

- Greedy
- Selectivity Increase
- Elastic

Selectivity increase → Mode Increase
SEL_region ≥ SEL_global
Selectivity decrease → Mode Decrease
SEL_region < SEL_global



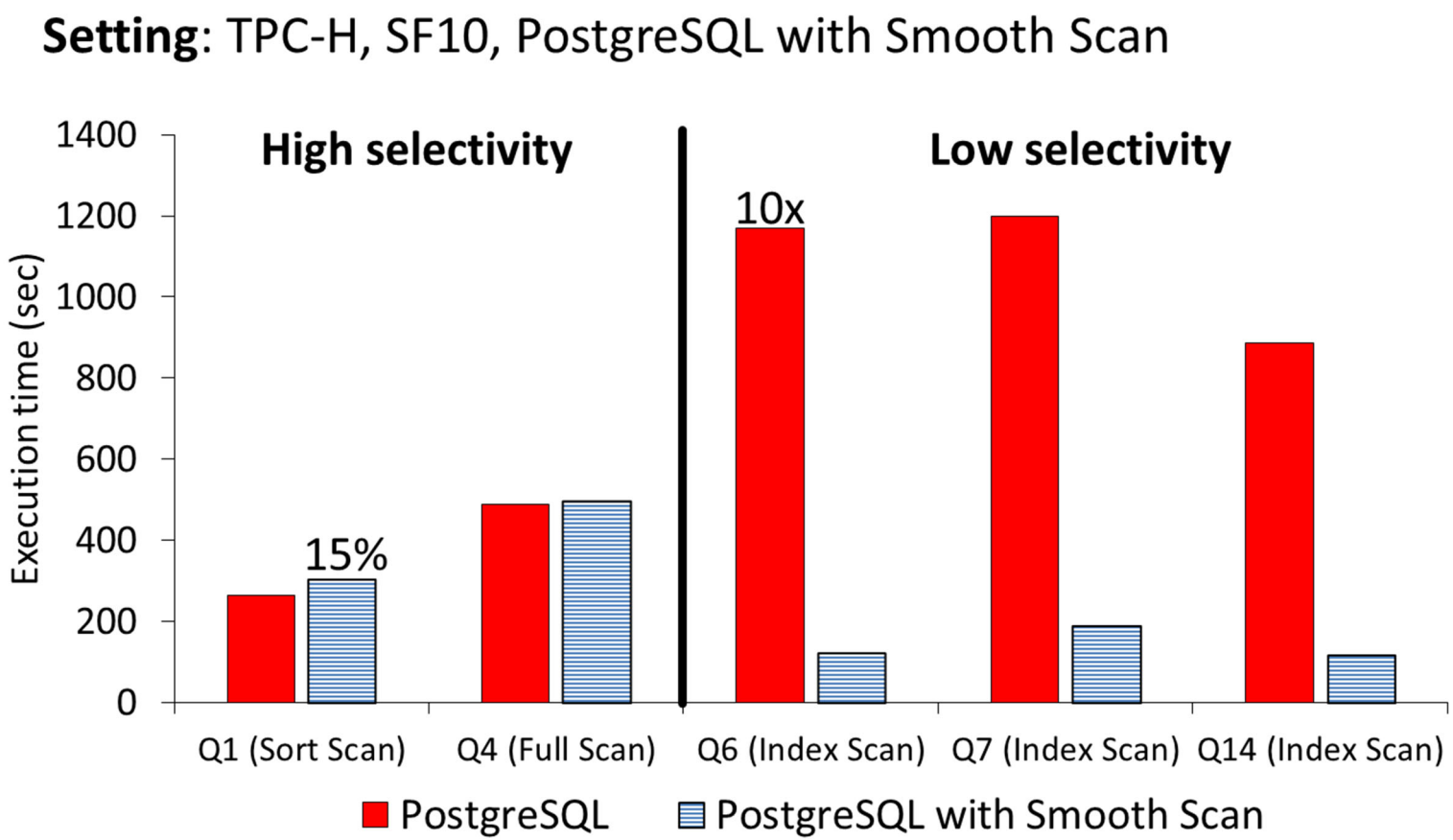
Region snooping = Selectivity driven adaptation

Smooth Scan benefits

	Index Scan	Full Scan	Sort Scan	Smooth Scan
Avoid repeated accesses	✗	✓	✓	✓
Fast sequential I/O	✗	✓	✓	✓
Avoid full table read	✓	✗	✓	✓
Tuples pipelining	✓	✓	✗	✓

Sort Scan: Get all qualifying RIDs from the index, sort them, and then sequentially retrieve the records.

TPC-H with Smooth Scan



Robust execution for all queries

Performance Guarantee

Ideal: SortScan without Sorting Cost – i.e. sequentially read only the relevant pages.

$$\frac{SmoothScan}{Ideal} \leq (1+\frac{\text{rand\_io\_cost}}{\text{seq\_io\_cost}})$$

For representative HDD parameters, factor is 11, while for SSD, factor is 6.

Limitations

- Several book-keeping data structures required to maintain result semantics (duplicates/ordering)
  - Page ID cache (to not process page twice)
  - Tuple ID Cache (to not produce same tuple twice)
  - Result Cache (for ordered output)
  - Memory Management (for above structures)
- Requires changes to database engine internals

G-join: Generalized Join [17]

(Morph across Indexed-NL Join, Sort-Merge Join, Hybrid-Hash Join)

Comparative Algorithm Strengths

	INL Join	SM Join	HH Join	G-join
Sorted inputs		✓		✓
Indexed input	✓			✓
Input size differences			✓	✓

Basic Idea

- Implement Sort-Merge using concepts from Hash-Join
- If inputs are already sorted, just do Merge Join
- If inputs are not sorted, create internally sorted runs (using replacement selection) as usual for both inputs, but do not carry out merging steps.
  Instead, similar to hash partitions, store "key-covering pages" from the small-input (**R**) in a buffer pool, and a single buffer page for the large-input (**S**). Dynamically expand the **R** buffer pool until it key-covers the buffer page of **S** – then join the memory-resident pages. After this is done, bring the next **S** page into memory. Shrink the **R** buffer pool if any page goes below the key coverage range.

G-join: Phase 1

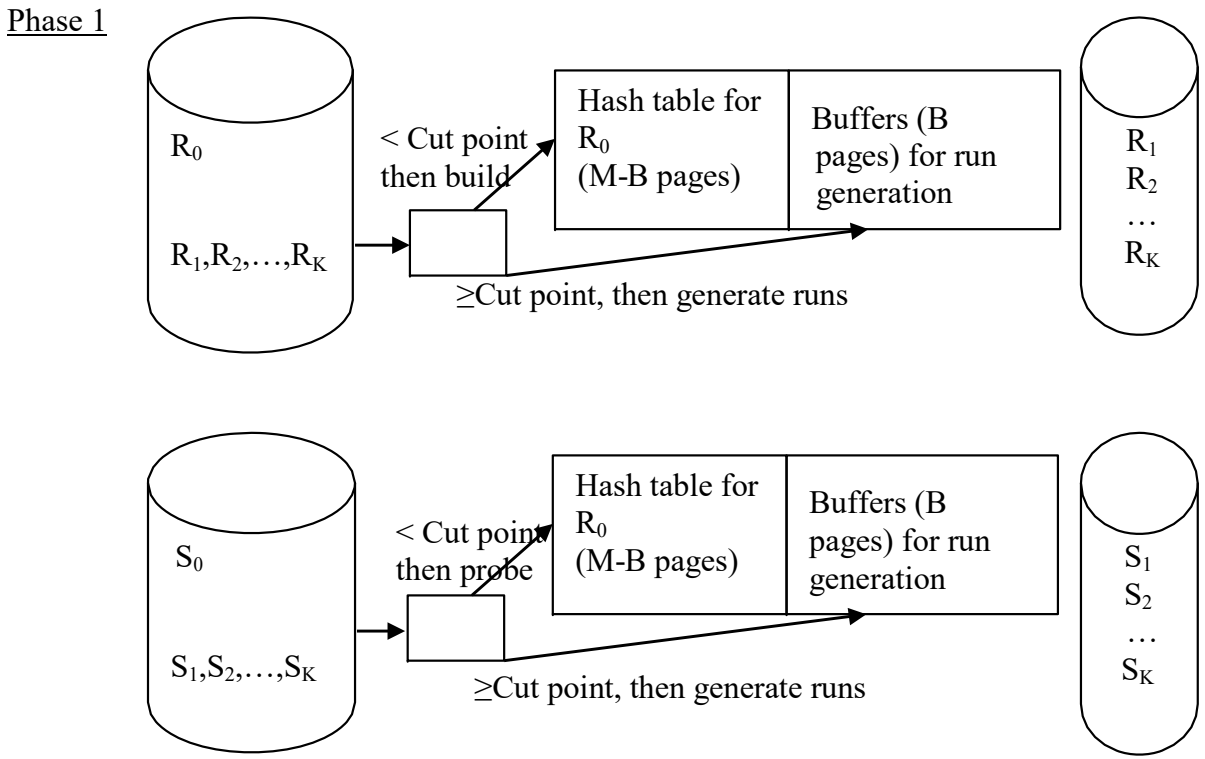


Figure 3.1 Phase 1 of G-Join

G-join: Phase 2

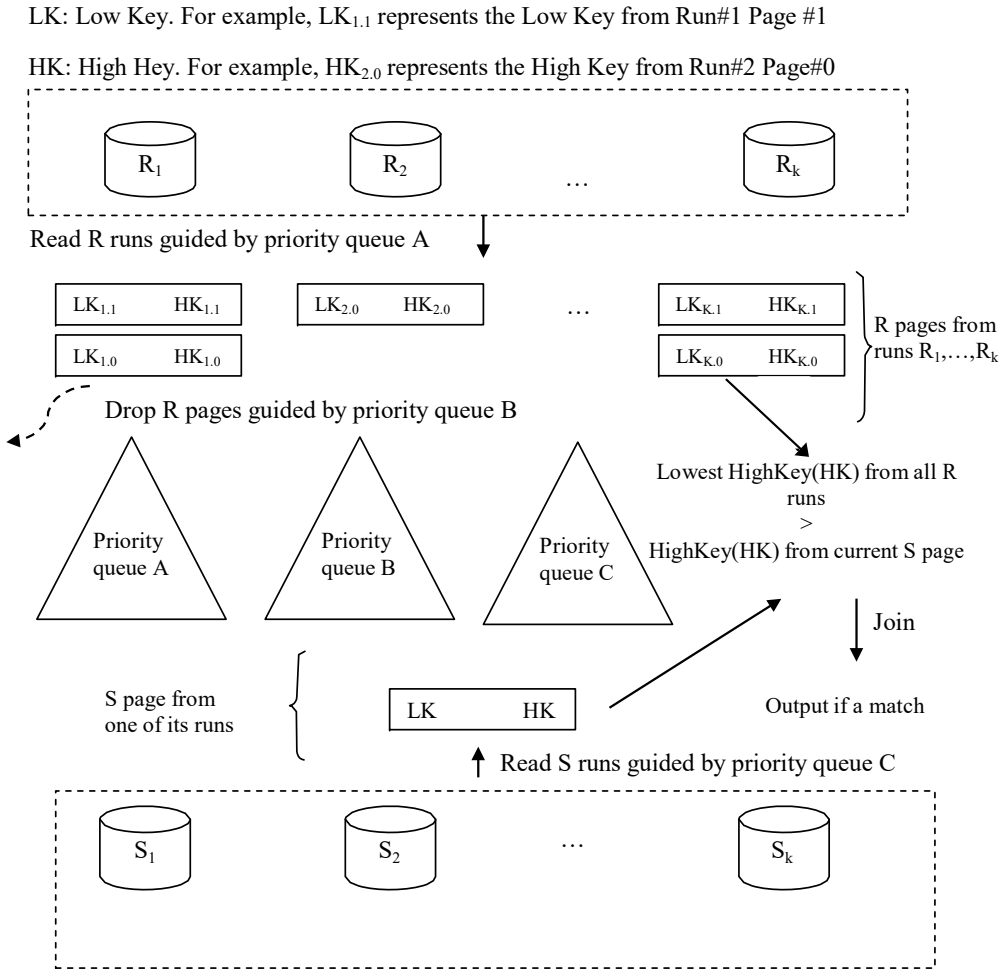
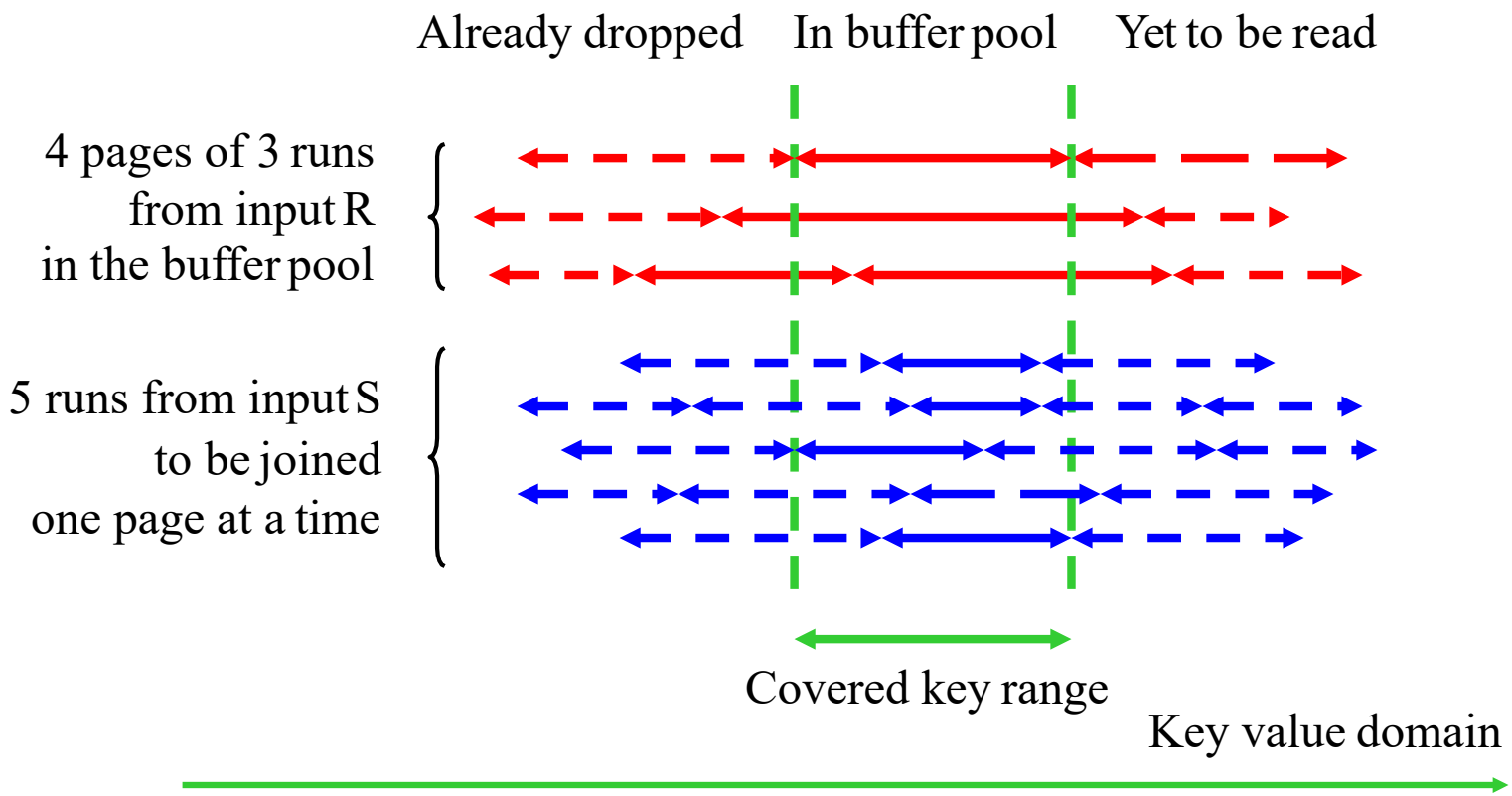
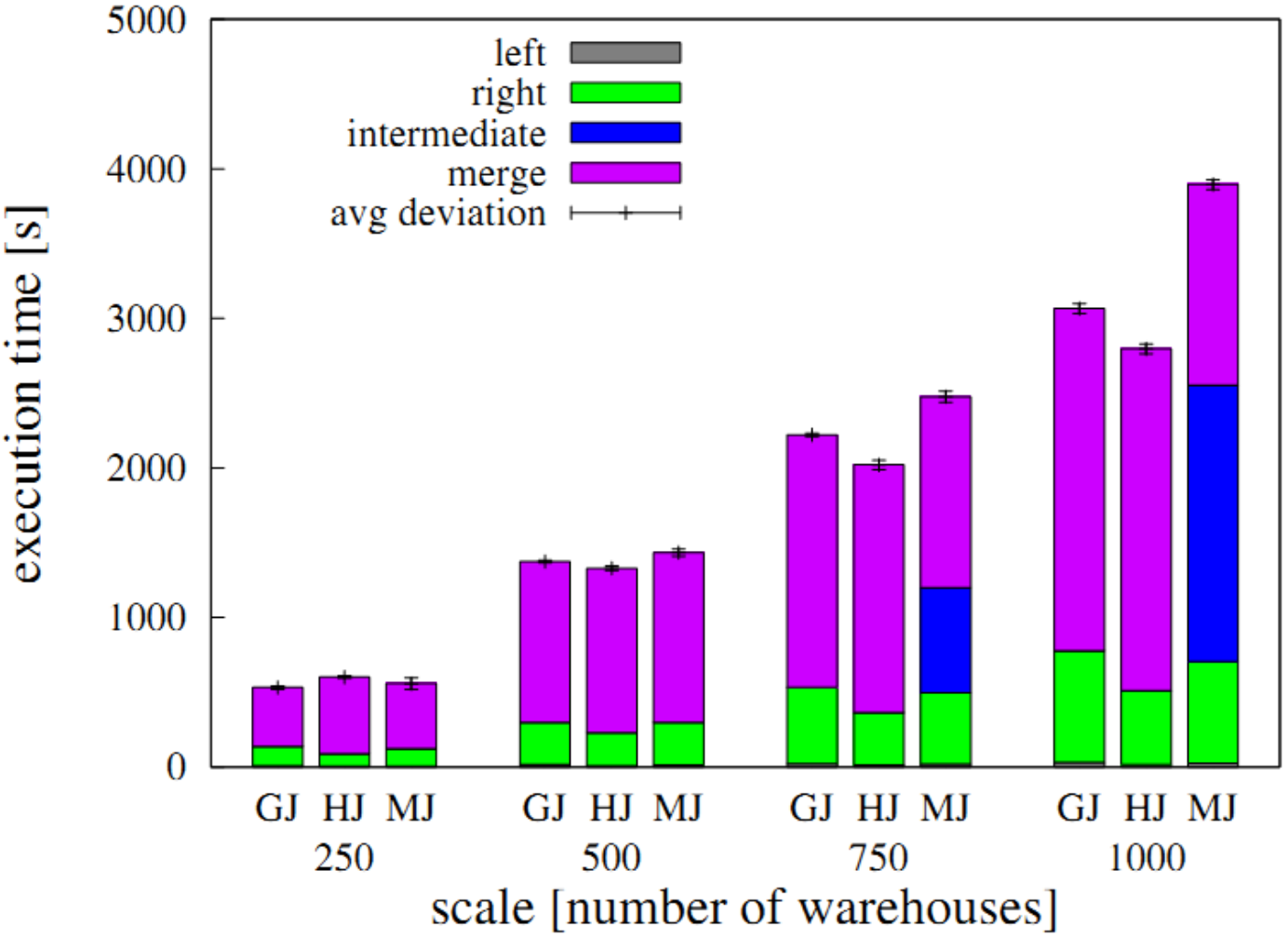


Figure 3.2 Phase 2 of G-Join

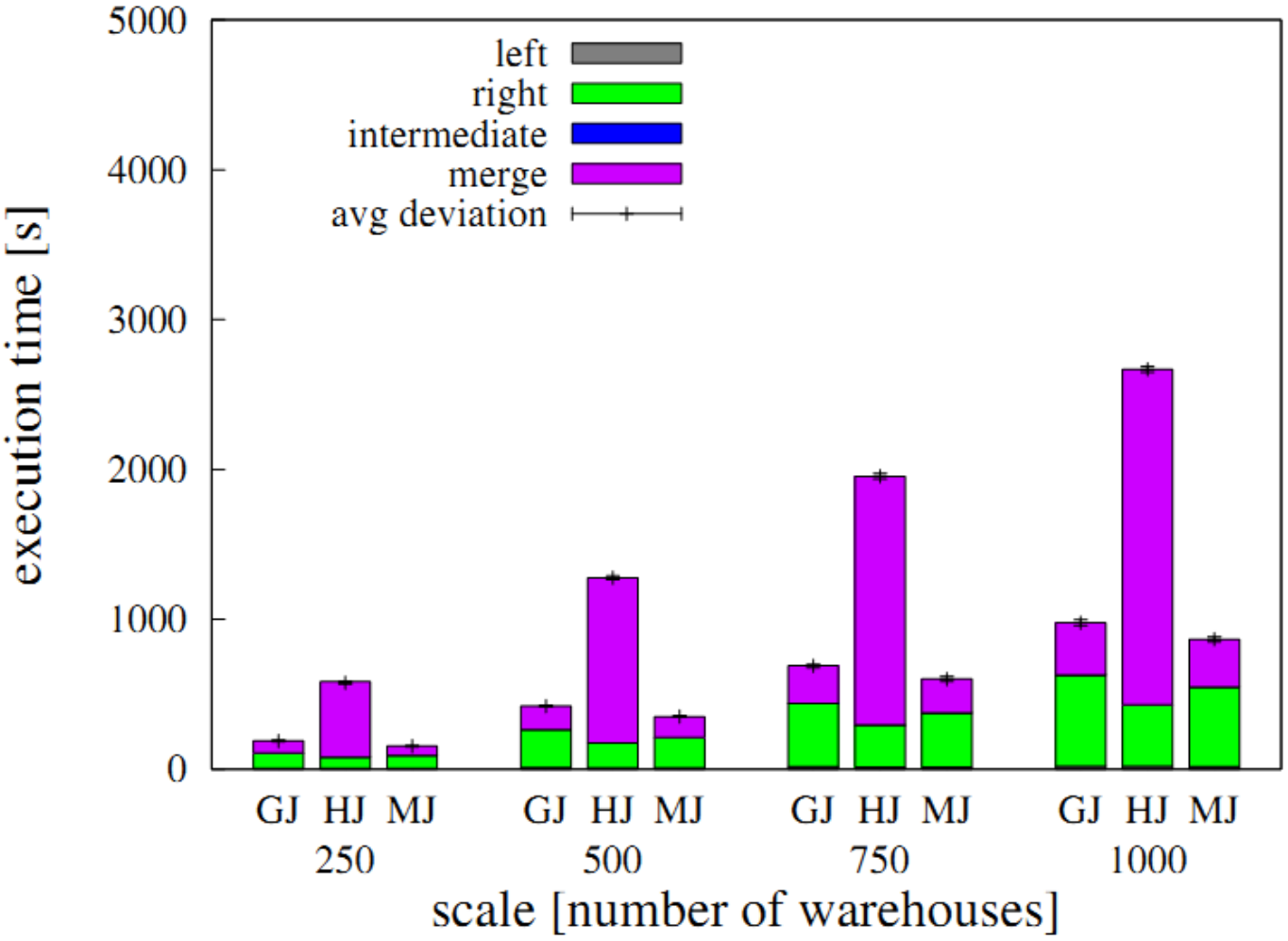
Merge algorithm illustrated



Unsorted inputs



~Sorted inputs



Robust Performance

- Performance Guarantee:
  - First-cut theoretical analysis shows rough equivalence to best of existing algorithms
- Limitations:
  - Skew in sizes of runs and skew in key value distribution can adversely impact performance
- Similar "unified" algorithms available for grouping and set operations. [17]

Stage 2: Robust Plans

APPROACHES

- Least Expected Cost (PODS 99 [11], PODS 02 [12])
  - estimate Distributions instead of Values for parameters
- Cost-Greedy (VLDB 2007 [18])
  - reduce parametric optimal set of plans (POSP) space into low-cardinality ("anorexic") approximation featuring relatively stable plans
- SEER (VLDB 2008 [19])
  - reduce POSP space into anorexic approximation that can handle arbitrary estimation errors

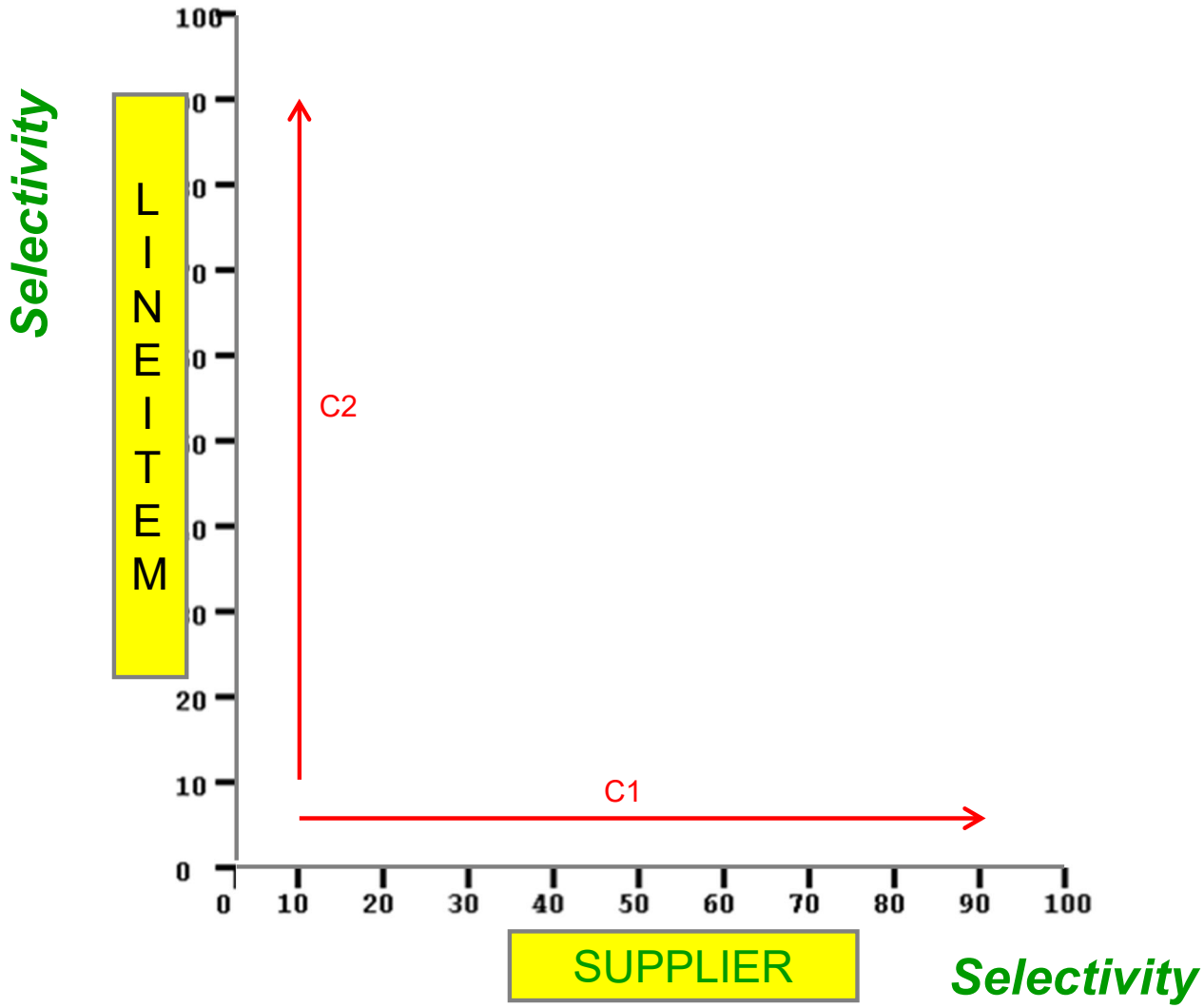
Cost-Greedy [18]

Query Template [Q8 of TPC-H]

Determines how the market share of Brazil in the USA has changed over 1995-1996 for Steel parts

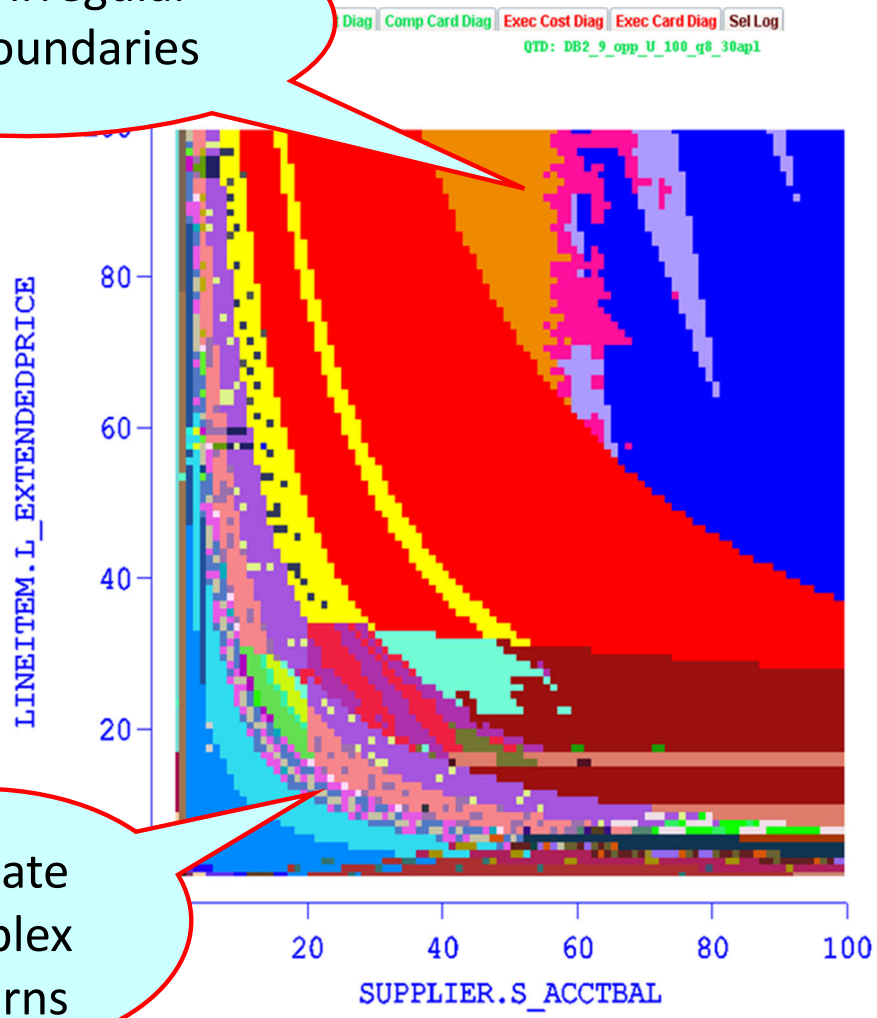
```
select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from   ( select   YEAR(o_orderdate) as o_year,
                  l_extendedprice * (1 - l_discount) as volume, n2.n_name as nation
         from     part, supplier, lineitem, orders, customer, nation n1, nation n2, region
         where    p_partkey = l_partkey and s_suppkey = l_suppkey and
                  l_orderkey = o_orderkey and o_custkey = c_custkey and
                  c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey  and
                   r_name = 'AMERICA' and s_nationkey = n2.n_nationkey and
                  o_orderdate between '1995-01-01' and '1996-12-31' and
                  p_type = 'ECONOMY ANODIZED STEEL'
                  and s_acctbal ≤ C1 and l_extendedprice ≤ C2
       ) as all_nations
group by o_year
order by o_year
```

POSP Plan Diagram



POSP Plan Diagram

Highly irregular plan boundaries



# of plans: 76

Extremely fine-grained coverage (P76 ~ 0.01%)

Intricate Complex Patterns

Problem Statement

Can the plan diagram be recolored with a smaller set of colors (i.e. some plans are "swallowed" by others), such that

Guarantee:

No query point in the original diagram has its estimated cost increased, post-swallowing, by more than λ percent (user-defined)

CostGreedy

- Optimal plan diagram reduction (w.r.t. minimizing the number of plans/colors) is NP-hard
  - through problem-reduction from classical Set Cover

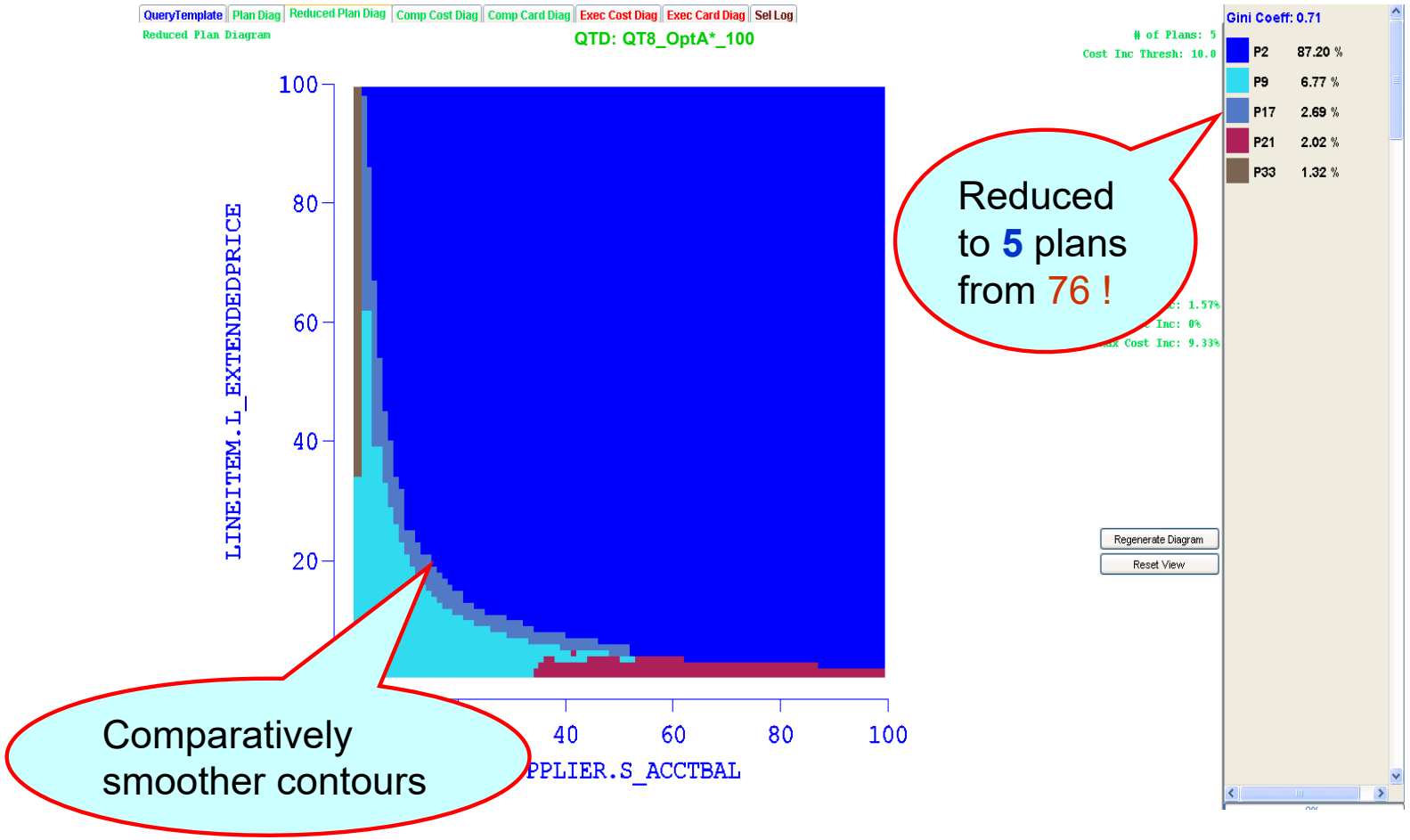
- CostGreedy is a greedy heuristic-based algorithm with following properties:
  [m is number of query points, n is number of plans in diagram]

  - Time complexity is O(mn)
    - linear in number of plans for a given diagram resolution
  - Approximation Factor is O(ln m)
    - bound is both tight and optimal
    - in practice, performance closely approximates offline optimal

Reduced Plan Diagram [λ=10%]

Complex Plan Diagram

[QT8, OptA*, Res=100]



Applications of Plan Diagram Reduction

- Quantifies redundancy in plan search space
- Provides better candidates for plan-cacheing
- Enhances viability of Parametric Query Optimization (PQO) techniques
- Improves efficiency/quality of LEC plans
- Minimizes overheads of multi-plan approaches (e.g. Adaptive Query Processing)

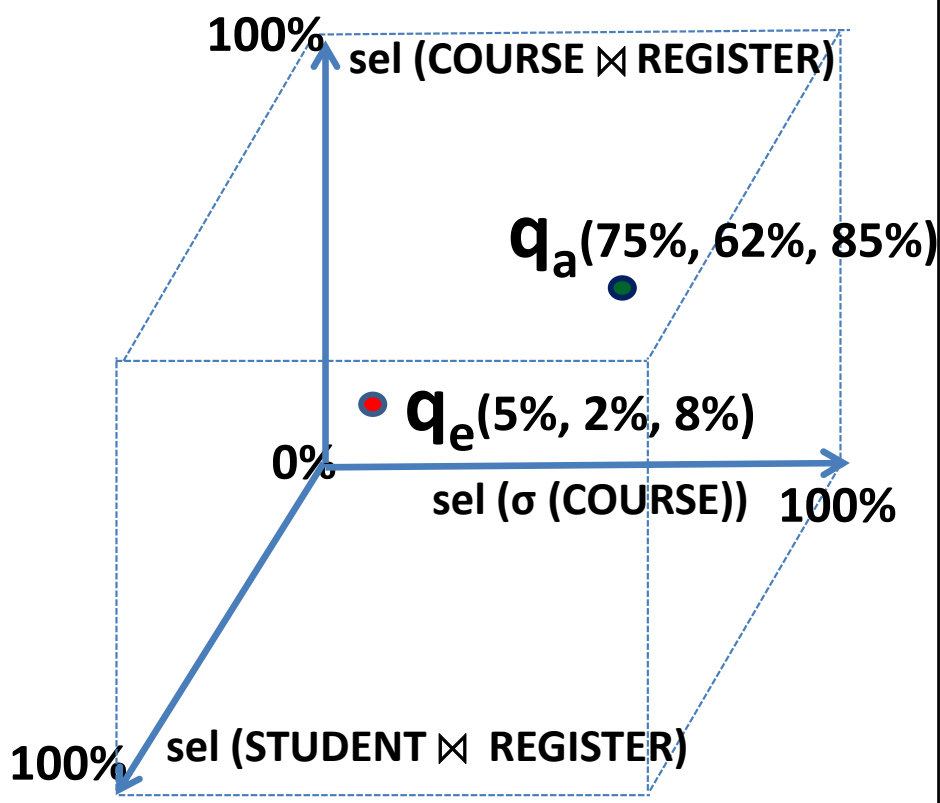
- Identifies selectivity-error resistant plan choices
  - retained plans are robust choices over larger selectivity parameter space

Limitation

Cost Greedy can cause arbitrarily poor performance if the selectivity error is large enough that the actual location of the query falls outside the swallowing region of the estimated location.

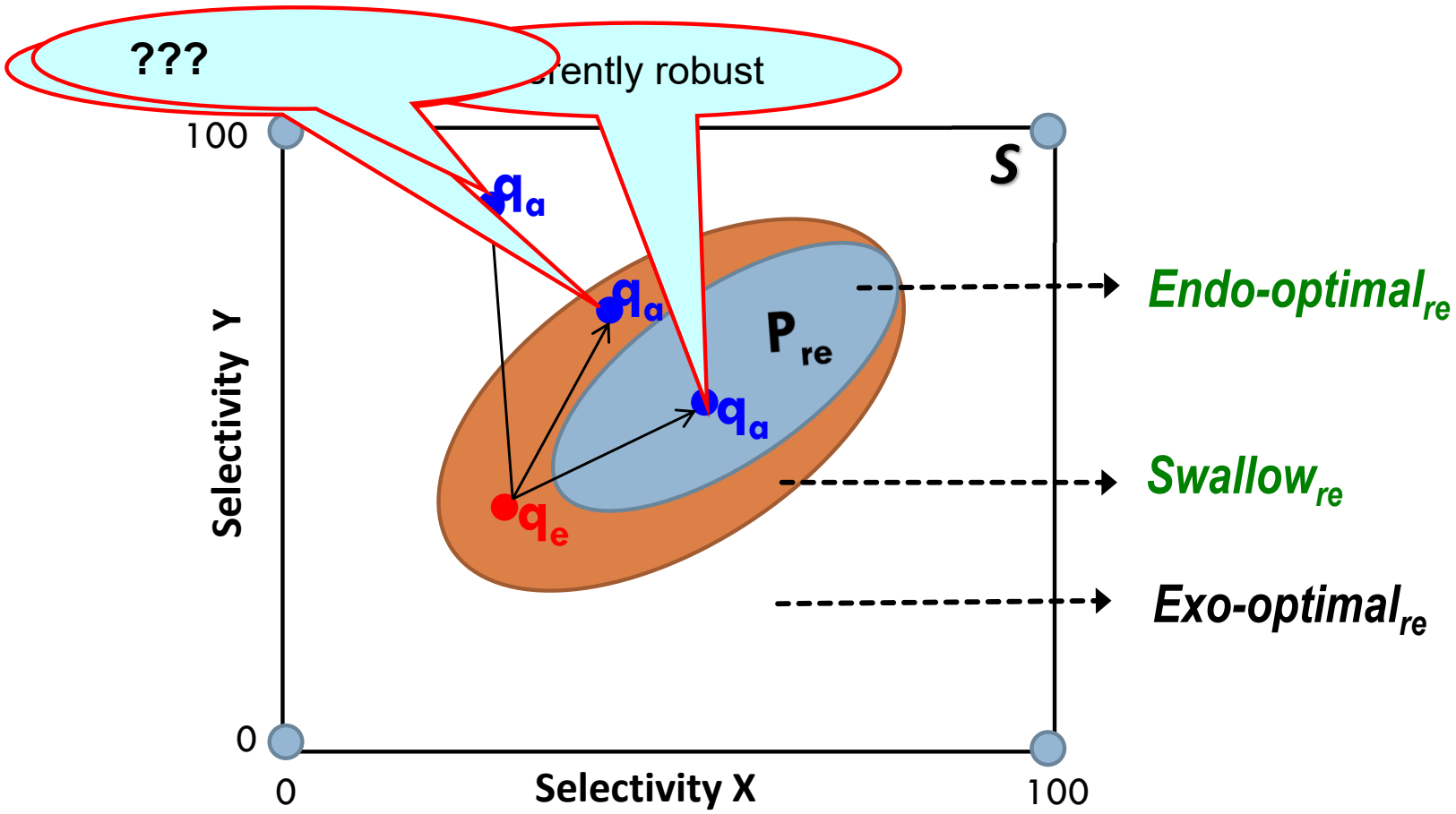
Notation

```
select  *
from    STUDENT, COURSE, REGISTER
where   S.RollNo = R.RollNo and
        C.CourseNo = R.CourseNo and
        C.fees < 1000
```



- q_e – estimated selectivity location in SS (Selectivity Space)
- q_a – actual run-time location in SS
- P_{oe} – optimal plan for q_e
- P_{oa} – optimal plan for q_a
- P_{re} – replacement plan for P_{oe}

Error Locations wrt Plan Replacement Regions



SEER [19]

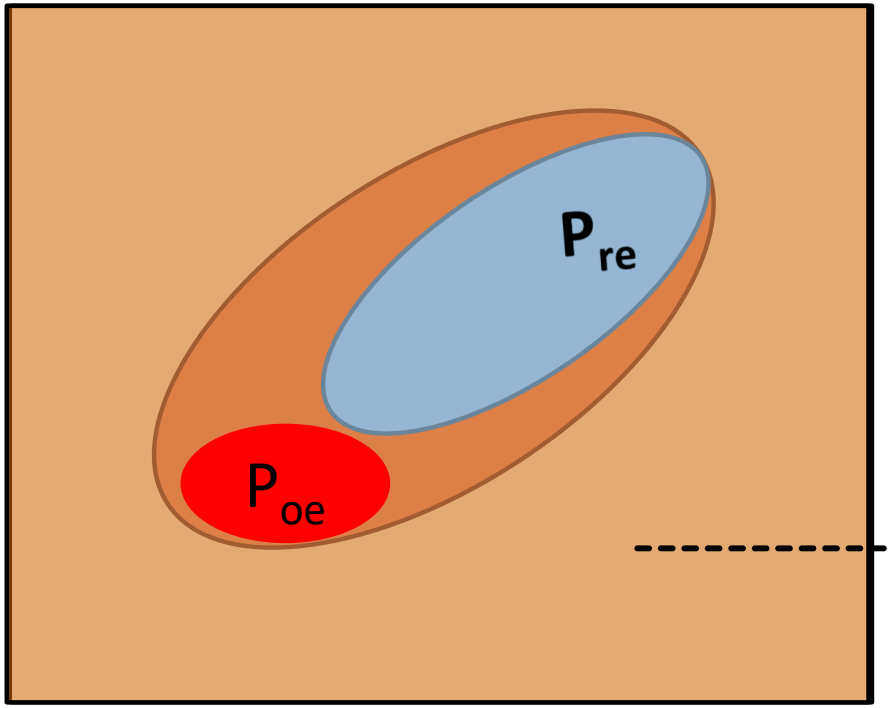
[Selectivity Estimate Error Resistance]

Globally Safe Replacement

- Earlier local constraint:
  P_{re} can replace P_{oe} if
  - $\forall$ points q in P_{oe}'s endo-optimality region,
    $cost(P_{re}, q) \leq (1+\lambda)\, cost(P_{oe}, q)$

- New global constraint:
  P_{re} can replace P_{oe} only if it guarantees a globally safe space
  - $\forall$ points q in selectivity space **S**,
    $cost(P_{re}, q) \leq (1+\lambda)\, cost(P_{oe}, q)$

Globally Safe Replacement



Safe (P_{re}, P_{oe})

Plan Cost Model (2D)

Given selectivity variations x and y, for any plan P in the plan dia[illegible] current optimizers, we can fit:

Index Scan; Aggregate

Join

$$PlanCost_P(x,y) = a_1 x + a_2 y + a_3 xy + a_4 x \log x + a_5 y \log y + a_6 xy \log xy + a_7$$

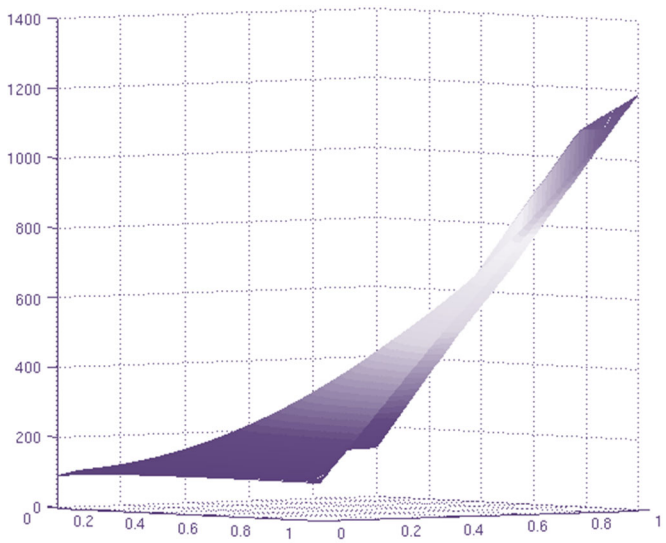
Sort; Group

TableScan

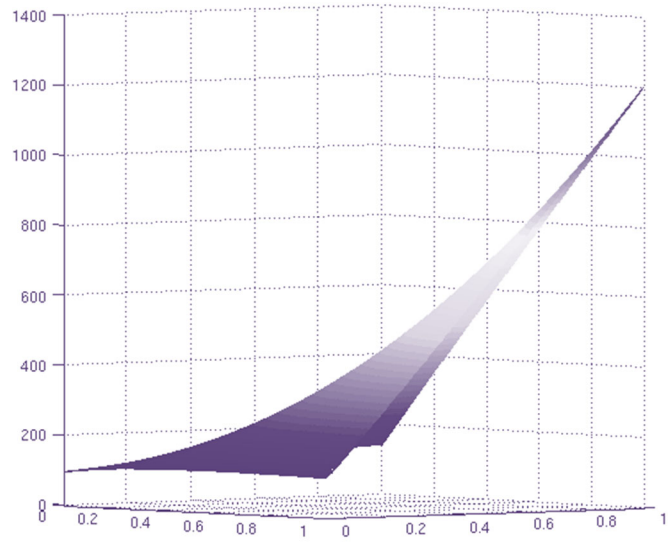
The specific values of a_1 through a_7 are a function of P .

Extension to n-dimensions is straightforward.

Cost Model Fit Example



Original Cost Function



Fitted Cost Function

$$Cost(x,y) = 17.9x + 45.9y + 1046xy - 39.5x\log x + 4.5y\log y + 27.6xy\log xy + 97.3$$

Main Result

Given the 7-coefficient plan cost model, need to perform APC at only the perimeter of the selectivity space to determine global safety

i.e. Border Safety ⇒ Interior Safety !

Limitation

- Although SEER introduces stability into the plan choices, its performance guarantees are with respect to $\mathbf{P_{oe}}$, the optimal plan at the estimated location (i.e. the native optimizer's plan)

- Ideally, we would like performance guarantees to be with respect to $\mathbf{P_{oa}}$, the optimal plan at the actual location (i.e. the "God's plan").

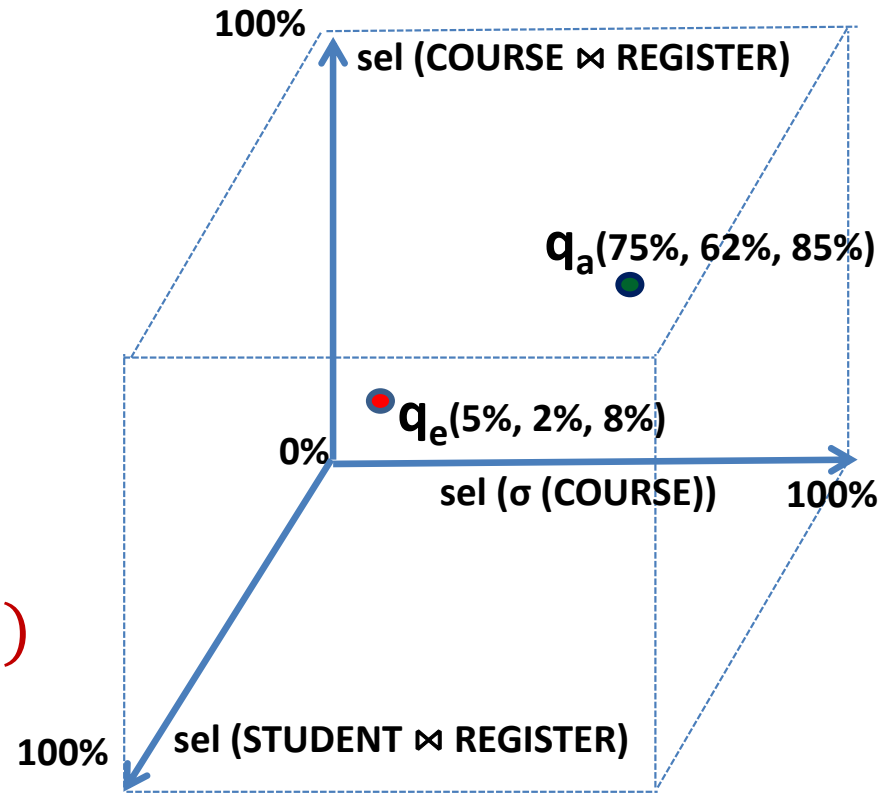
Stage 3: Robust Execution

Performance Metrics

- q_e – estimated selectivity location in SS
- q_a – actual run-time location in SS

- P_{oe} – optimal plan for q_e
- P_{oa} – optimal plan for q_a

$$\boldsymbol{SubOpt(q_e, q_a) = \frac{cost(P_{oe}, q_a)}{cost(P_{oa}, q_a)} \quad [1, \infty)}$$



$$\boldsymbol{MaxSubOpt\ (MSO) = MAX[SubOpt(q_e, q_a)] \quad \forall q_e, q_a \in SS}$$

Note: Metric is now with respect to the ideal plan

APPROACHES

- Bounded Impact (PVLDB 2009 [36])
  - performance guarantee with quartic dependency on error magnitude
- Plan Bouquet (SIGMOD 14 / TODS 16 [14])
  - discovery-based approach to selectivities
  - error-independent guarantees with linear dependency on plan diagram density
- Spill-Bound (ICDE 16 / TKDE 19 [25])
  - platform-independent guarantee with quadratic dependency on error dimensionality
- Frugal Spill-Bound (PVLDB 2018 [26])
  - extension to ad-hoc queries

Measuring Cardinality Estimation Errors

Popular error metrics (= optimization goals)

$$l_2 = \sqrt{(f_e(x) - f_a(x))^2}$$

$$l_\infty = \max |(f_e(x) - f_a(x)|$$

Minimizing these error metrics can lead to **arbitrarily bad plans**!

Q(uotient) Error

- Errors propagate multiplicatively, so metric should also be multiplicative
- It should be symmetric wrt over- and under-estimation

q-error is defined as:

$$l_q = \max_{(x)} \frac{max\ (f_e(x), f_a\ (x))}{min\ \ (f_e(x), f_a\ (x))}$$

- actual cardinality 10, estimation 100 $\Rightarrow$ l_q = 10
- actual cardinality 10, estimation 1 $\Rightarrow$ l_q = 10

Knowing *q*-error provides **bounds on resulting plan performance**!

Cost Bounds Implied by Q-error

- **Theorem:**
  Let all joins be Sort-Merge or all be Grace-Hash. Then

$$MSO \leq q^4$$

where q is the maximum q-error taken over all intermediate results.

Problems: q can be arbitrarily large
q is usually not known in advance

Plan Bouquet [14]

Approach

- Plan Bouquet is a new query processing technique, that completely abandons estimating operator selectivities
- Instead, run-time selectivity discovery using compile-time selected bouquet of plans
  - provides worst case performance guarantees wrt ideal that magically knows the correct selectivities
    e.g. for single error-prone selectivity, relative guarantee of 4
  - empirical performance well within guaranteed bounds on industrial-strength environments

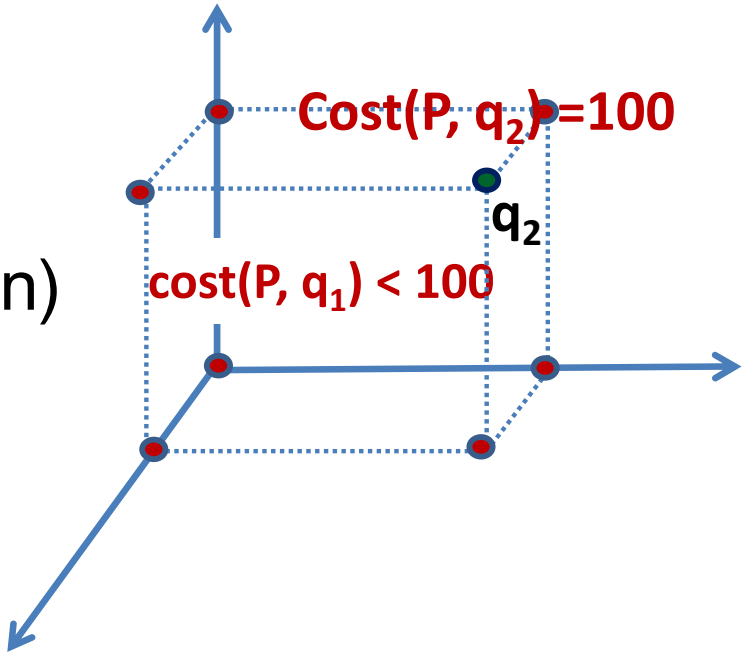
Basic Assumption

- Plan Cost Monotonicity (PCM)

For any plan P and distinct locations q_1 and q_2

$$\text{Cost}(P, q_1) < \text{Cost}(P, q_2) \quad \text{if } q_1 \prec q_2$$

(i.e. spatial domination $\Rightarrow$ cost domination)

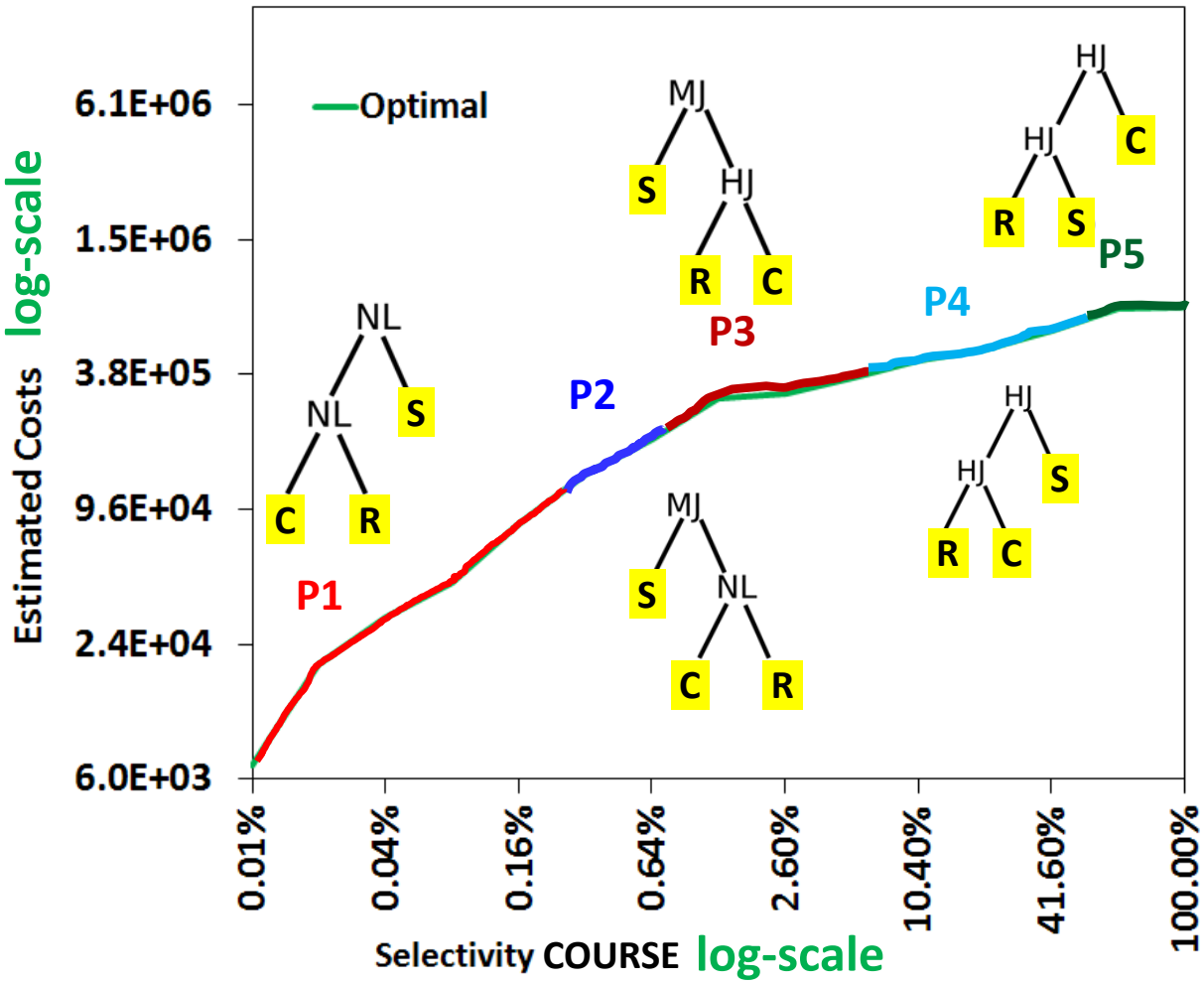


Contemporary Optimizer Behavior on 1D Selectivity Space

Parametric Optimal Set of Plans (POSP)

(Parametric version of Example Query)

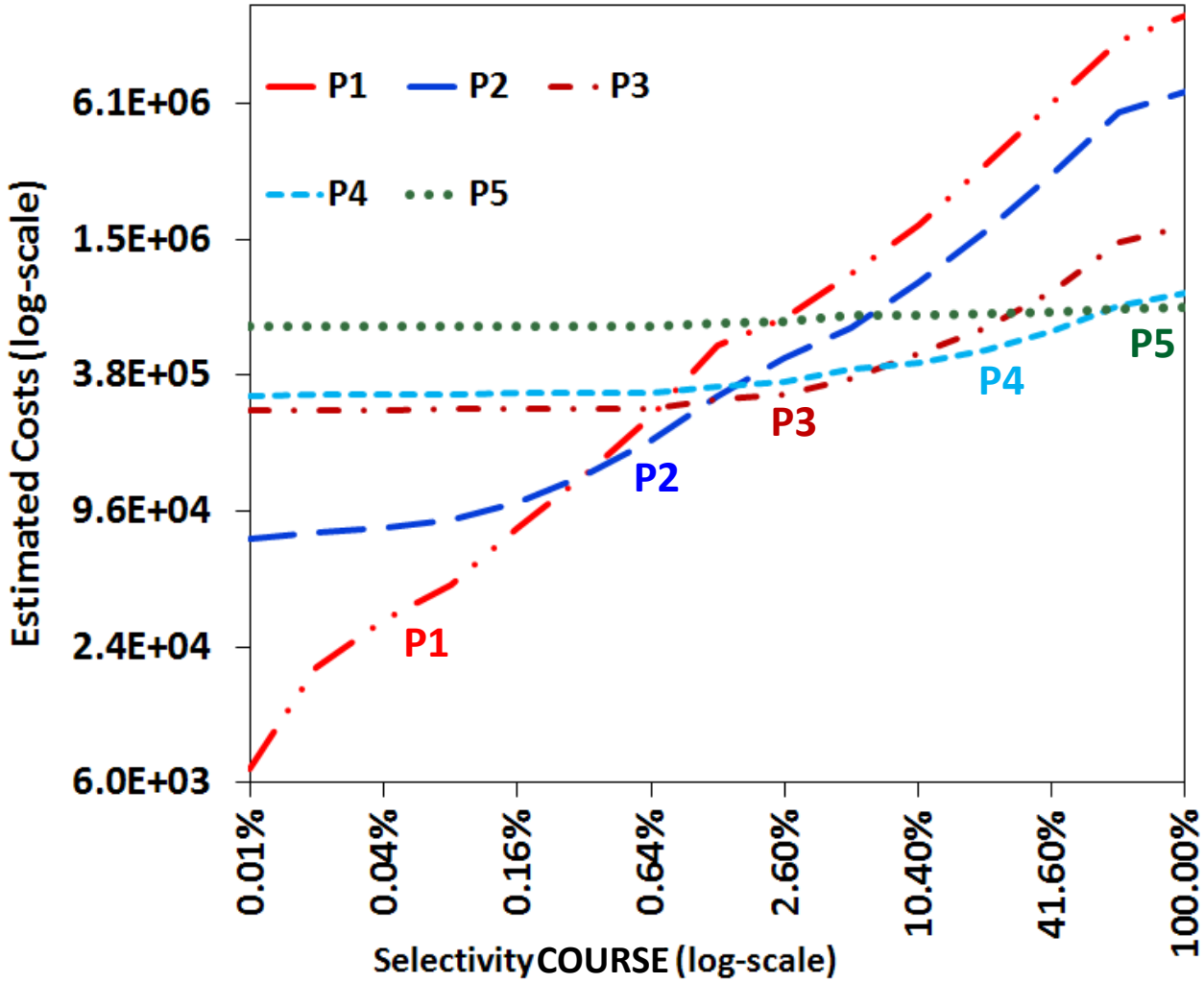
```
select  *
from    STUDENT, COURSE, REGISTER
where   S.RollNo = R.RollNo and
        C.CourseNo = R.CourseNo and
        C.fees < $1
```



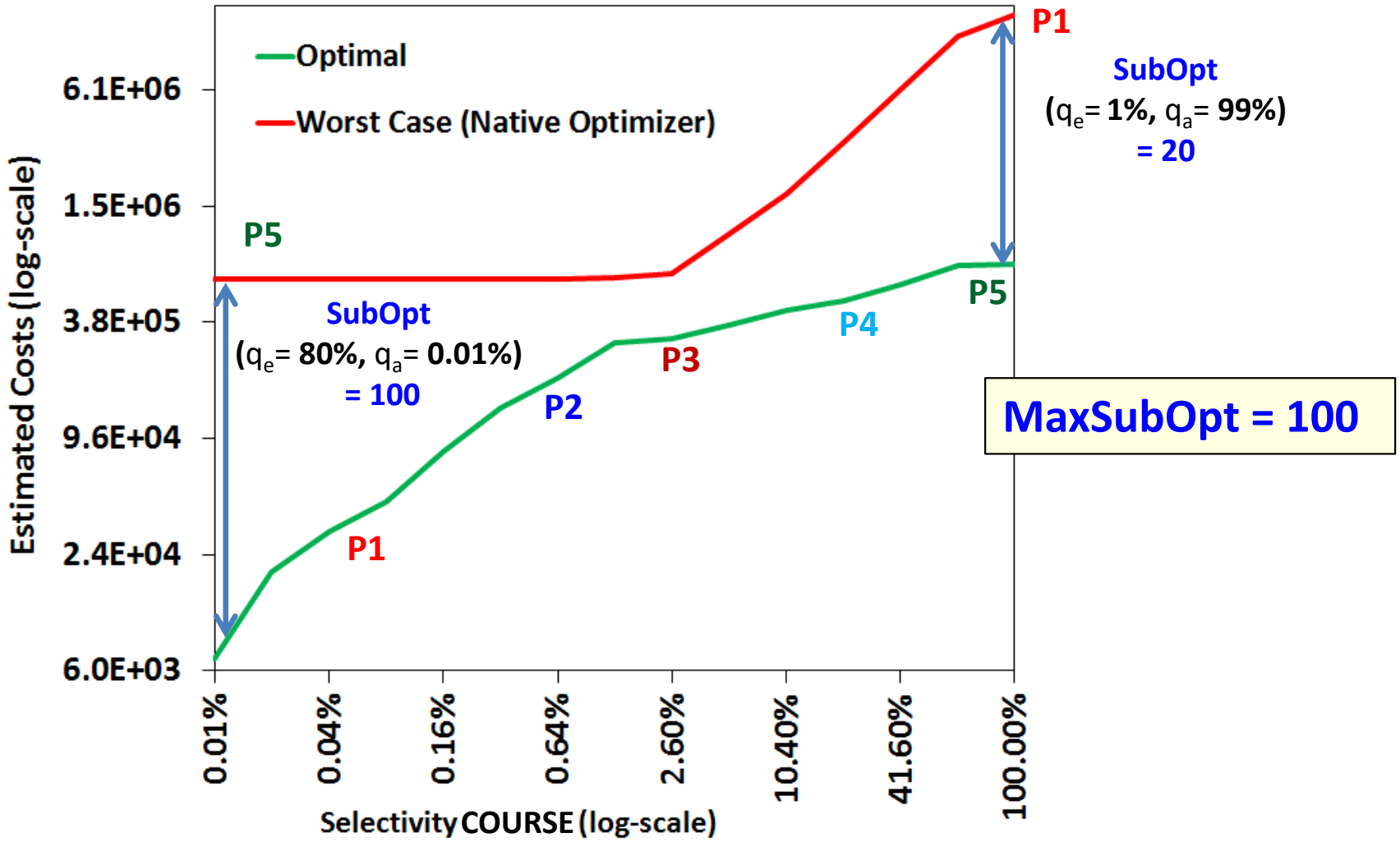
S: Student
C: Course
R: Register

NL: Nested Loop Join
MJ: Merge Join
HJ: Hash Join

POSP Performance Profile (across SS)

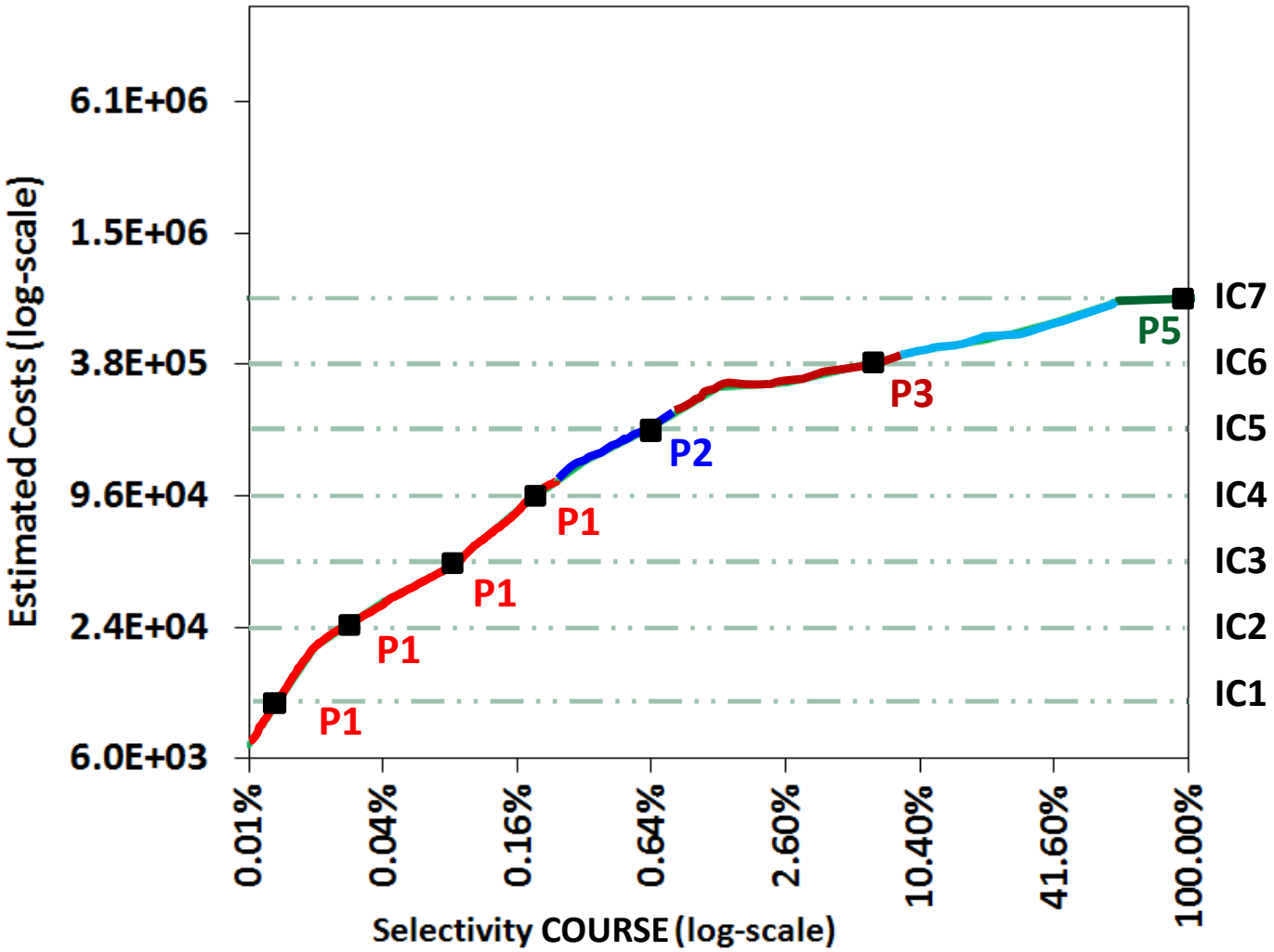


Sub-optimality Profile (across SS)



Plan Bouquet Behavior on 1D Selectivity Space

Bouquet Identification

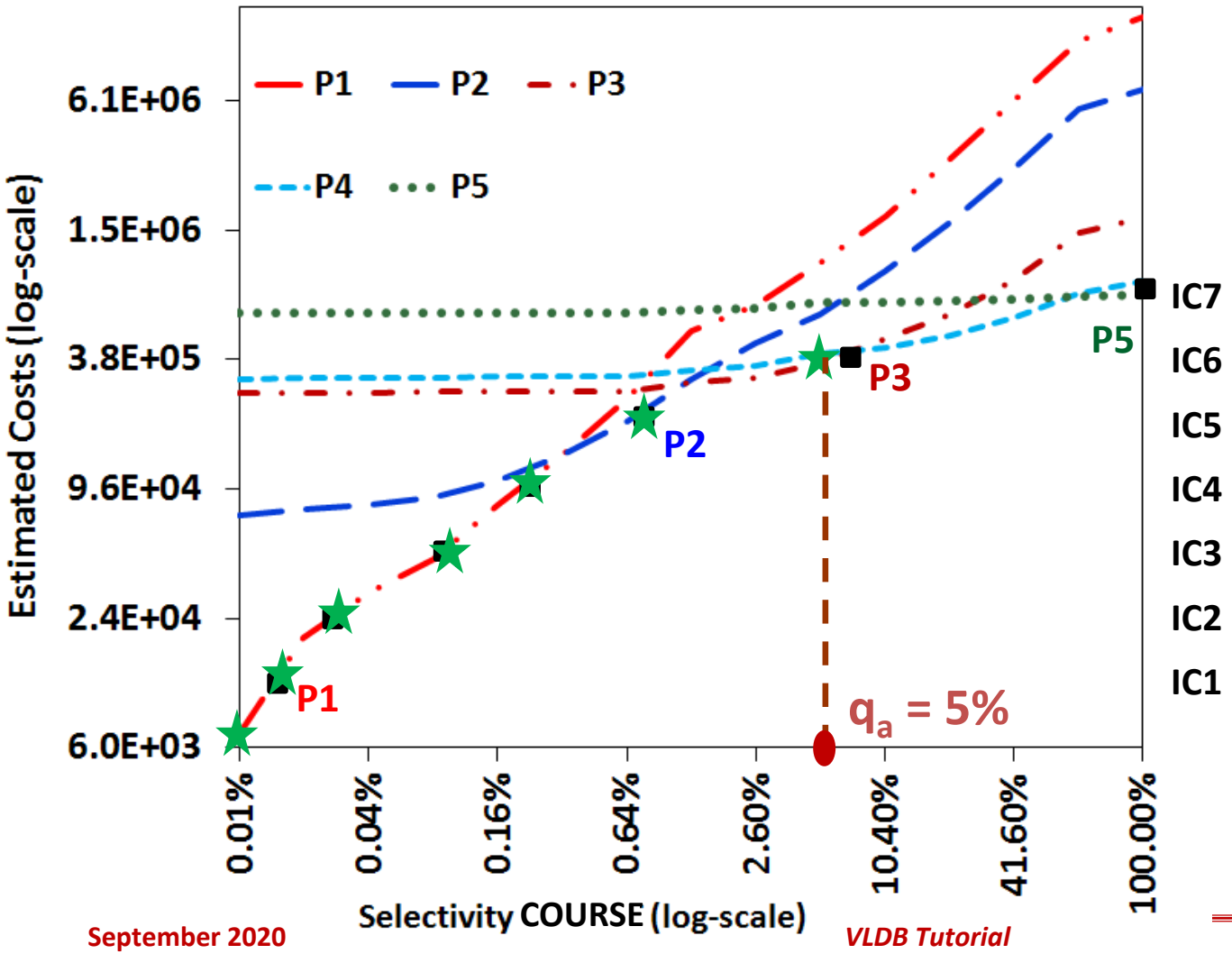


Step 1: Draw cost steps with cost-ratio **r=2** (geometric progression).

Step 2: Find plans at intersection of optimal profile with cost steps

Bouquet = {P1, P2, P3, P5}

Bouquet Execution



Let q_a = 5%

(1) Execute P1
with budget IC1(1.2E4)
Throw away results of P1

(2) Execute P1
with budget IC2(2.4E4)
Throw away results of P1

(3) Execute P1
with budget IC3(4.8E4)
Throw away results of P1

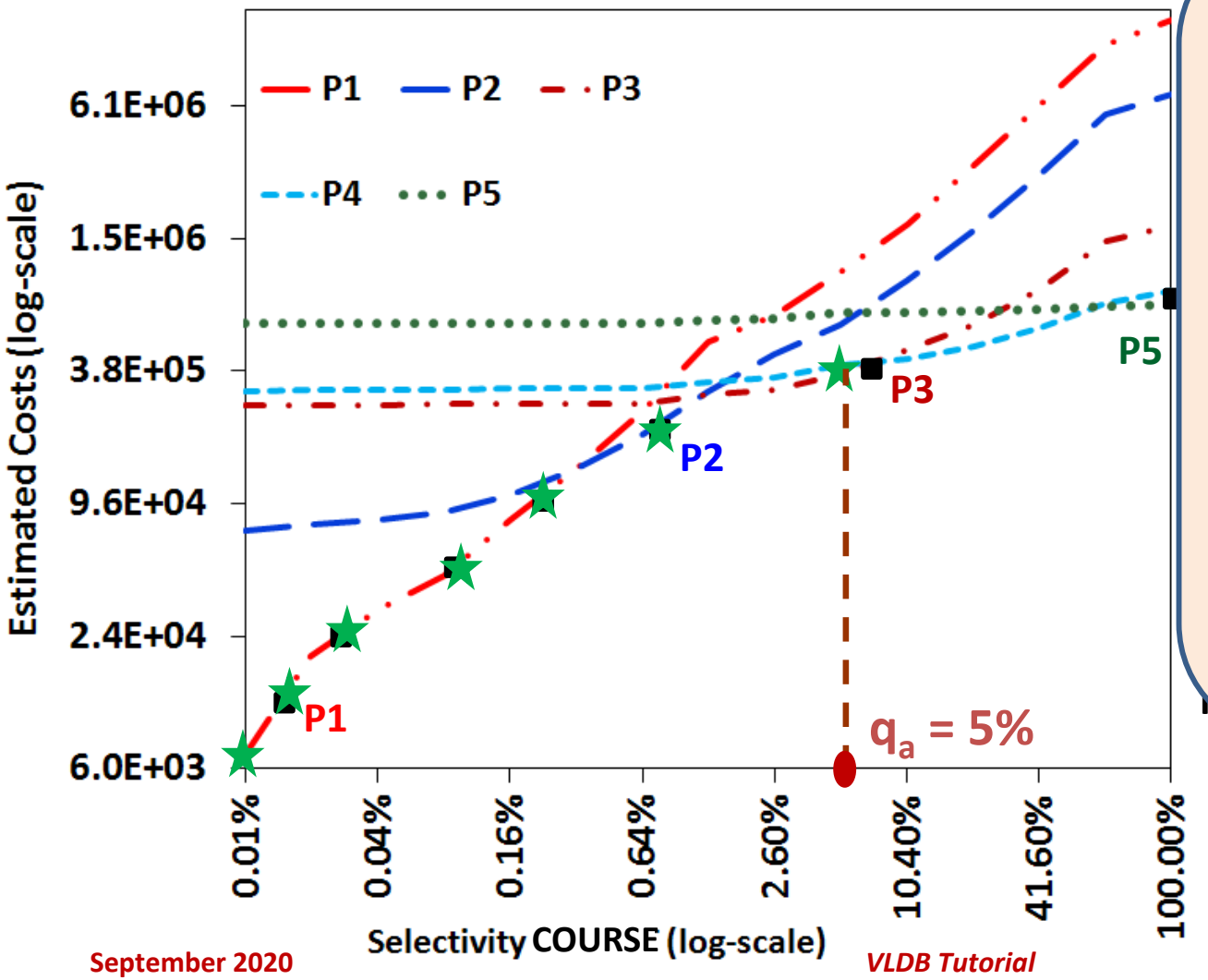
(4) Execute P1
with budget IC2(9.6E4)
Throw away results of P1

(5) Execute P2
with budget IC5(1.9E5)
Throw away results of P2

(6) Execute P3
with budget IC6(3.8E5)

P3 completes with cost 3.4E5

Bouquet Execution



Let q_a = 5%

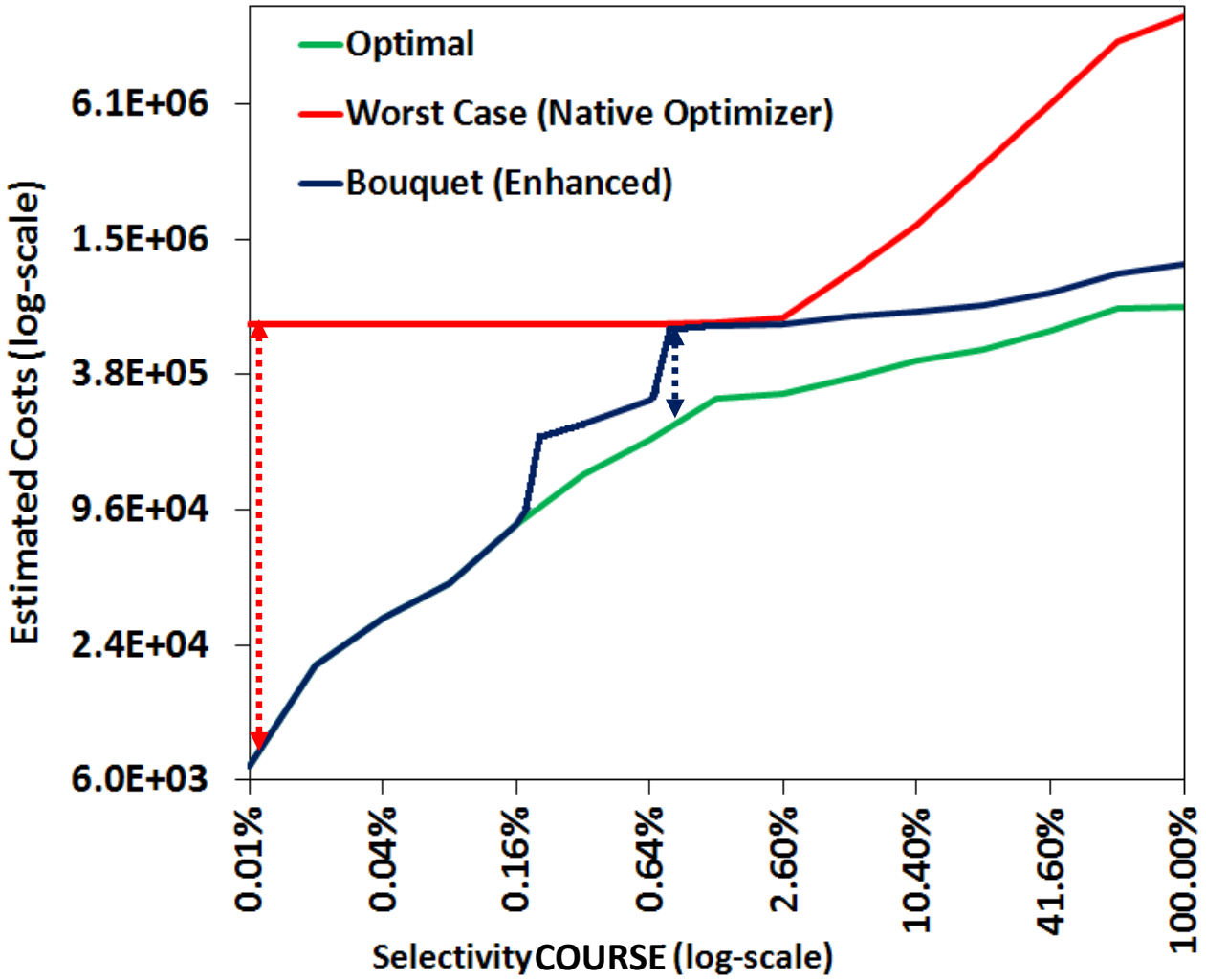
Bouquet Cost = 3.4 E5 (P3) +
1.92 E5 (P2) +
0.96 E5 (P1) +
0.48 E5 (P1) +
0.24 E5 (P1) +
0.12 E5 (P1)
= 7.1 E5

SubOpt (*, 5%) = 7.1/3.4 = 2.1

With obvious optimization
SubOpt(*, 5%) = 6.3/3.4 = 1.8

with budget IC6(3.8E5)
P3 completes with cost 3.4E5

Bouquet Performance (EQ)



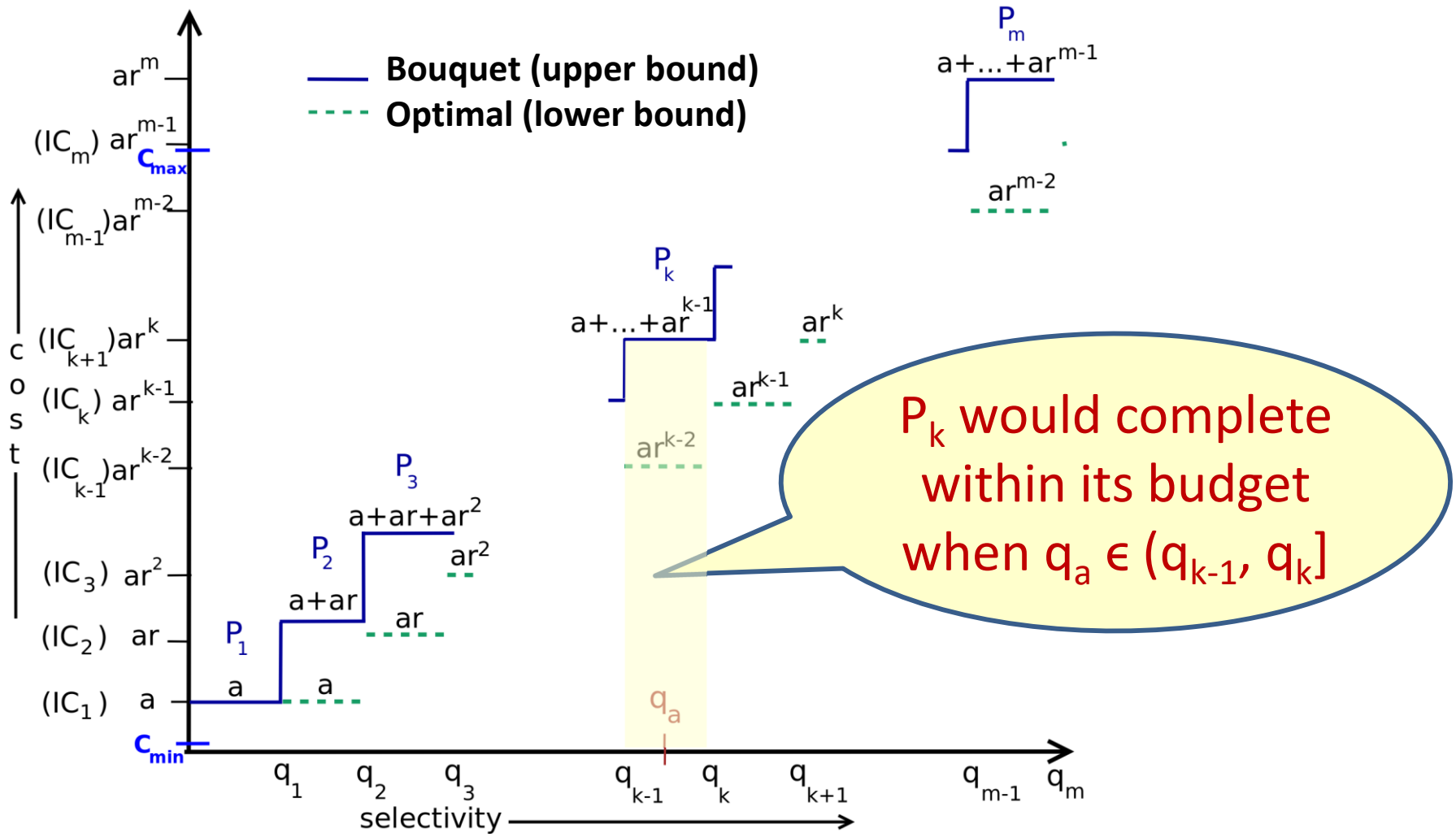
Native Optimizer

MaxSubOpt = 100

Bouquet

MaxSubOpt = 3.1

Worst Case Cost Analysis



1D Performance Bound

$$C_{bouquet}(q_{k-1}, q_k] = cost(IC_1) + cost(IC_2) + \ldots + cost(IC_{k-1}) + cost(IC_k)$$
$$= a + ar + \ldots + ar^{k-2} + ar^{k-1}$$
$$= \frac{a(r^k - 1)}{(r-1)}$$

$$C_{optimal}(q_{k-1}, q_k] \geq ar^{k-2}$$ (Implication of PCM)

$$SubOpt_{bouquet}(*, q_a) \leq \frac{1}{ar^{k-2}} \times \frac{a(r^k - 1)}{(r-1)} \leq \frac{r^2}{r-1} \quad \forall q_a \in (q_{k-1}, q_k]$$

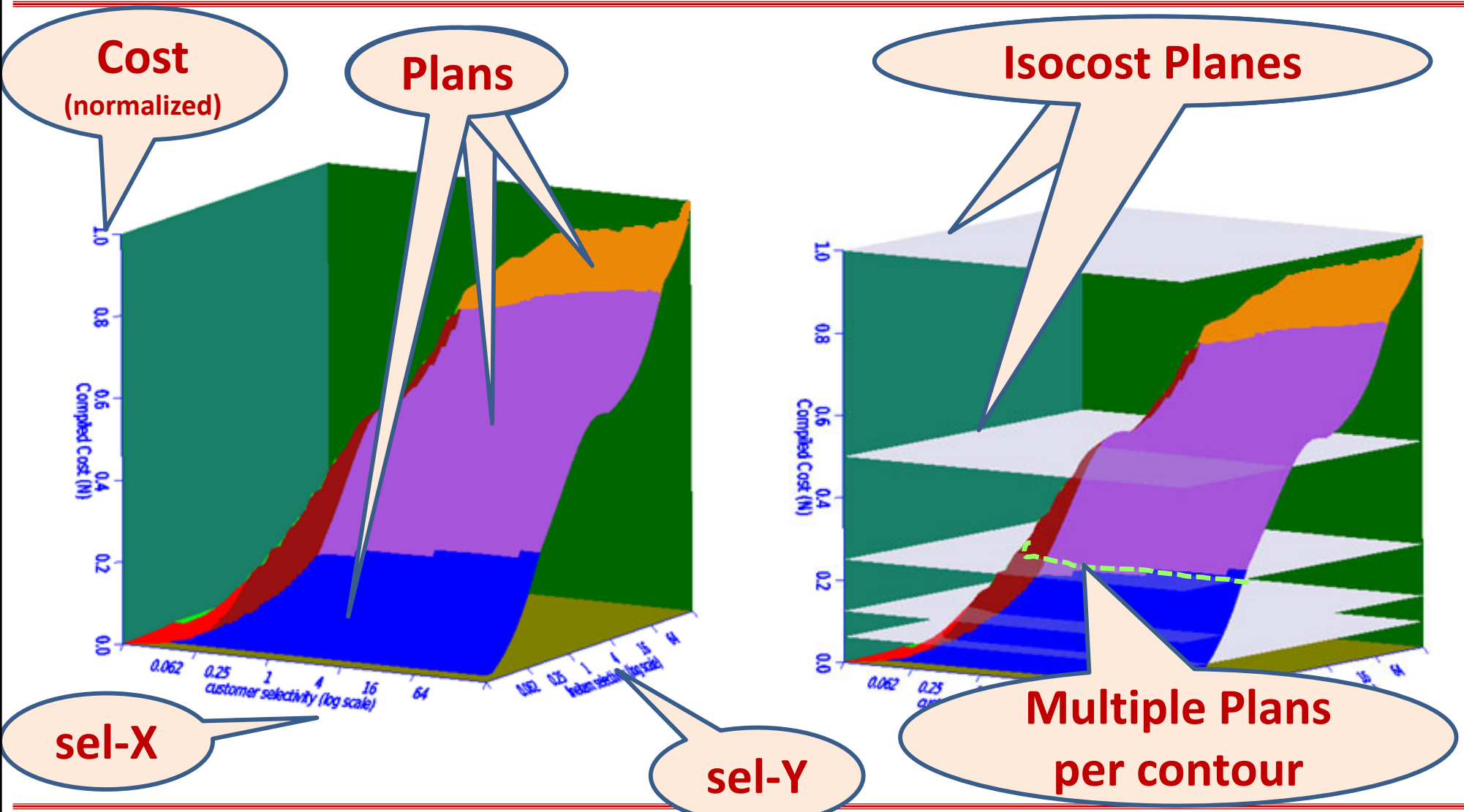
Reaches minima at r = 2

➔ MSO = 4

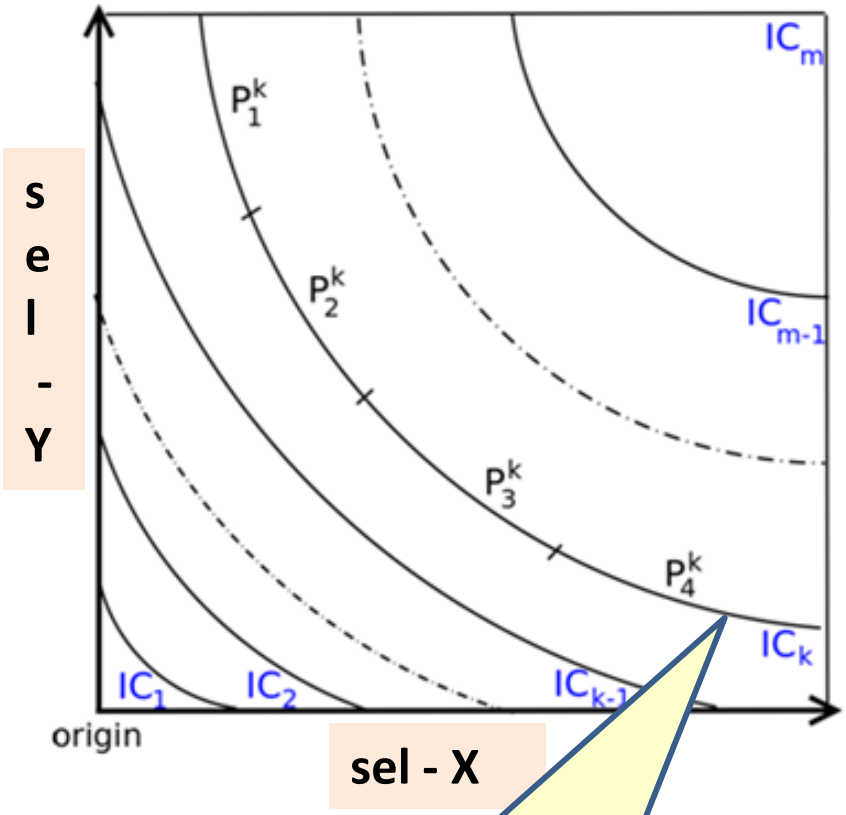
Best performance achievable by <u>any</u> deterministic online algorithm!

Bouquet Approach in 2D SS

2D Bouquet Identification



Characteristics of 2D Contours

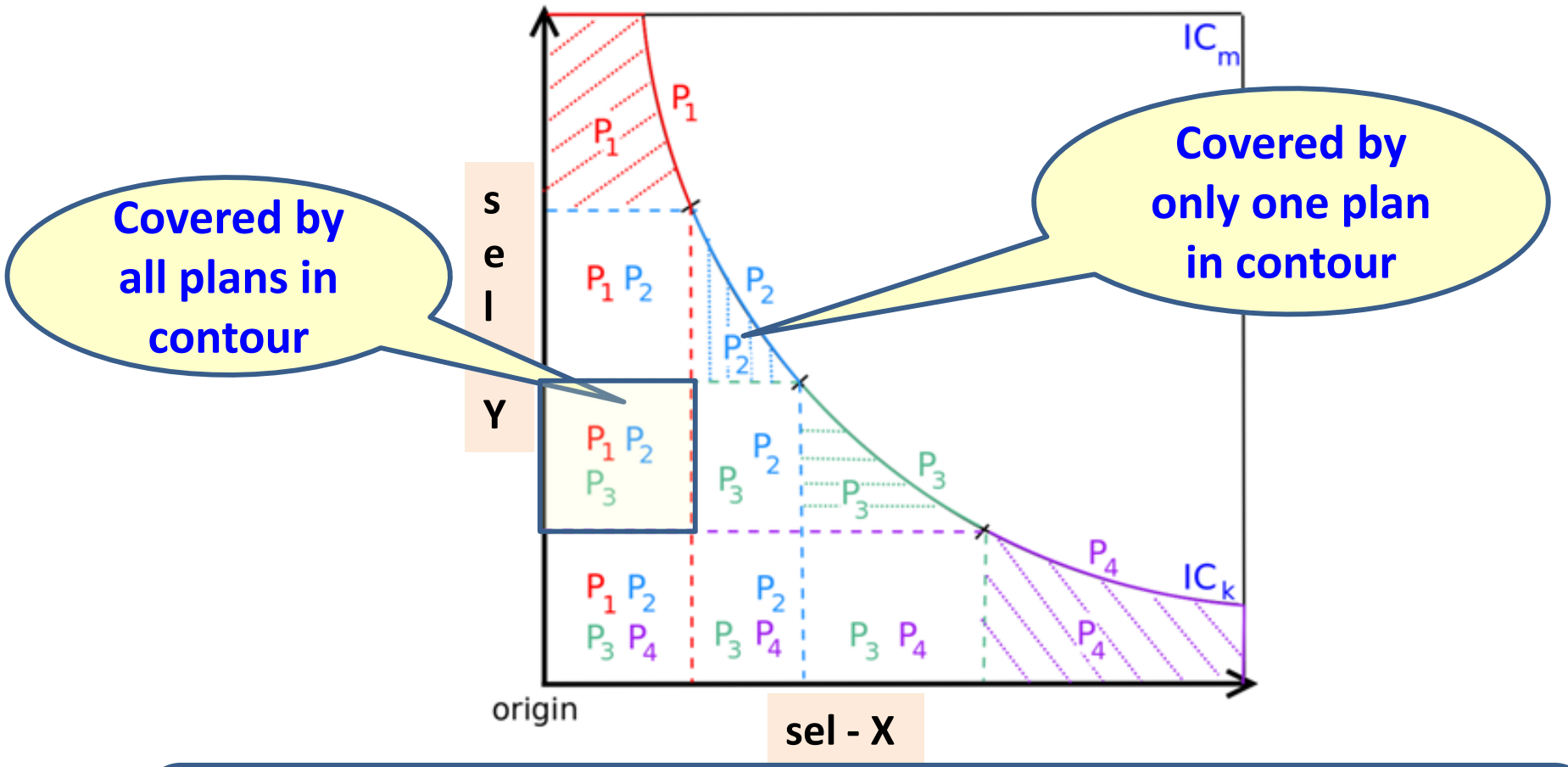


2D contours
- **Hyperbolic curves**
- **Multiple plans per contour**

Third quadrant coverage (due to PCM)

P_2^k can complete any query with actual selectivitiy (q_a) in the shaded region within cost(IC_k)

Crossing 2D Contours



⇒ **Entire set of contour plans must be executed to fully cover all locations under IC_k**

2D Performance Analysis

- When $q_a \in (IC_{k-1}, IC_k]$

Number of plans on i^{th} contour

$$C_{bouquet}(q_a) = \sum_{i=1}^{k} [n_i \times cost(IC_i)]$$

$\rho = \max(n_i)$

$$C_{bouquet}(q_a) \leq \rho \times \sum_{i=1}^{k} cost(IC_i)$$

$$SubOpt_{bouquet}(q_a) = 4\rho \quad \text{(Using 1D Analysis)}$$

- MSO = 4ρ

Bound for N-dimensions: $MSO = 4 \times \rho_{ICsurface}$

Dealing with large ρ

- In practice, ρ can often be large, even in 100s, making the performance guarantee of 4ρ impractically weak

- Reducing ρ:

  Anorexic POSP reduction
  (from CostGreedy)

MSO guarantees (compile time)

	Query (dim)	MSO Bound
TPC-H	Q5 (3D)	14.4
	Q7 (3D)	14.4
	Q8 (4D)	33.6
	Q7 (5D)	43.2
TPC-DS	Q15 (3D)	14.4
	Q96 (3D)	14.4
	Q7 (4D)	19.2
	Q19 (5D)	38.4
	Q26 (4D)	24.0
	Q91 (4D)	43.2

Empirical Evaluation

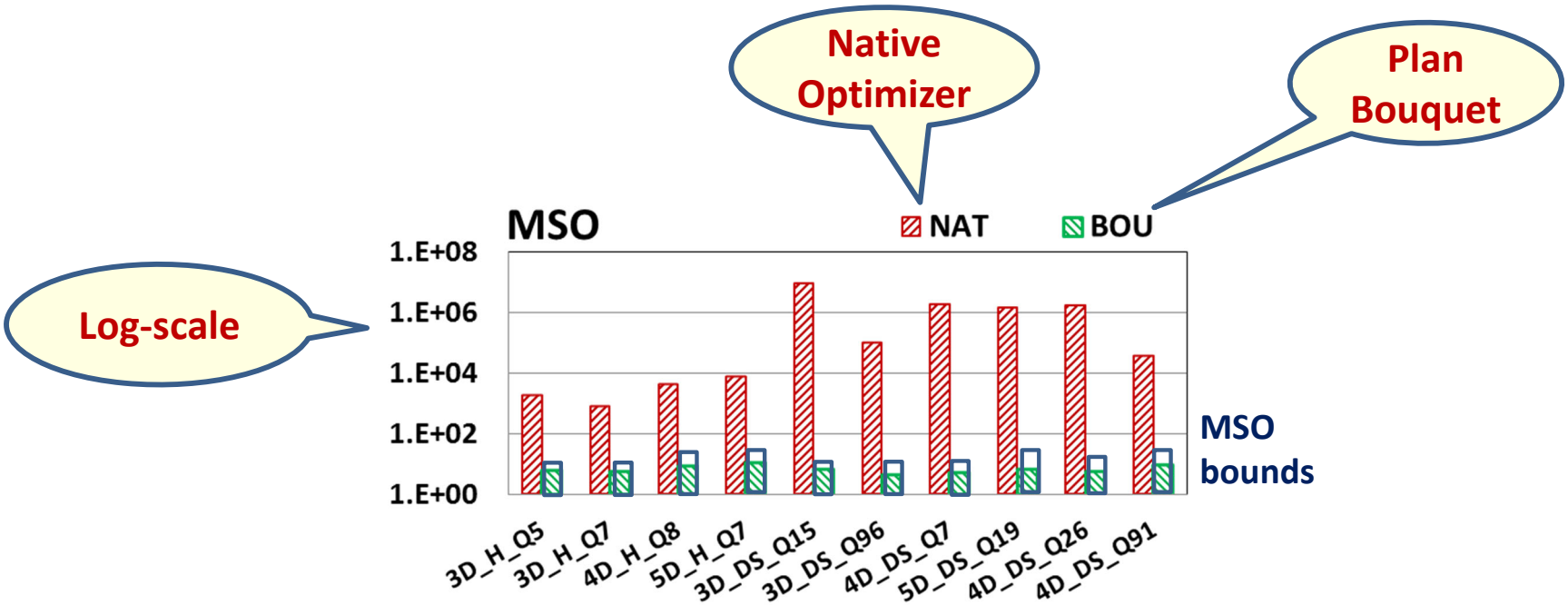
Experimental Testbed

- Database Systems: PostgreSQL and COM (commercial engine)
- Databases: TPC-H and TPC-DS (standard benchmarks)
- Physical Schema: Indexes on all attributes present in query predicates

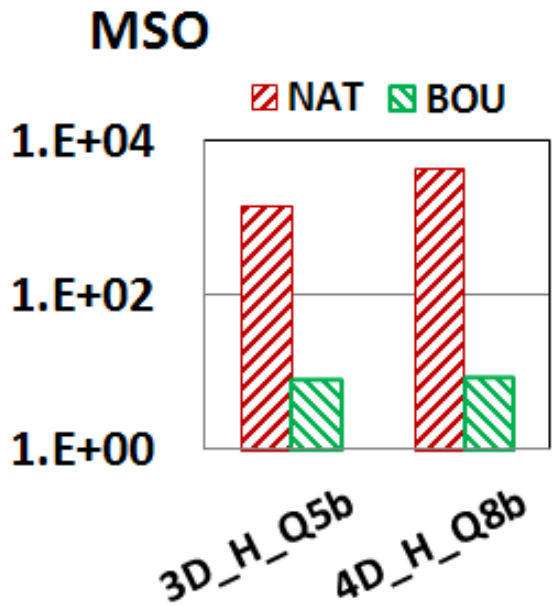
- Workload: 10 complex queries from TPC-H and TPC-DS
  - with SS having upto **5 error dimensions** (join-selectivities)

- Metrics: Computed MSO using Abstract Plan Costing over SS

Performance on PostgreSQL



Performance with Commercial System



Summary

- Plan bouquet approach achieves
  - bounded performance sub-optimality
    - using a (cost-limited) plan execution sequence guided by isocost contours defined over the optimal performance curve
  - robust to changes in data distribution
    - only q_a changes – bouquet remains same
  - easy to deploy
    - bouquet layer on top of the database engine
  - repeatability in execution strategy (important for industry)
    - q_e is always zero, depends only on q_a
    - independent of metadata contents

Limitations of PlanBouquet

- Enormous offline computational effort to produce the plan diagram, suitable only for "canned" queries
  - Partially addressed by enumerating only the contours, not the entire diagram
- Practical guarantee values are predicated on anorexic reduction holding true
- Guarantee of 4ρ depends on plan diagram complexity, making it not portable across query optimizers, databases and hardware systems

FOLLOW-UP WORK

- Spill-Bound (ICDE 16 / TKDE 19 [25])
  - Half-space pruning instead of hypograph pruning
  - MSO = $D^2 + 3D$ (where D is dimensionality of SS)
  - platform-independent guarantee
  - Lower bound of $\Omega(D)$
- Frugal Spill-Bound (PVLDB 2018 [26])
  - extension to ad-hoc queries
  - exponential decrease in overheads for linear relaxation in MSO guarantee

Stage 4: Robust Cost Models

Approaches

- Learning-based approaches (ICDE 2009, ICDE 2012 [4], PVLDB 2019 [40])

- Statistical approaches (ICDE 2013 [45], PVLDB 2013 [44], PVLDB 2014 [47])

Optimizer's Cost Estimates: Unusable

Direct Scaling:
Predict the *execution time T*
by *scaling* the *cost estimate C*,
i.e., $T = a \cdot C$

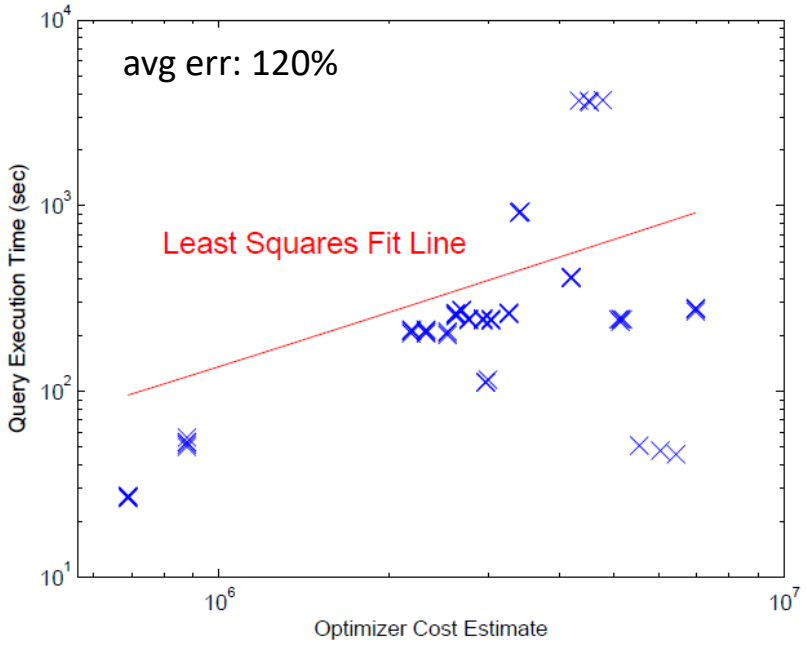


Fig. 5 of [4]

Why Does Direct Scaling Fail?

- PostgreSQL's cost model

$$C = n_s c_s + n_r c_r + n_t c_t + n_i c_i + n_o c_o$$

Scaling

$$T = a \cdot C = c_s' \cdot (n_s + n_r \frac{c_r}{c_s} + n_t \frac{c_t}{c_s} + n_i \frac{c_i}{c_s} + n_o \frac{c_o}{c_s})$$

$$c_s' = a \cdot c_s = a \cdot 1.0 = a$$

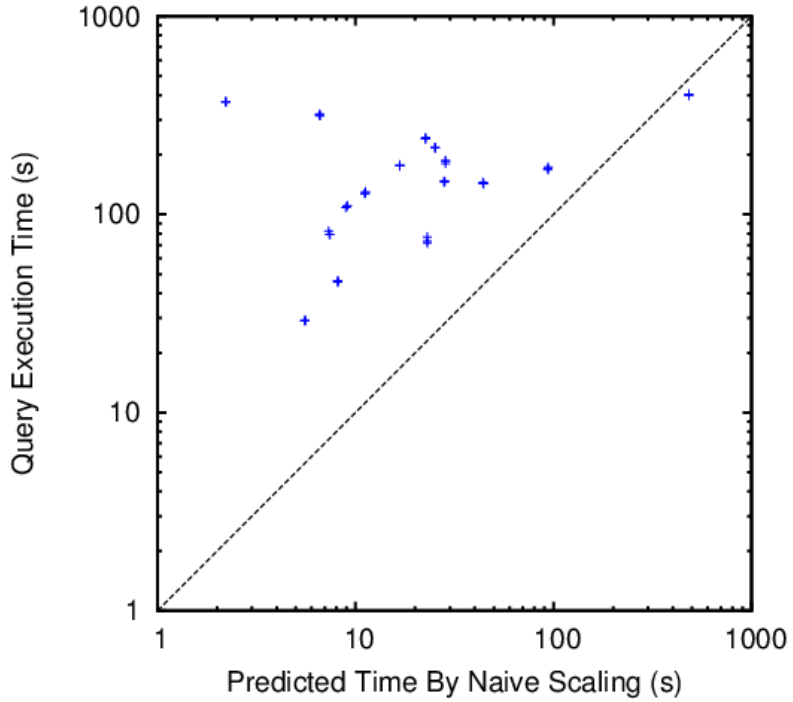
Should be correct!

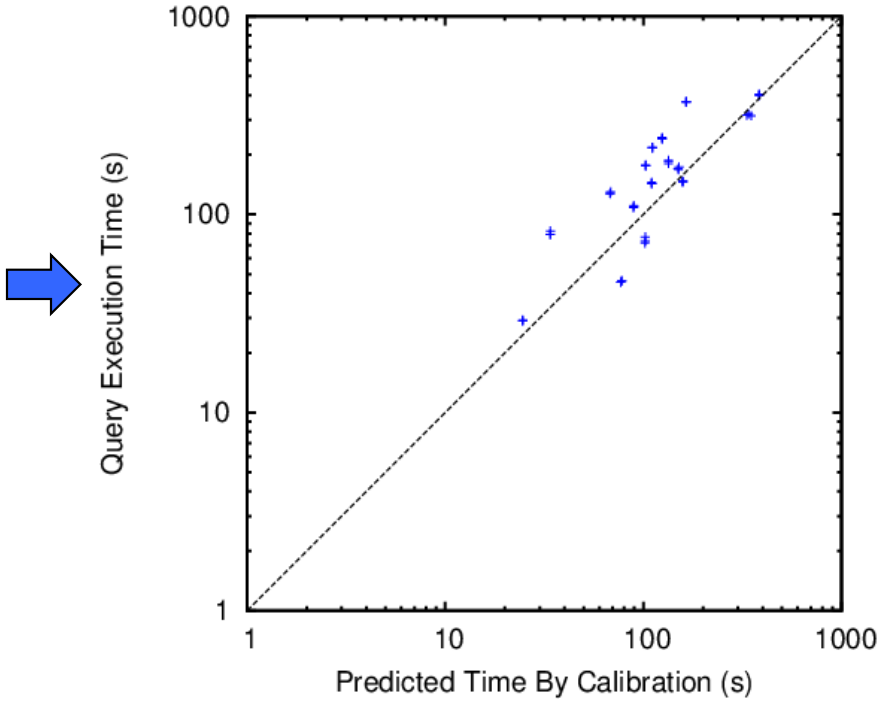
Cost Unit	Value
c_s: seq_page_cost	1.0
c_r: rand_page_cost	4.0
c_t: cpu_tuple_cost	0.01
c_i: cpu_index_tuple_cost	0.005
c_o: cpu_operator_cost	0.0025

- Assumptions for scaling fail in practice
  - *Ratios* between the *c* values are incorrect.
  - *n* values are incorrect.
- Solution: Proper calibration

Calibrated *c* and *n*

– Cost models become much more effective.





Prediction by Scaling:

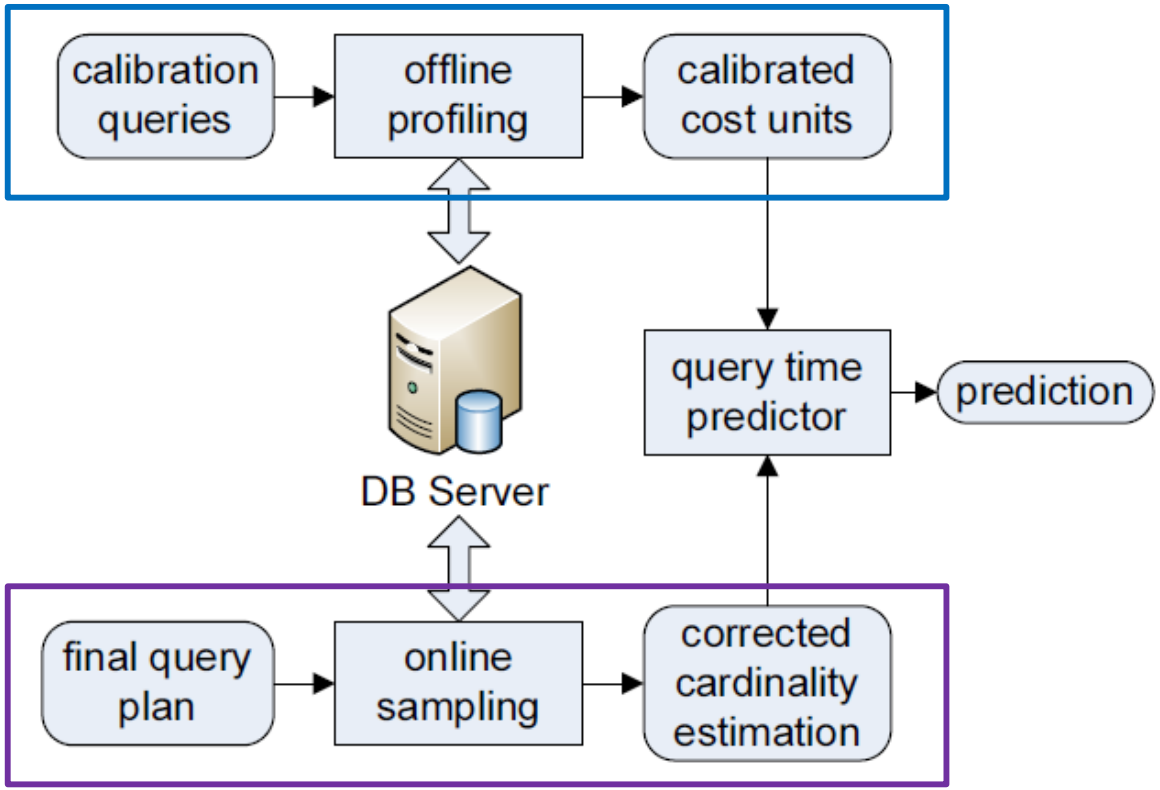
$$T_{pred} = a \cdot (\sum c \cdot n)$$

Prediction by Calibration:

$$T_{pred} = \sum c' \cdot n'$$

Main Idea

- Calibrate *c*: *use profiling queries*
- Calibrate *n*: *refine cardinality estimates*



Profiling Queries For PostgreSQL

Isolate the unknowns and solve them *one per equation*

q₁: select * from R	R in memory	$t_1 = c_t \cdot n_{t1}$
q₂: select count(*) from R	R in memory	$t_2 = c_t \cdot n_{t2} + c_o \cdot n_{o2}$
q₃: select * from R where R.A < a (R.A with an index)	R in memory	$t_3 = c_t \cdot n_{t3} + c_i \cdot n_{i3} + c_o \cdot n_{o3}$
q₄: select * from R	R on disk	$t_4 = c_s \cdot n_{s4} + c_t \cdot n_{t4}$
q₅: select * from R where R.B < b (R.B *unclustered* index)	R on disk	$t_5 = c_s \cdot n_{s5} + c_r \cdot n_{r5} + c_t \cdot n_{t5} + c_i \cdot n_{i5} + c_o \cdot n_{o5}$

Calibrating the *n values*

- The *n* values are *functions* of *N* values (i.e., input cardinalities).
  - Calibrating the *n values* ⇒ Calibrating the *N* values

Example 1 (In-Memory Sort)

$$sc = \lceil 2 \cdot N_t \cdot \log N_t \rceil \cdot c_o + tc\ of\ child$$

n_o

$$rc = c_t \cdot N_t$$

Example 2 (Nested-Loop Join)

$$sc = sc\ of\ outer\ child + sc\ of\ inner\ child$$

$$rc = c_t \cdot N_t^o \cdot N_t^i + N_t^o \cdot rc\ of\ inner\ child$$

n_t

sc: start-cost rc: run-cost $tc = sc + rc$: total-cost

N_t: # of input tuples

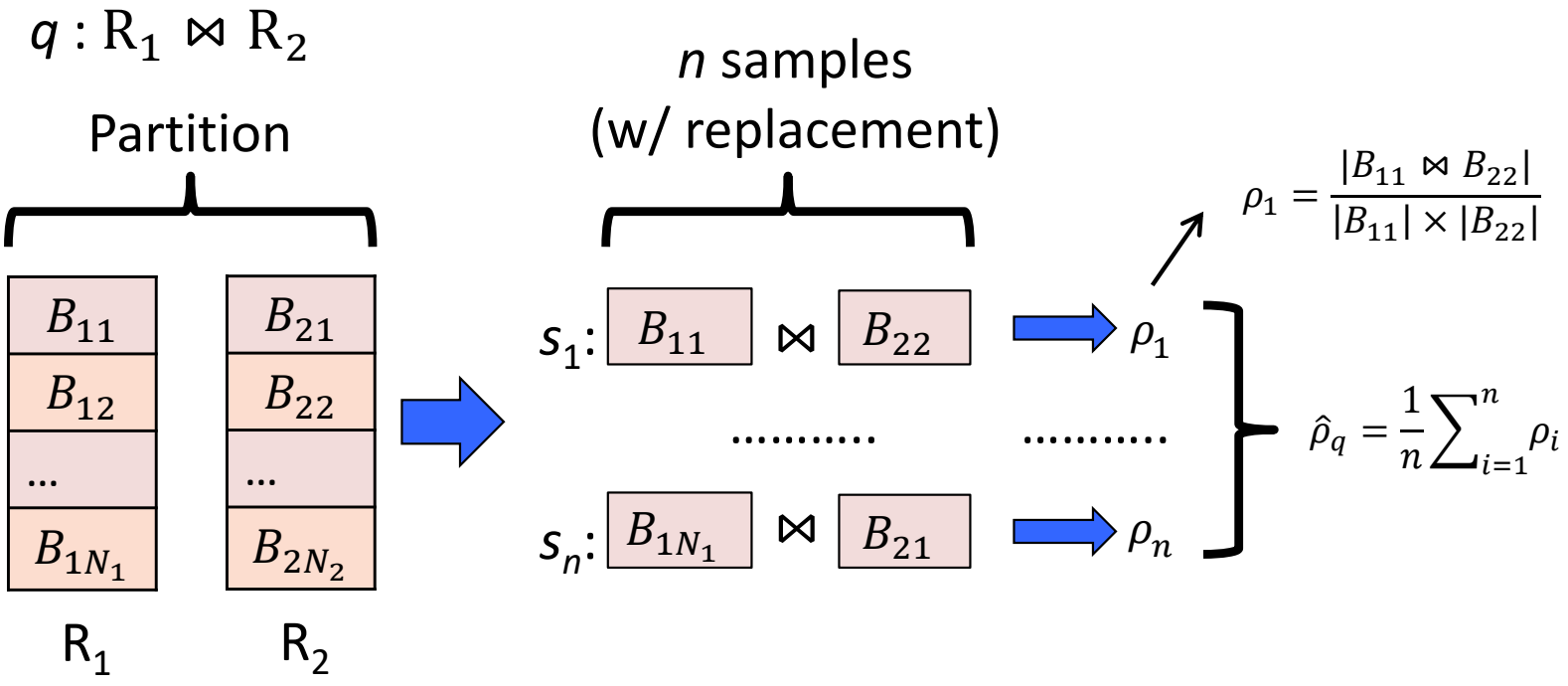
Refine Cardinality Estimates

- Different perspective than the norm (query optimization)

	Query Optimization	**Execution Time Prediction**
# of Plans	Hundreds/Thousands	1
Time per Plan	Must be very short	Can be a bit *longer*
Precision	Important	*Critical*
Approach	Histograms (dominant)	*Sampling* (one option)

A Sampling-Based Estimator

- Estimate the *selectivity* ρ_q of a select-join query q.
  [Haas et al., J. Comput. Syst. Sci. 1996]



The estimator $\hat{\rho}_q$ is *unbiased* and *strongly consistent*

Cardinality Refinement Algorithm

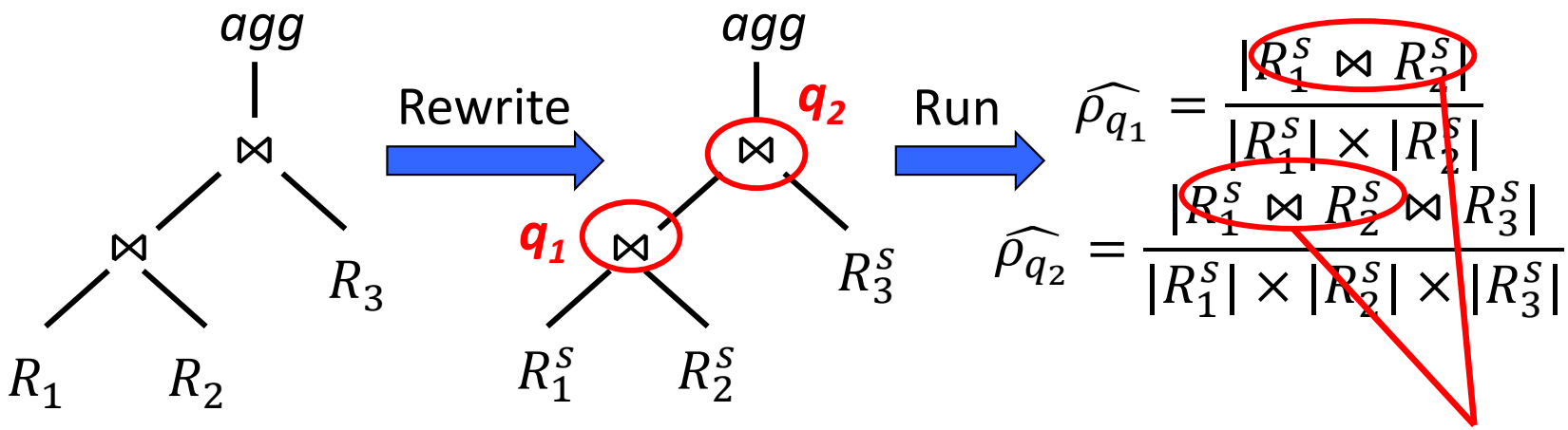
- Design the refinement algorithm based on the previous sampling formula.

Problem	Solution
The estimator needs *random* I/Os at *runtime* to take samples.	Take samples *offline* and store them as tables in the database.
Query plans usually contain *more than one* operator.	Estimate multiple operators in a *single* run, by *reusing* partial results.
The estimator only works for *select/join* operators.	Rely on PostgreSQL's cost models for *aggregates*.

Cardinality Refinement Algorithm (Example)

Plan for q:

$$q_1 = R_1 \bowtie R_2$$
$$q_2 = R_1 \bowtie R_2 \bowtie R_3$$



R_1^s, R_2^s, R_3^s are samples (as tables) of R_1, R_2, R_3

For *agg*, use PostgreSQL's estimates based on the *refined* input estimates from q_2.

Experimental Settings

- PostgreSQL 9.0.4, Linux 2.6.18
- TPC-H 1GB and 10GB databases
  - Both uniform and skewed data distribution
- Two different hardware configurations
  - PC1: 1-core 2.27 GHz Intel CPU, 2GB memory
  - PC2: 8-core 2.40 GHz Intel CPU, 16GB memory

Calibrating Cost Units

PC1:

Cost Unit	Calibrated (ms)	Calibrated (normalized to c_s)	Default
c_s: seq_page_cost	5.53e-2	1.0	1.0
c_r: rand_page_cost	6.50e-2	1.2	4.0
c_t: cpu_tuple_cost	1.67e-4	0.003	0.01
c_i: cpu_index_tuple_cost	3.41e-5	0.0006	0.005
c_o: cpu_operator_cost	1.12e-4	0.002	0.0025

PC2:

Cost Unit	Calibrated (ms)	Calibrated (normalized to c_s)	Default
c_s: seq_page_cost	5.03e-2	1.0	1.0
c_r: rand_page_cost	4.89e-1	9.7	4.0
c_t: cpu_tuple_cost	1.41e-4	0.0028	0.01
c_i: cpu_index_tuple_cost	3.34e-5	0.00066	0.005
c_o: cpu_operator_cost	7.10e-5	0.0014	0.0025

Prediction Precision

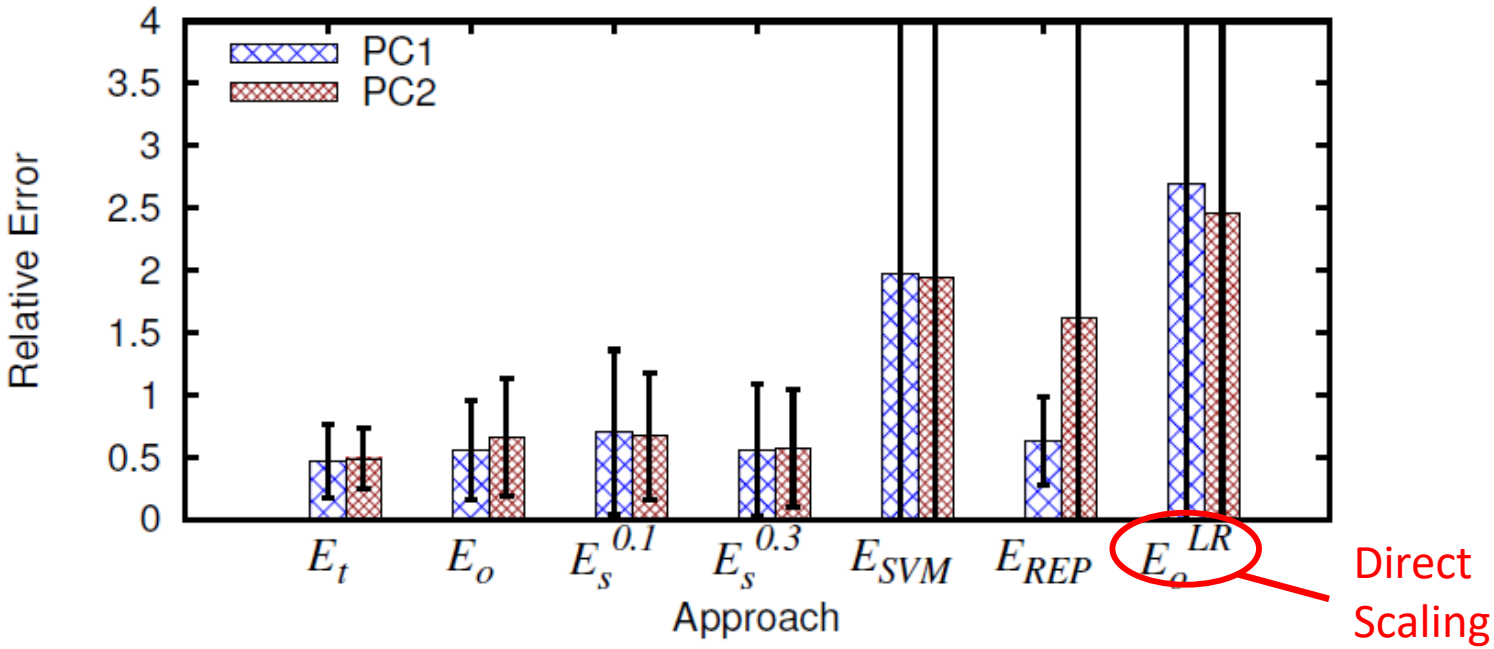
- Metric of precision
  - Mean Relative Error (MRE)
  - (questionable as compared to q-error)

$$\frac{1}{M}\sum_{i=1}^{M}\frac{|T_i^{pred} - T_i^{act}|}{T_i^{act}}$$

- Dynamic database workloads
  - Unseen queries frequently occur
- Compare with existing approaches
  - Direct scaling
  - Machine learning approaches

Precision on TPC-H 1GB DB

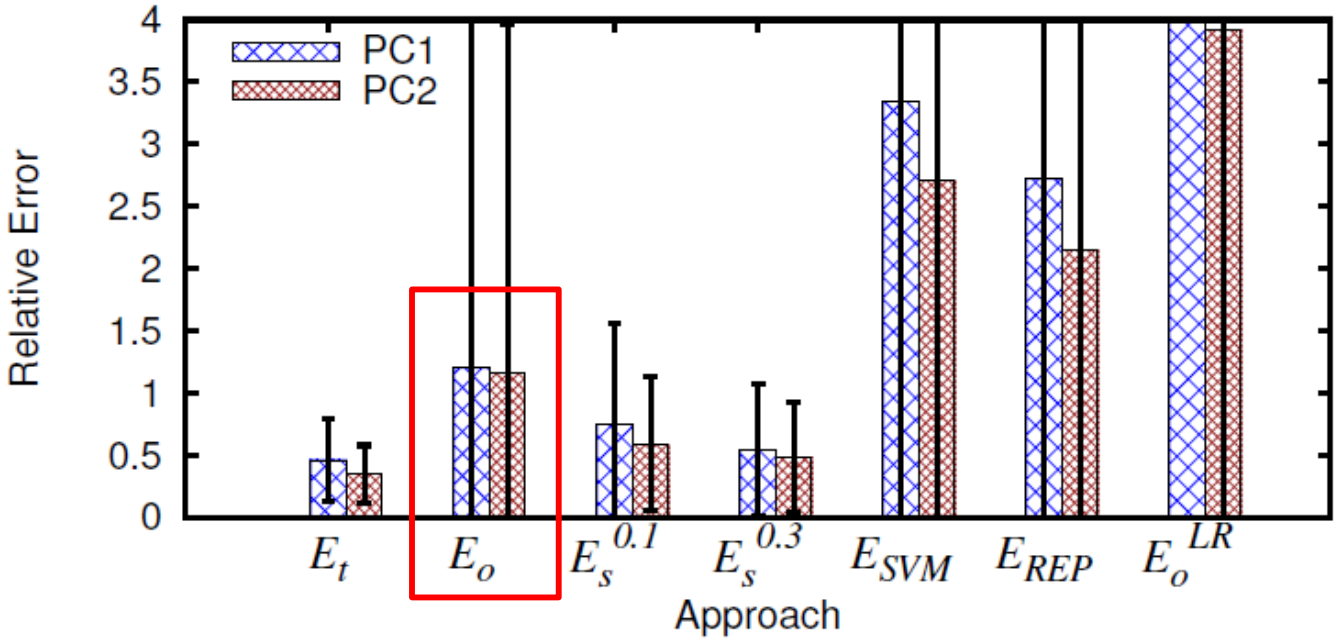
Uniform data:



E_t: *c*'s (calibrated) + *n*'s (*true* cardinalities)
E_o: *c*'s (calibrated) + *n*'s (cardinalities by *optimizer*)
E_s: *c*'s (calibrated) + *n*'s (cardinalities by *sampling*)

Precision on TPC-H 1GB DB (Skewed)

Skewed data:



E_t: *c*'s (calibrated) + *n*'s (*true* cardinalities)
E_o: *c*'s (calibrated) + *n*'s (cardinalities by *optimizer*)
E_s: *c*'s (calibrated) + *n*'s (cardinalities by *sampling*)

Summary

- Systematic framework to calibrate the cost units and refine the cardinality estimates used by current cost models.

- Showed that current statistical cost models are quite effective in query execution time prediction after proper calibration, and the additional overhead is affordable in practice.

Stage 5: ML Approaches

Motivation

- Over the past three years, a flood of publications [16, 22, 23, 27, 28, 34, 37, 40, 48, 49, 50, 51, 52, 53, 54, 56, 57, 59, … !] advocating deep-learning-based approaches for both cardinality-estimation and cost-estimation.

- Basic idea is to replace coarse parametrized models with fine-grained learnt models. The expectation is that these deep models are better able to capture the in situ data and system behavior due to their flexibility, scalability and lack of prior assumptions.

Approaches

- Two broad classes
  - query-based (supervised learning)
    - Models constructed by training on a large set of queries and leveraging the observed values during execution as labels
  - data-based (unsupervised learning)
    - Model the joint probability density functions of the underlying data to capture distributions and correlations

MSCN [28]

(Multi-set Convolutional Neural Network)

Framework

- Estimating cardinalities for correlated joins, since they are especially hard to model well
  - e.g. French actors are more likely to participate in romantic movies than actors of other nationalities
- Key Ideas
  - Set-based model (based on DeepSets):
    $(A \bowtie B) \bowtie C$ and $A \bowtie (B \bowtie C)$ are both represented as {A, B, C}
  - Integrates sampling:
    use bitmaps of qualifying base table samples as ML features
- Advantages
  - Learns join-crossing correlations
  - Addresses "0-tuple" situations: model relies on query features in cases when no or very few samples qualify

1) Obtaining Training Data

- Generate synthetic queries using schema information (data types and constraints) and the actual values from the database
- Execute queries on a snapshot of the database to obtain true cardinalities
- Annotate queries with bitmaps indicating qualifying base table samples

2) Feature Selection and Representation

- Query features (tables, predicates, joins, ...) are one-hot encoded
- Values (literals) and true cardinalities are normalized to [0,1]

SELECT * FROM title t, movie_companies mc WHERE t.id = mc.movie_id

Table set { [0 1 0 1 … 0], [0 0 1 0 … 1] } Join set { [0 0 1 0] }

table id samples join id

AND t.production_year > 2010 AND mc.company_id = 5

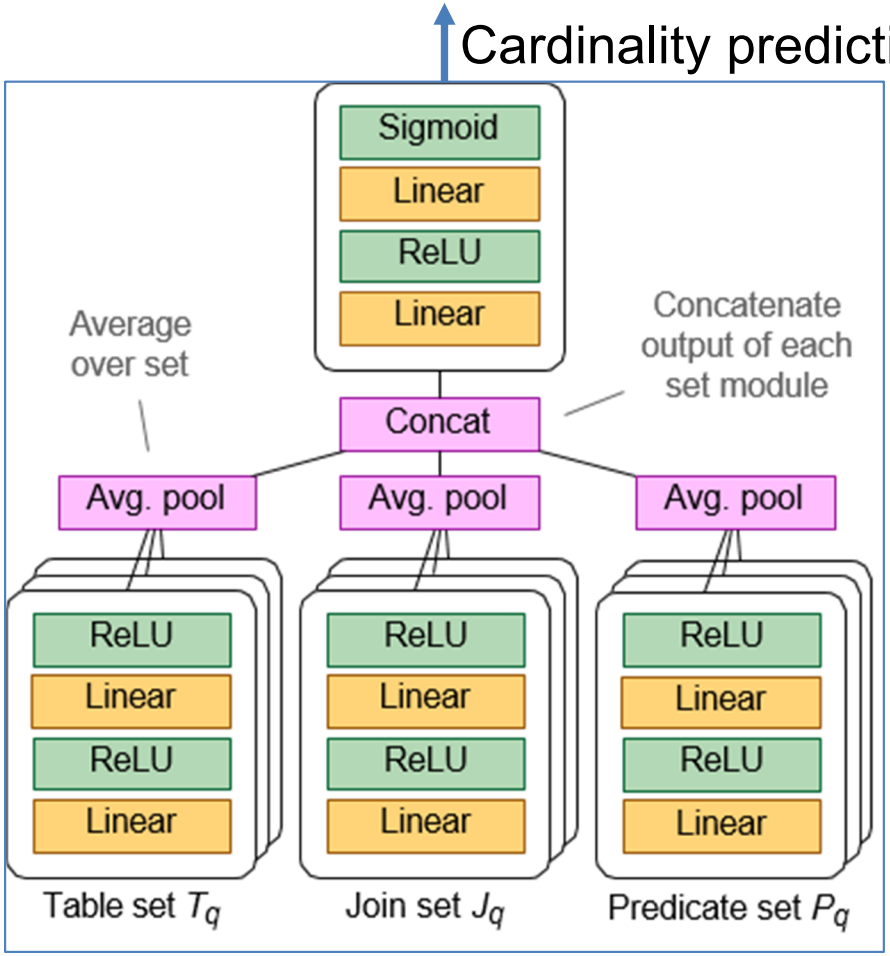
Predicate set { [1 0 0 0 0 1 0 0 0.72], [0 0 0 1 0 0 1 0 0.14] }

column id op. id value

- True cardinality (label): 665 (encoded as 0.1 if max = 6650)

3) Set-based ML model

- Four fully-connected multi-layer neural networks (MLPs)



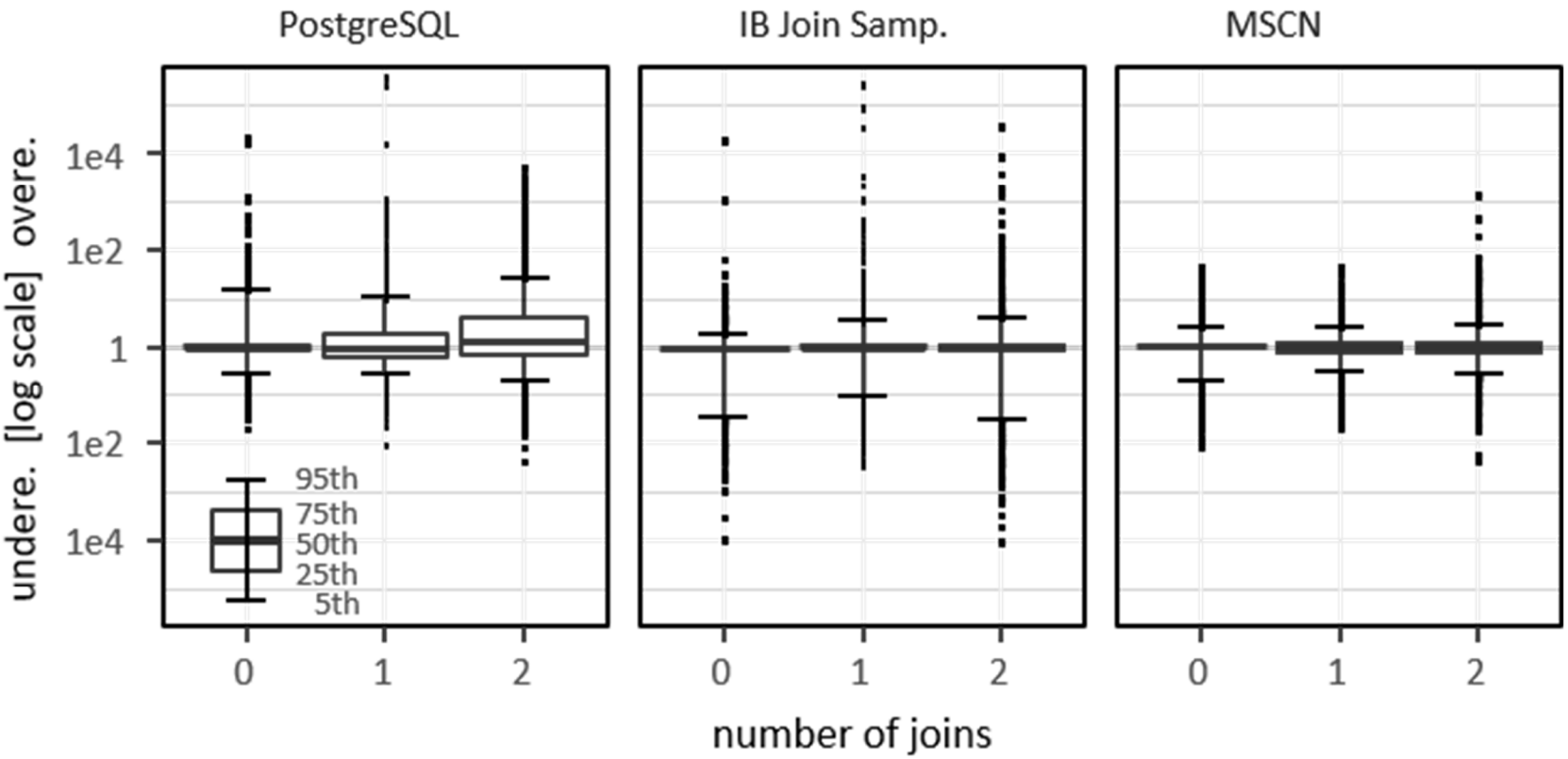
4) Optimization Metric

- q-error: multiplicative ratio between true and estimated cardinalities.

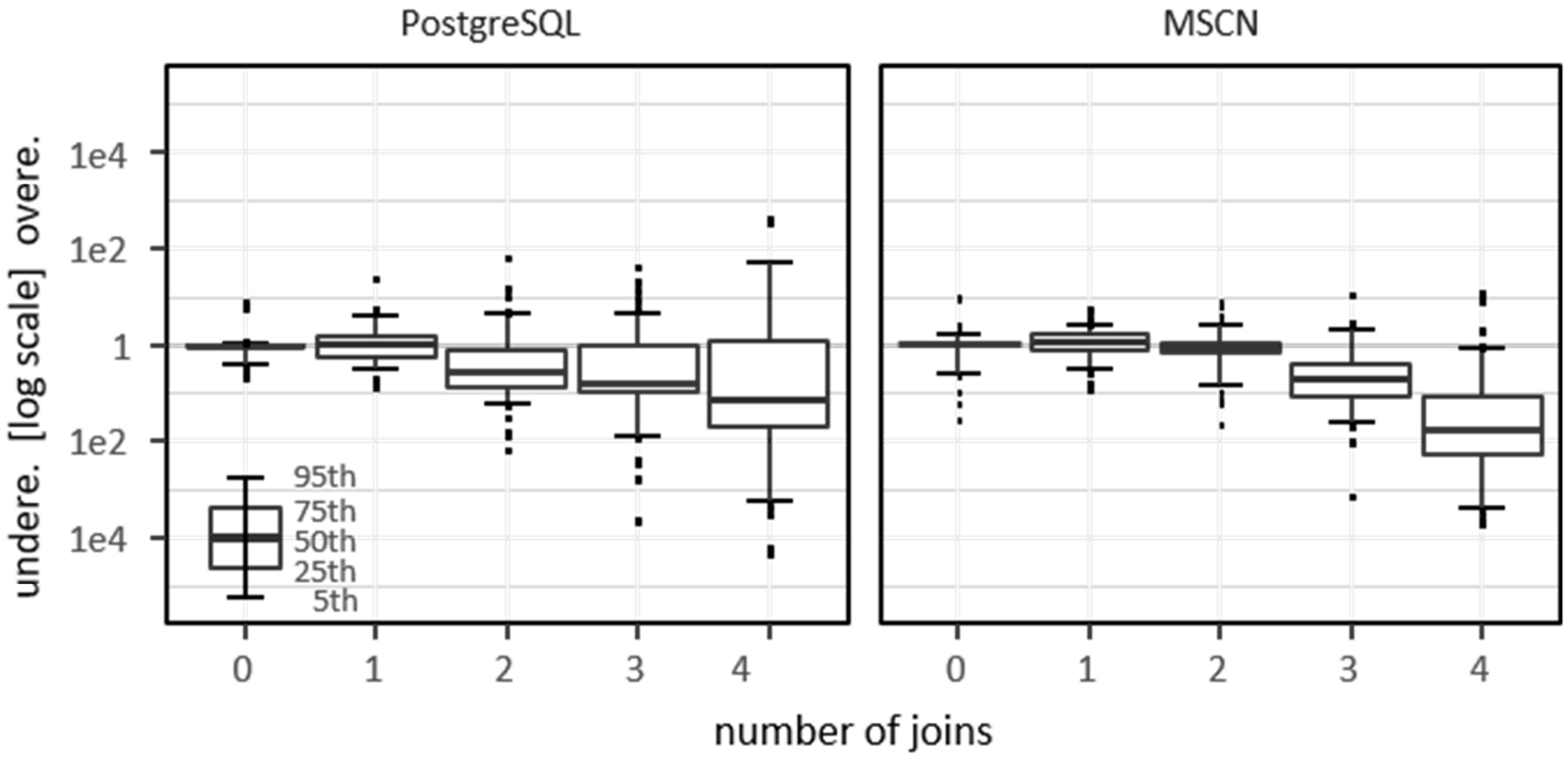
- Goal: minimize mean q-error over the training set

Estimation Quality

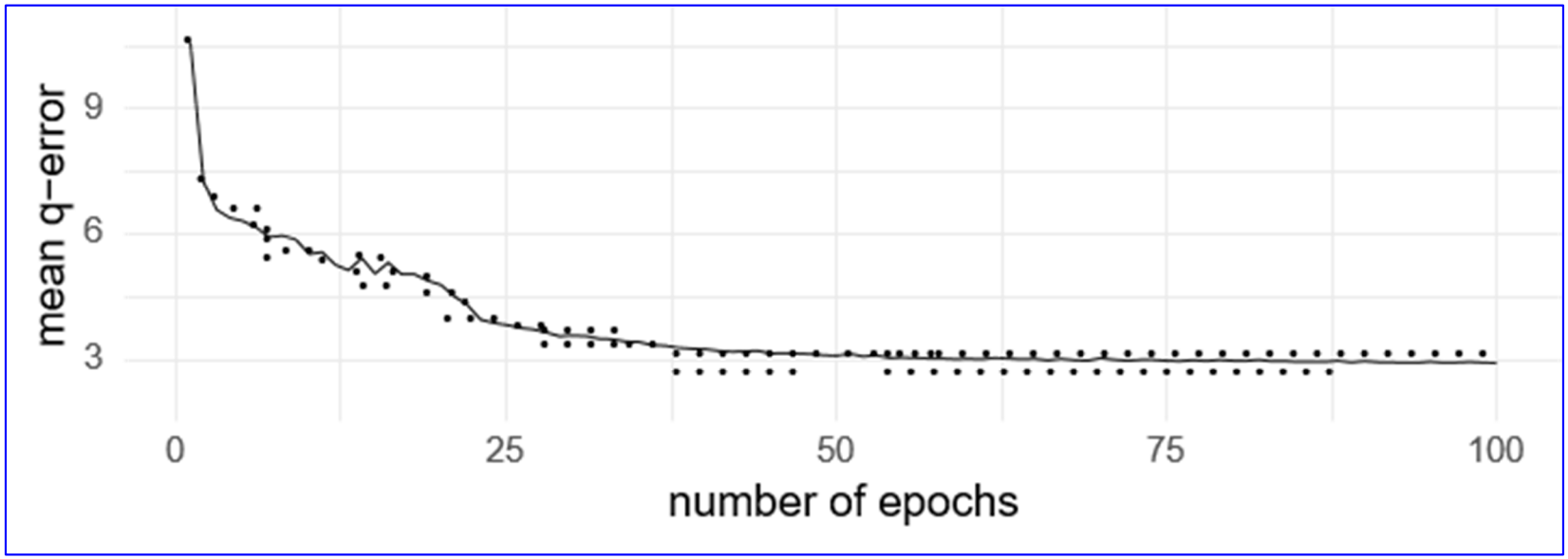
- IMDB data-set: contains many correlations
- Synthetic queries: only equality and range predicates



Generalizing to More Joins



Training Convergence



Summary

- Deep learning can capture complex correlations and address limitations of pure sampling when there is a good match between the training and testing environments

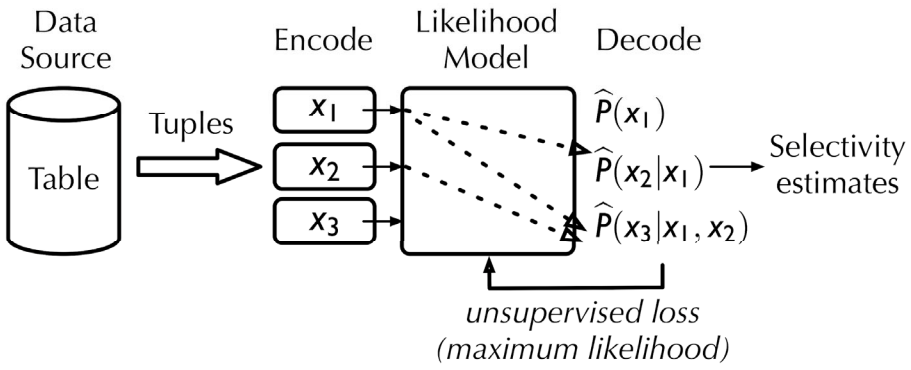
NARU [48]

(Neural Relation Understanding)

Learning Model

Training

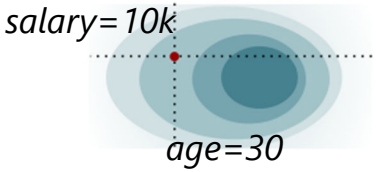
Learn joint data distribution with deep autoregressive model



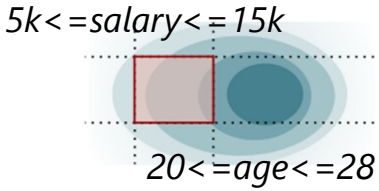
Inference

Monte Carlo integration to answer range density queries

Point query



Range query



Joint Distribution

Age	Salary
25	2000
25	10,000
24	2000
24	2000

Data
(strings, nums, dates, …)

induces →

Age	Salary	P(A,S)
25	2000	1/4
25	10,000	1/4
24	2000	2/4
everything else		0

Joint Distribution (P)

%rows matched by the query?

```
SELECT * FROM T
WHERE Age <= 25 AND Salary <= 2000
```

selectivity(Q) ≡ density(Q):
density(Age<=25 && Salary<=2000)

Integrating the joint yields density(Q):

Valid Age: [24, 25]; Valid Salary: [2000].
Sum up the densities from valid points.

= P(25, 2000) + P(24, 2000)
= 1/4 + 2/4 = 0.75

Learning the Joint Distribution

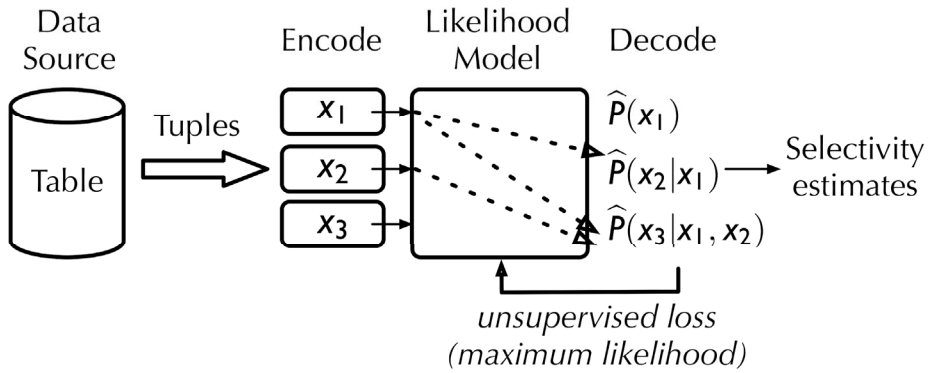
Age	Salary	P(A,S)
25	2000	1/4
25	10,000	1/4
24	2000	2/4
everything else		0

Joint Distribution (P)

Use a *deep autoregressive model* to learn:

$$P(x) = \prod_{i=1}^{n} P(x_i | x_{<i})$$

where **x** is an n-dimensional tuple.



Calculation becomes

density(Age<=25 && Salary<=2000)
~= Model(25, 2000) + Model(24, 2000)

Learning the Joint

Age	Salary	P(A,S)
25	2000	1/4
25	10,000	1/4
24	2000	2/4
everything else		0

Joint Distribution (P)

Not materialized; Emitted **on-demand** by model

Use a *deep autoregressive model* to learn:

$$P(x) = \prod_{i=}^{n} P(x_i | x_{<i})$$

where **x** is an n-dimensional tuple.

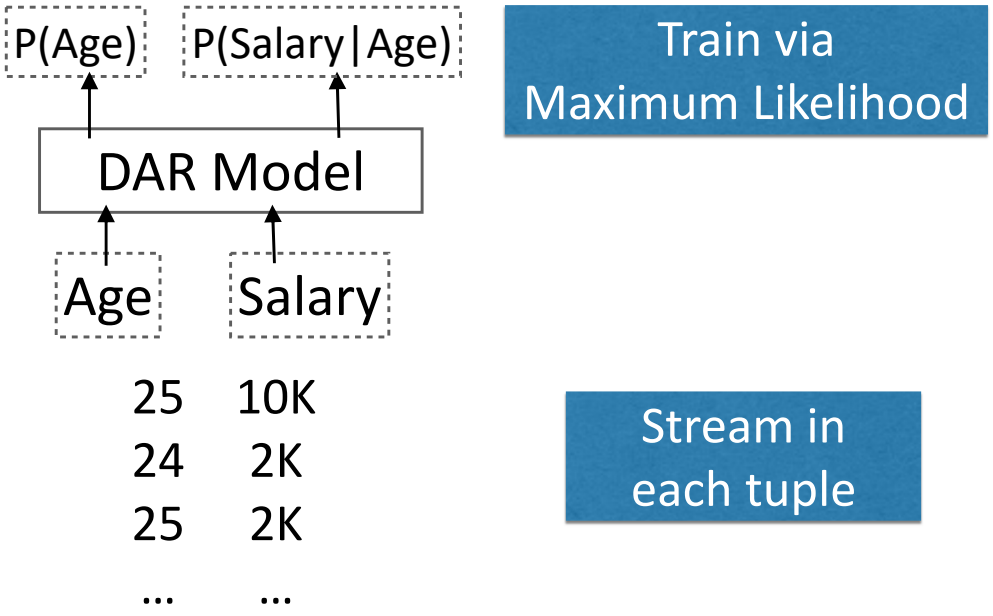
Compared to previous work:
chain-rule factorization means **no** information loss

Independence assumption **loses** information

- 1D Histogram: P(A,B,C) ~= P(A) P(B) P(C)
- Partial Independence: P(A,B,C) ~= P(A,B) P(C)
- Conditional Independence: P(A,B,C) ~= P(B) P(C|B) P(A|C)

Model Training

Output: probability distributions over columns

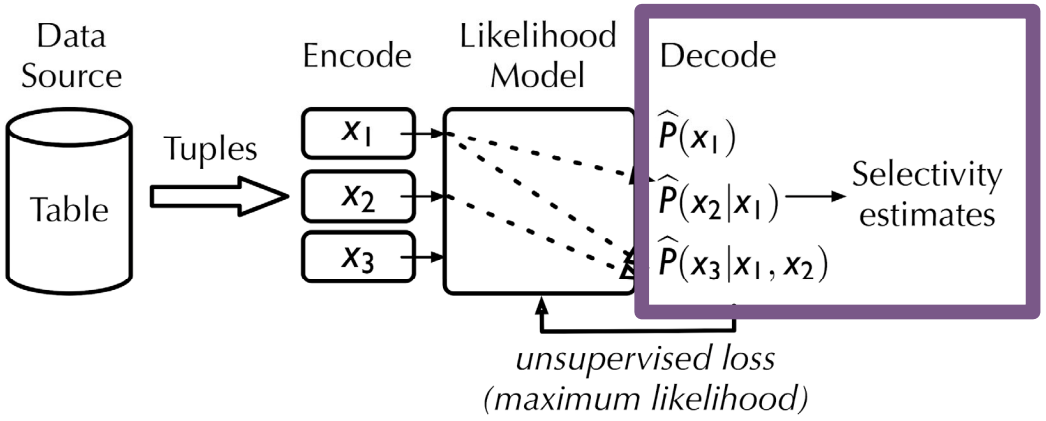


Input: each tuple

Plug in any deep autoregressive model:

- Masked MLP
  - MADE [ICML'15]
  - ResMADE [Nash et al. '19]
- Transformer [NIPS'17] and variants
- WaveNet
- …

Range Density Estimates



DAR model outputs point density.
Require range density at inference time.

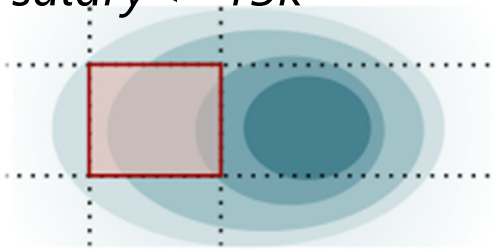
density(Age<=25 && Salary<=2000)
2 valid points to forward pass

density(X1 in R1 ,..., Xn in Rn)
has |R1| x |R2| x … x |Rn| points
(exponential)

Insight: not all points in the queried region are meaningful
→ use **Monte Carlo integration** (sampling) to approximate range density

Approximate Inference

5k<=salary<=15k



20<=age<=28

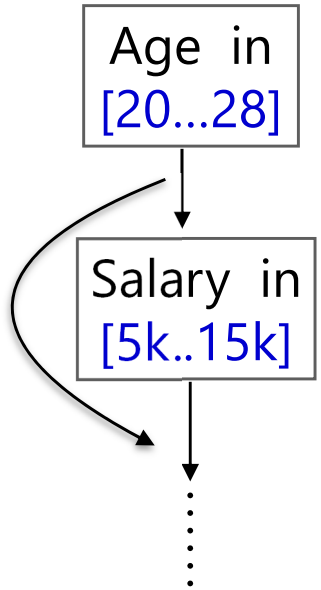
P(Age in [20…28] && Salary in [5k…15k])

Exact inference: exponential in #columns of the table

Progressive Sampling

sample X1
~ P(X1 | X1 in R1)

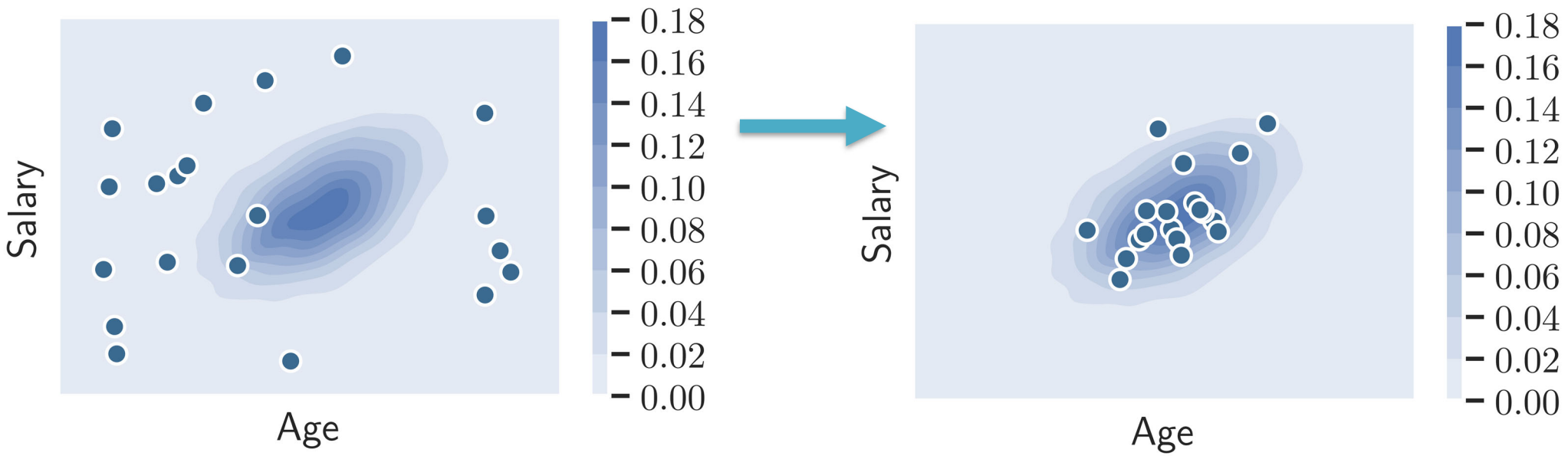
sample X2
~ P(X2 | X2 in R2, x1_sample)



Weight densities appropriately

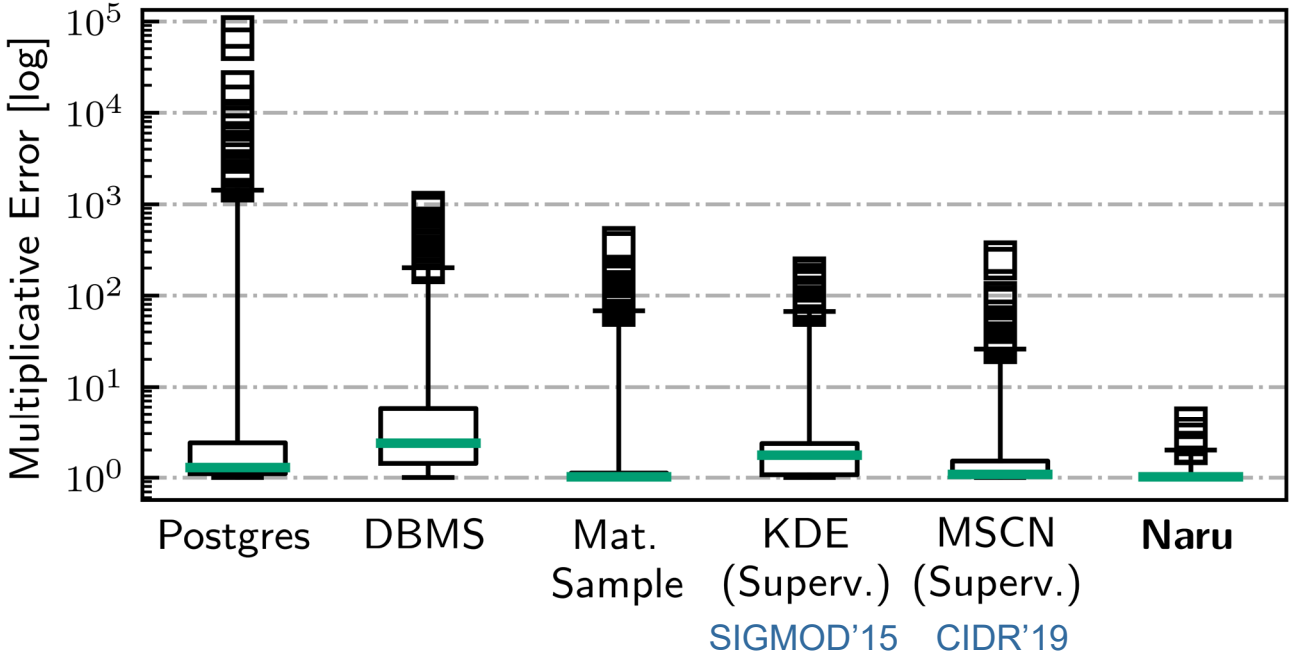
Use sample from each dimension to *progressively zoom into* high-mass region

Progressive Sampling



Estimation Accuracy

- Supervised: Few hours to collect 10K training queries
- Unsupervised: Few minutes to read and train



Dataset DMV (11M tuples, 11 columns)
Workload 5-11 range/eq filters; 2K queries
Model Masked MLP (#params: 3M; ~1% data size)

Limitations of Learning approaches

- Universality
  - Ability to handle unseen adhoc queries is suspect
- Explainability
  - Do not provide an intuitive confirmation of the approach
- Guarantees
  - Average case may be excellent, but worst-case can be arbitrarily poor
- Heavy-weight
  - May require expensive training phase
- Uncertainty estimation
  - Hard to quantify the risk involved in trusting the model

Open Problem:

Compare Algorithmic (Algebra+Geometry) vs Function-fitting approaches

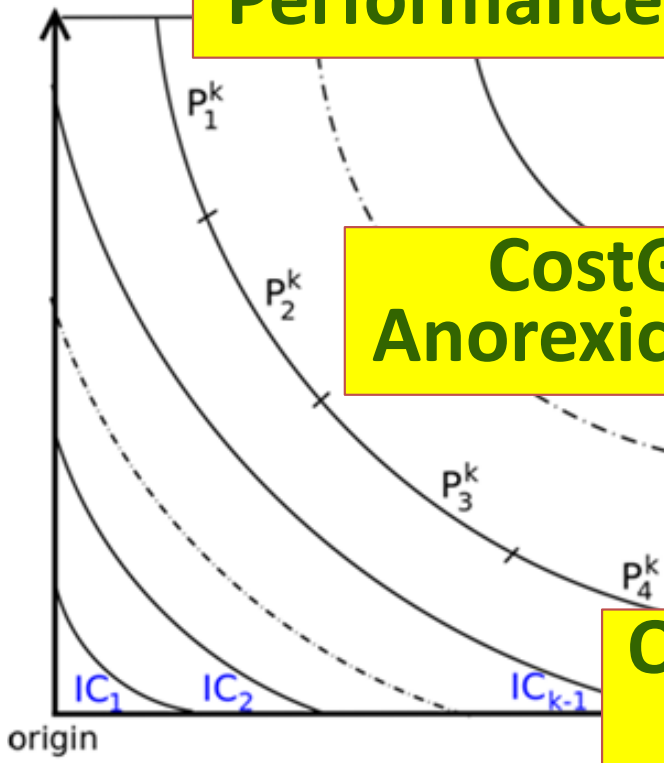
Putting it all together

Good news

The proposed techniques are complementary and can work together!

New RQP Architecture: Plan-level

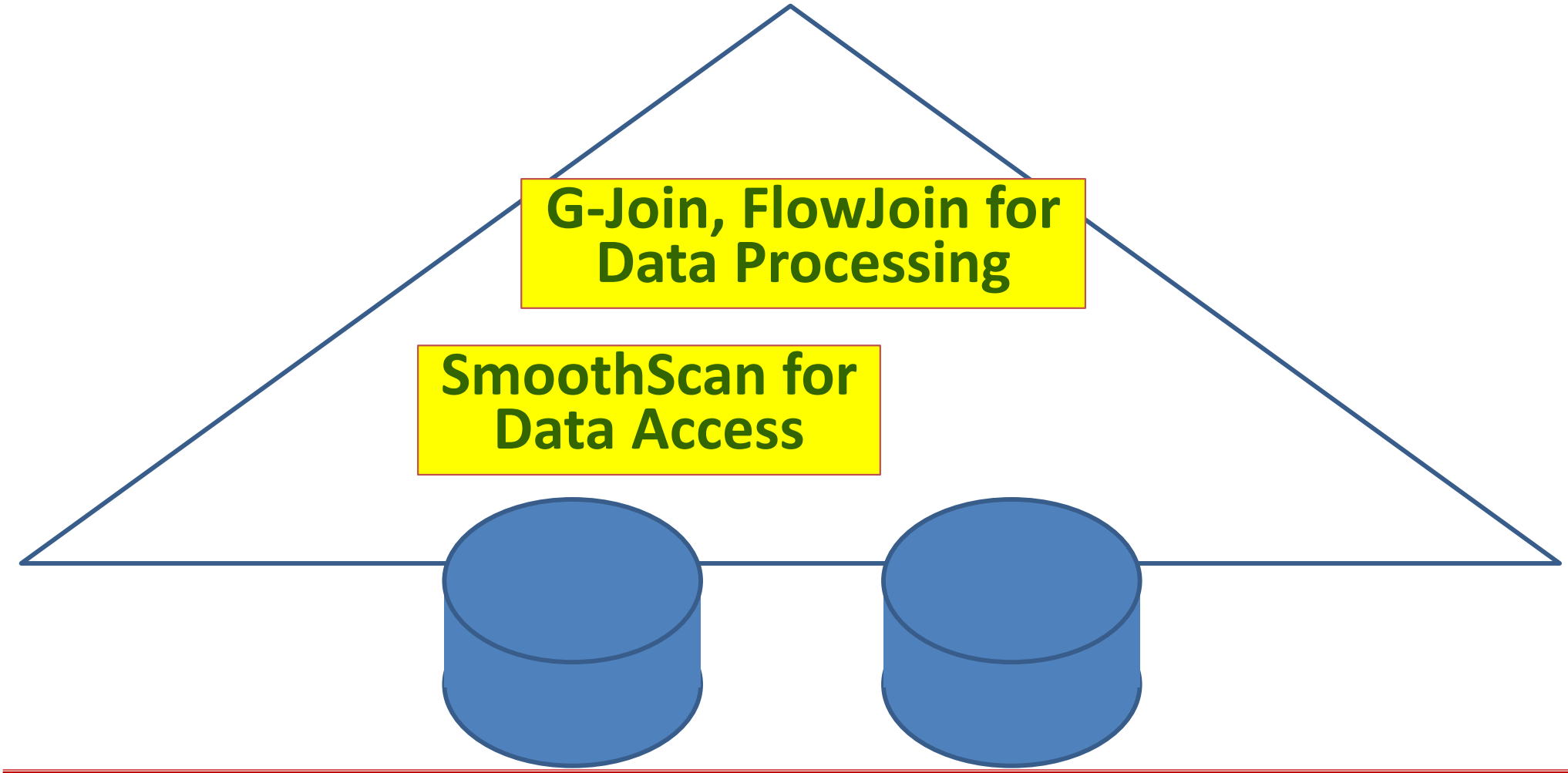
SpillBound for Performance Guarantees



CostGreedy for Anorexic Plan Density

Q-Calibrated + Naru-Sampled Cost Model for Contours

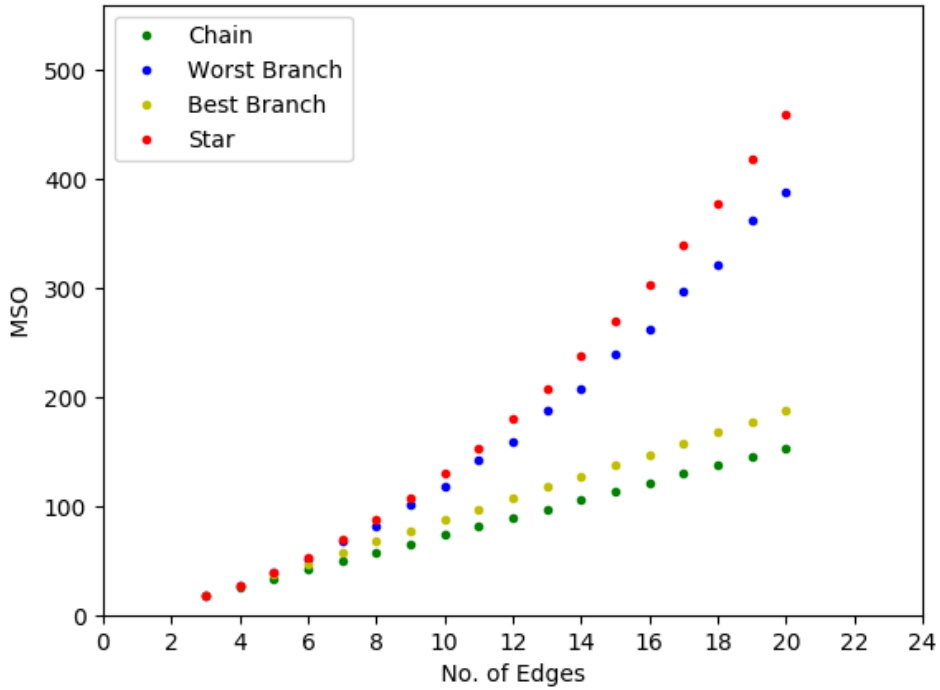
New RQP Architecture: Intra-Plan



Stage 6: Future Research

1) Structure of Query Graphs

- Graph structure (chain, star, cycle, etc.) has significant impact on robustness guarantees
  - Tighter guarantees for chain (8D – 6) as compared to star (D^2+3D)



Open Problem: MSO derivations based on query graph type

2) Refined Cost Model Calibration

- Calibration discussed previously assumed the Postgres basic 5-parameter model as a given for the entire suite of operators.

- **Open Problem:** Add operator-specific features and operator-specific calibration of the coefficients, and see if accuracy can be improved.

3) Robustness Benchmarks

- Standard industry benchmarks (e.g. TPC-DS) are oriented towards performance, not robustness.
- Recent proposals on benchmarks:
  - Optimizer Benchmark (OptMark) (CIKM 16 [28])
    - TPC-DS synthetic data, examines plan coverage and estimation of plans better than optimizer's choice; does not cover magnitude of cost differences
  - Join-Order Benchmark (JOB) (VLDBJ 18 [26])
    - Based on IMDB real data with heavy skew and correlation, and join-heavy queries, q-error
  - Optimizer Torture Test (OTT) (SIGMOD 16 [43])
    - Two-column relations, one join attribute and one selection, the two columns are highly correlated (in fact, identical values!)
- **Open Problem: Design non-pathological realistic benchmarks that highlight robustness issues (e.g. performance cliffs)**

END RQP TUTORIAL

BIBLIOGRAPHY

REFERENCES

1. Robust Query Processing. Dagstuhl Seminar, 2010. www.dagstuhl.de/en/program/calendar/semhp/?semnr=10381.
2. Robust Query Processing. Dagstuhl Seminar, 2012. www.dagstuhl.de/en/program/calendar/semhp/?semnr=12321.
3. Robust Performance in Database Query Processing. Dagstuhl Seminar, 2017. www.dagstuhl.de/en/program/calendar/semhp/?semnr=17222.
4. M. Akdere, U. Cetintemel, M. Riondato, E. Upfal, S. Zdonik. Learning-based query performance modeling and prediction. ICDE, 2012.
5. R. Avnur, J. Hellerstein. Eddies: Continuously Adaptive Query Processing. SIGMOD, 2000.
6. S. Babu, P. Bizarro, D. DeWitt. Proactive Re-optimization. SIGMOD, 2005.
7. R. Borovica-Gajic, S. Idreos, A. Ailamaki, M. Zukowski, C. Fraser. Smooth Scan: Robust Access Path Selection without Cardinality Estimation. VLDBJ, 27(4), 2018.
8. S. Chaudhuri. An Overview of Query Optimization in Relational Systems. PODS, 1998.
9. S. Chaudhuri. Query Optimizers: Time to rethink the contract? SIGMOD, 2009.
10. S. Chaudhuri. Interview in XRDS. 19(1), 2012.
11. F. Chu, J. Halpern, P. Seshadri. Least Expected Cost Query Optimization: An Exercise in Utility. PODS, 1999.
12. F. Chu, J. Halpern, J. Gehrke. Least Expected Cost Query Optimization: What can we expect? PODS, 2002.
13. D. Dewitt. Interview in Sigmod Record. 31(2), 2002.
14. A. Dutt, J. Haritsa. Plan Bouquets: A Fragrant Approach to Robust Query Processing. ACM TODS, 41(2), 2016.

REFERENCES (contd)

15. A. Dutt, V. Narasayya, S. Chaudhuri. Leveraging re-costing for online optimization of parameterized queries with guarantees. SIGMOD, 2017.
16. A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya, S. Chaudhuri. Selectivity estimation for range predicates using lightweight models. PVLDB, 12(9), 2019.
17. G. Graefe. New algorithms for join and grouping operations. Computer Science – R&D, 27(1), 2012.
18. Harish, D., P. Darera, J. Haritsa. On the Production of Anorexic Plan Diagrams. VLDB, 2007.
19. Harish, D., P. Darera, J. Haritsa. Identifying Robust Plans through Plan Diagram Reduction. PVLDB, 1(1), 2008.
20. H. Harmouch, F. Naumann. Cardinality Estimation: An Experimental Survey. PVLDB, 11(4), 2017.
21. S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, G. Das. Deep Learning Models for Selectivity Estimation of Multi-attribute Queries. SIGMOD, 2020.
22. D. Havenstein, P. Lysakovski, N. May, G. Moerkotte, G. Steidl. Fast Entropy Maximization for Selectivity Estimation of Conjunctive Predicates on CPUs and GPUs. EDBT, 2020.
23. R. Hayek and O. Shmueli. Improved Cardinality Estimation by Learning Queries Containment Rates. EDBT, 2020.
24. N. Kabra, D. DeWitt. Efficient Mid-Query Re-Optimization of Sub- Optimal Query Execution Plans. SIGMOD, 1998.
25. S. Karthik, J. Haritsa, S. Kenkre, V. Pandit, L. Krishnan. Platform-independent Robust Query Processing. IEEE TKDE, 31(1), 2019.
26. S. Karthik, J. Haritsa, S. Kenkre, V. Pandit. A Concave Path to Low-overhead Robust Query Processing. PVLDB, 11(13), 2018.

REFERENCES (contd)

27. M. Kiefer, M. Heimel, S. Bress, V. Markl. Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models. PVLDB, 10(13), 2017.
28. A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, A. Kemper. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. CIDR, 2019.
29. V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, T. Neumann. Query Optimization through the looking glass, and what we found running the Join Order Benchmark. VLDBJ, 27(5), 2018.
30. V. Leis, B. Radke, A. Gubichev, A. Kempers, T. Neumann. Cardinality Estimation Done Right: Index-based Join Sampling. CIDR, 2017.
31. Z. Li, O. Papaemmanouil, M. Cherniack. OptMark: A Toolkit for Benchmarking Query Optimizers. CIKM, 2016.
32. G. Lohman. Is Query Optimization a Solved Problem? ACM Sigmod Blog, 2014. wp.sigmod.org/?p=1075.
33. T. Malik, R. Burns, N. Chawla. A Black-Box Approach to Query Cardinality Estimation. CIDR, 2007.
34. R. Marcus, O. Papaemmanouil. Towards a Hands-Free Query Optimizer through Deep Learning. CIDR, 2019.
35. V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, M. Cilimdzic. Robust query processing through progressive optimization. SIGMOD, 2004.
36. G. Moerkotte, T. Neumann, G. Steidl. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. PVLDB, 2(1), 2009.
37. T. Neumann, B. Radke. Adaptive Optimization of Very Large Join Queries. SIGMOD, 2018.

REFERENCES (contd)

38. K. Ramachandra, K. Park, K. Emani, A. Halverson, C. Galindo-Legaria, C. Cunningham. Froid: Optimization of Imperative Programs in a Relational Database. PVLDB, 11(4), 2017.
39. W. Roediger, S. Idicula, A. Kemper, T. Neumann. Flow-join: Adaptive skew handling for distributed joins over high-speed networks. ICDE, 2016.
40. J. Sun and G. Li. An End-to-End Learning-based Cost Estimator. PVLDB, 13(3), 2019.
41. K. Tzoumas, A. Deshpande, C. Jensen. Efficiently adapting graphical models for selectivity estimation. VLDBJ, 22(1), 2013.
42. J. Wiener, H. Kuno, G. Graefe. Benchmarking Query Execution Robustness. TPCTC, 2009.
43. F.Wolf, M. Brendle, N. May, P. Willems, K. Sattler, M. Grossniklaus. Robustness Metrics for Relational Query Execution Plans. PVLDB, 11(11), 2018.
44. W. Wu, Y. Chi, H. Hacigumus, J. Naughton. Towards predicting query execution time for concurrent, dynamic databae workloads. PVLDB, 6(10), 2013.
45. W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigumus, J. Naughton. Predicting query execution time: Are optimizer cost models really unusable? ICDE, 2013.
46. W. Wu, J. Naughton, H. Singh. Sampling-based Query Reoptimization. SIGMOD, 2016.
47. W. Wu, X. Wu, H. Hacigumus, J. Naughton. Uncertainty Aware Query Execution Time Prediction. PVLDB, 7(14), 2014.
48. Z. Yang et al. Deep Unsupervised Selectivity Estimation. PVLDB, 13(3), 2019.

Additional References

49. A. Dutt, C. Wang, V. Narasayya, S. Chaudhuri. Efficiently Approximating Selectivity Functions using Low Overhead Regression Models. PVLDB, 13(11), 2020.
50. B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, C. Binnig. Deep DB: Learn from Data, not from Queries! PVLDB, 13 (7), 2020.
51. A. Kipf, M. Freitag, D. Vorona, P. Boncz, T. Neumann, A. Kemper. Estimating Filtered Group-By Queries is Hard: Deep Learning to the Rescue. VLDB AIDB Workshop, 2019.
52. R. Marcus et al. Neo: A Learned Query Optimizer. PVLDB, 12(11), 2019.
53. R. Marcus, O. Papaemmanouil. Plan-Structured Deep Neural Network Models for Query Performance Prediction. PVLDB, 12(11), 2019.
54. M. Mueller, G. Moerkotte, O. Kolb. Improved Selectivity Estimation by Combining Knowledge from Sampling and Synopses. PVLDB, 11(9), 2018.
55. T. Neumann, B. Radke. Adaptive Optimization of Very Large Join Queries. SIGMOD, 2018.
56. J. Ortiz, M. Balazinska, J. Gehrke, S. Keerthi. Learning State Representations for Query Optimization with Deep Reinforcement Learning. SIGMOD DEEM Workshop, 2018.
57. Y.Park, S.Zhong, B. Mozafari. Quicksel: Quick Selectivity Learning with Mixture Models. SIGMOD, 2020.
58. F. Wolf, N. May, P. Willems, K. Sattler. Robustness Metrics for Relational Query Execution Plans. PVLDB 11(11), 2018.
59. L. Woltmann, C. Hartmann, M. Thiele, D. Habich, W. Lehner. Cardinality Estimation with Local Deep Learning Models. SIGMOD aiDM Workshop, 2019.