SCIENCE
PASSION
TECHNOLOGY

Architecture of DB Systems
09 Adaptive Query Processing

Prof. Dr. Matthias Boehm

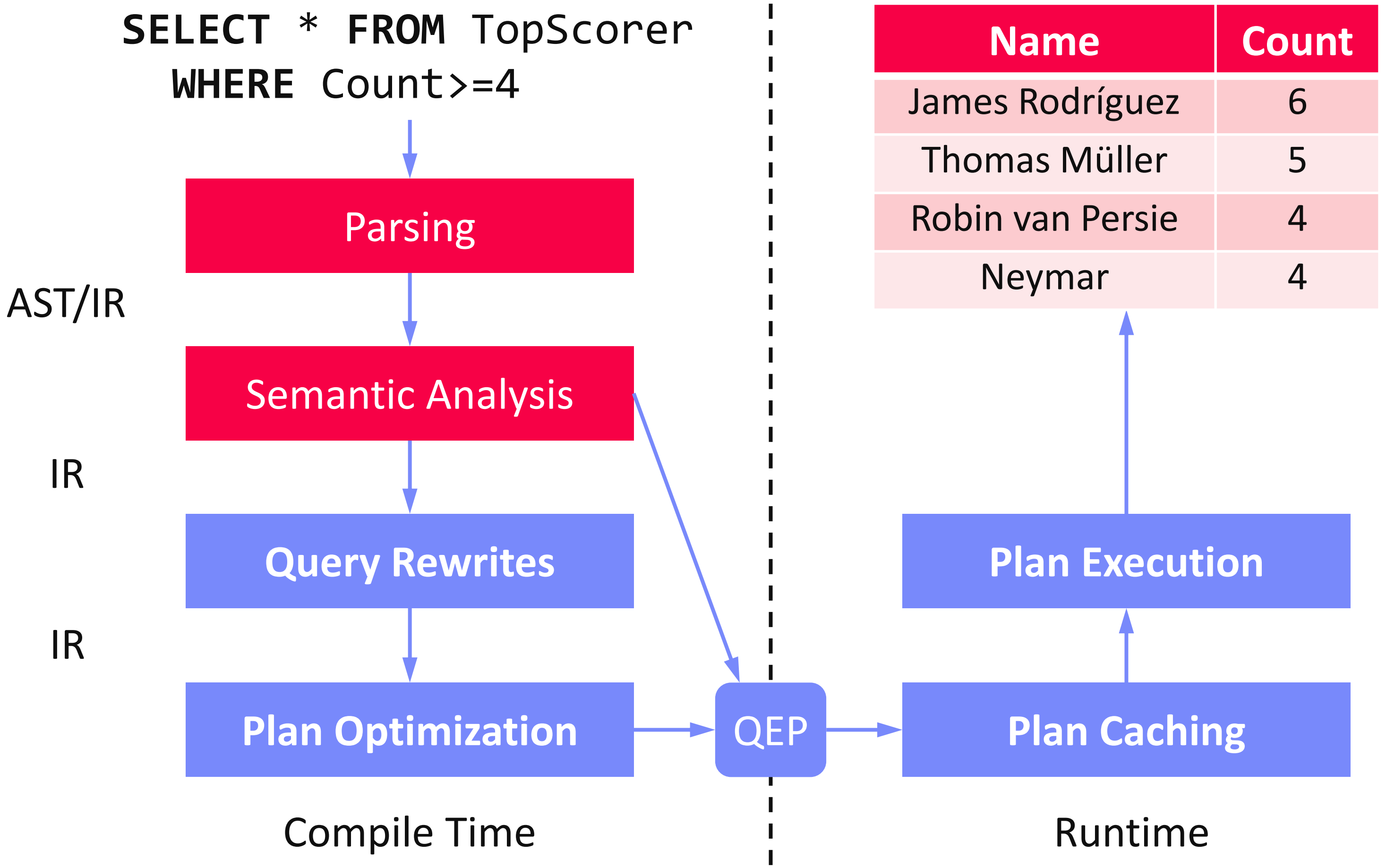
Technische Universität Berlin
Faculty IV - Electrical Engineering and Computer Science
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

Last update: Dec 02, 2023

Announcements/Org

- **#1 Lecture Format**
  - Introduction virtual, remaining lectures blocked **Dec 04 - Dec 07**
  - Optional attendance
  - **Hybrid**, in-person but live-streaming / video-recorded lectures
    - **HS i10** + Zoom: https://tu-berlin.zoom.us/j/9529634787?pwd=R1ZsN1M3SC9BOU1OcFdmem9zT202UT09

Recap: Overview Query Processing

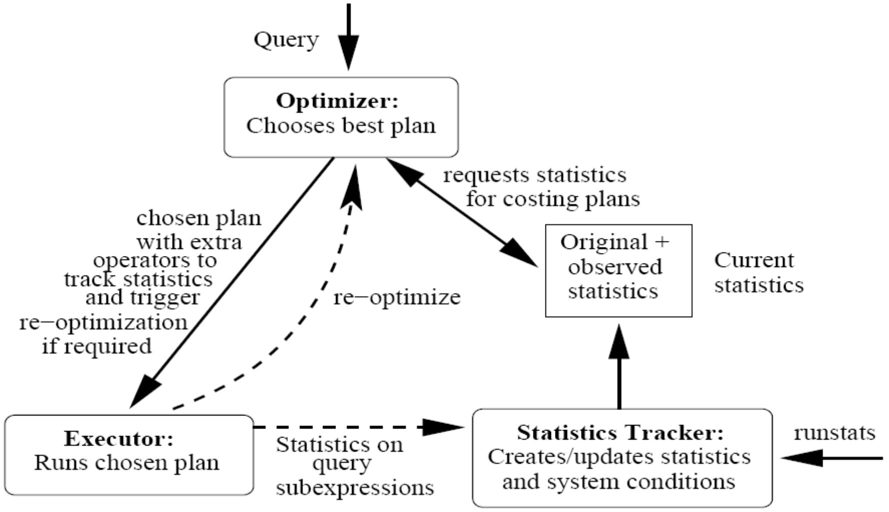


```
SELECT * FROM TopScorer
  WHERE Count>=4
```

Name	Count
James Rodríguez	6
Thomas Müller	5
Robin van Persie	4
Neymar	4

Agenda

- Recap: **Join Enumeration / Ordering**
- **AQP Fundamentals**
- **Learned Cardinalities**
- **Intra-Query Adaptivity**



[Shivnath Babu, Pedro Bizarro: Adaptive Query Processing in the Looking Glass. **CIDR 2005**]

[Amol Deshpande, Zachary G. Ives, Vijayshankar Raman: Adaptive Query Processing. **Found. Trends Databases 1(1) 2007**]

Join Enumeration / Ordering

Plan Optimization Overview

- **Plan Generation Overview**
  - Selection of **physical access path and plan operators**
  - Selection of **execution order** of plan operators (**joins**, group-by)
  - **Input:** logical query plan → **Output:** optimal physical query plan
  - Costs of query optimization should not exceed yielded improvements
- **Interesting Properties**
  - Interesting orders (sorted vs unsorted), partitioning (e.g., join column), pipelining
  - Avoid unnecessary sorting operations

[Ihab F. Ilyas, Jun Rao, Guy M. Lohman, Dengfeng Gao, Eileen Tien Lin: Estimating Compilation Time of a Query Optimizer. **SIGMOD 2003**]

- **Simple Cost Functions**
  - Join-specific cost functions (Cnlj, Chj, Csmj)
  - Cardinalities Cout

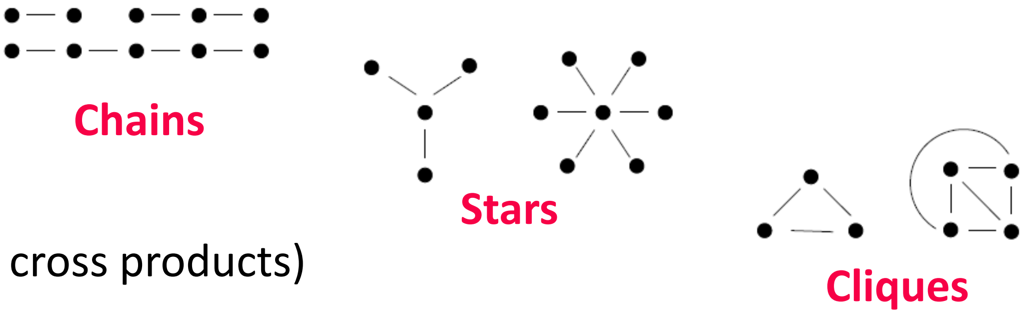
[Guido Moerkotte, Building Query Compilers, **2020**]

$$C_{\text{out}}(T) = \begin{cases} 0 & \text{if } T \text{ is a single relation} \\ |T| + C_{\text{out}}(T_1) + C_{\text{out}}(T_2) & \text{if } T = T_1 \bowtie T_2 \end{cases}$$

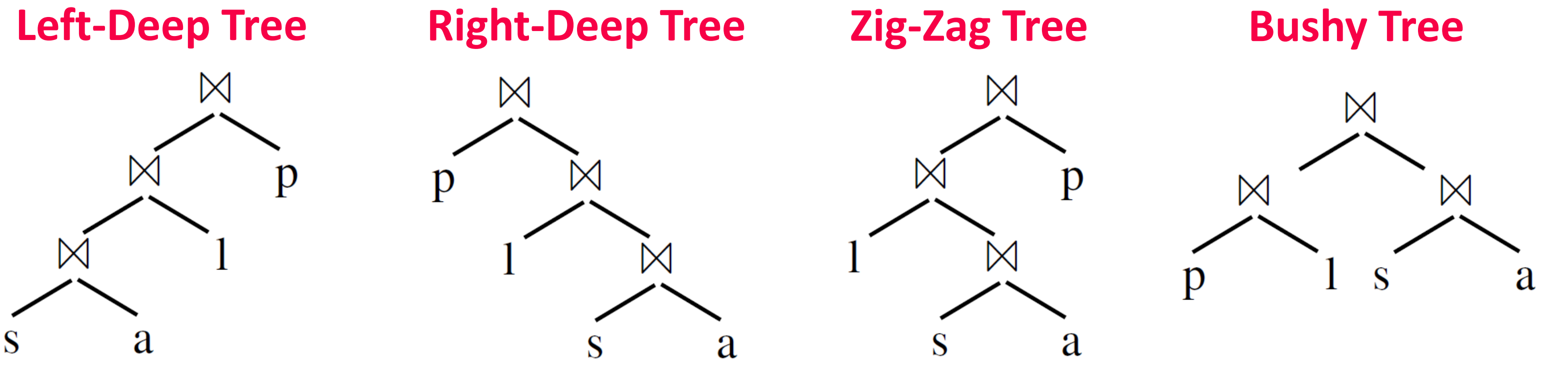
Query and Plan Types

[Guido Moerkotte, Building Query Compilers, **2020**]

- **Query Types**
  - **Nodes:** Tables
  - **Edges:** Join conditions
  - Determine **hardness of query optimization** (w/o cross products)



- **Join Tree Types / Plan Types**
  - Data flow graph of tables and joins (logical/physical query trees)
  - **Edges:** data dependencies (fixed execution order: bottom-up)



Join Ordering Problem

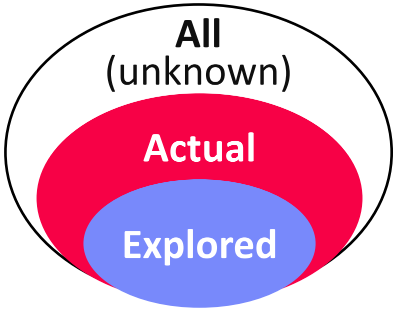
[Guido Moerkotte, Building Query Compilers, **2020**]

- **Join Ordering**
  - Given a join query graph, find the optimal join ordering
  - In general, **NP-hard**; but polynomial algorithms exist for special cases
- **Search Space**
  - Dependent on query and plan types
  - **Note:** if we allow cross products similar to cliques (fully connected)

	Chain (no CP)			**Star (no CP)**		**Clique / CP (cross product)**		
	left-deep	**zig-zag**	**bushy**	**left-deep**	**zig-zag/ bushy**	**left-deep**	**zig-zag**	**bushy**
n	2^{n-1}	2^{2n-3}	$2^{n-1}C(n-1)$	$2(n-1)!$	$2^{n-1}(n-1)!$	$n!$	$2^{n-2}n!$	$n!\,C(n-1)$
5	16	128	224	48	384	120	960	1,680
10	512	~131K	~2.4M	~726K	~186M	~3.6M	~929M	~17.6G

C(n) … Catalan Numbers

Join Order Search Strategies

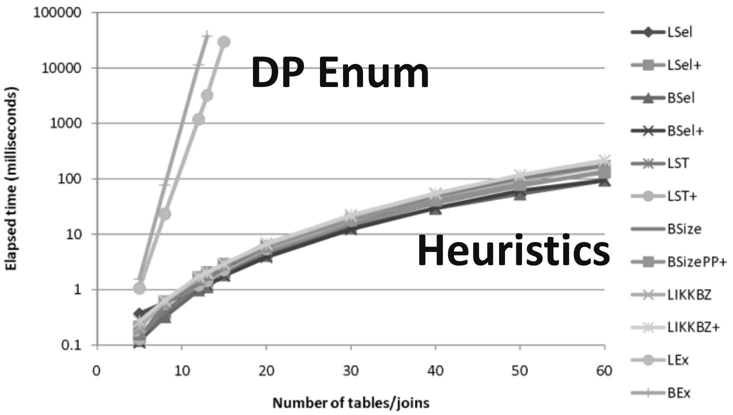


All (unknown)

Actual

Explored

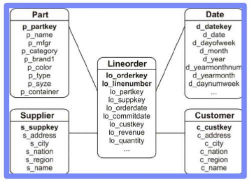
- **Tradeoff: Optimal (or good)** plan vs **compilation time**
- **#1 Naïve Full Enumeration**
  - Infeasible for reasonably large queries (long tail up to 1000s of joins)
- **#2 Exact Dynamic Programming / Memoization**
  - Guarantees optimal plan, often too expensive (beyond 20 relations)
  - Bottom-up vs top-down approaches
- **#3 Greedy / Heuristic Algorithms**
- **#4 Approximate Algorithms**
  - E.g., Genetic algorithms, simulated annealing, mixed-integer linear prog.
- **Example PostgreSQL**
  - Exact optimization (DPSize) if < 12 relations (geqo_threshold)
  - Genetic algorithm for larger queries
  - Join methods: NLJ, SMJ, HJ



[Nicolas Bruno, César A. Galindo-Legaria, Milind Joshi: Polynomial heuristics for query optimization. **ICDE 2010**]

Greedy Join Ordering

Star Schema Benchmark



- **Example**
  - Part ⋈ Lineorder ⋈ Supplier ⋈ σ(Customer) ⋈ σ(Date), **left-deep plans**

#	Plan	Costs
1	Lineorder ⋈ Part	30M
	Lineorder ⋈ Supplier	20M
	Lineorder ⋈ σ(Customer)	90K
	Lineorder ⋈ σ(Date)	**40K**
	~~Part ⋈ Customer~~	N/A
	…	…

2	(Lineorder ⋈ σ(Date)) ⋈ Part	150K
	(Lineorder ⋈ σ(Date)) ⋈ Supplier	100K
	(Lineorder ⋈ σ(Date)) ⋈ σ(Customer)	**75K**

#	Plan	Costs
3	((Lineorder ⋈ σ(Date)) ⋈ σ(Customer)) ⋈ Part	120M
	((Lineorder ⋈ σ(Date)) ⋈ σ(Customer)) ⋈ Supplier	**105M**

4	**(((Lineorder ⋈ σ(Date)) ⋈ σ(Customer)) ⋈ Supplier) ⋈ Part**	**135M**

Note: Simple O(n^2) algorithm for left-deep trees; O(n^3) algorithms for bushy trees existing (e.g., GOO)

Greedy Join Ordering, cont.

[Guido Moerkotte, Building Query Compilers, **2020**]

- **Basic Algorithms**
  - GreedyJO-1: sort by relation weights (e.g., card)
  - GreedyJO-2: greedy selection of next best relation
  - GreedyJO-3: Greedy-JO-2 w/ start from each relation

Previous example as a hybrid w/ O(n^2)

- **GOO Algorithm**

```
GOO($\{R_1, \ldots, R_n\}$)   // Greedy Operator Ordering
Input:  a set of relations to be joined
Output:  join tree
Trees := $\{R_1, \ldots, R_n\}$
while (|Trees| != 1) {
   find $T_i, T_j$ ∈ Trees such that $i \neq j$, $|T_i \bowtie T_j|$ is minimal
         among all pairs of trees in Trees
   Trees −= $T_i$;
   Trees −= $T_j$;
   Trees += $T_i \bowtie T_j$;
}
return the tree contained in Trees;
```

[Leonidas Fegaras: A New Heuristic for Optimizing Large Queries. **DEXA 1998**]

Dynamic Programming Join Ordering

- **Exact Enumeration via Dynamic Programming**
  - **#1: Optimal substructure** (Bellman's Principle of Optimality)
  - **#2: Overlapping subproblems** allow for memorization

- **Bottom-Up** (Dynamic Programming)
  - Split in independent sub-problems (optimal plan per set of quantifiers and interesting properties), solve sub-problems, combine solutions
  - **Algorithms:** DPsize, DPsub, DPcpp

- **Top-Down** (Memoization)
  - Recursive generation of join trees w/ memorization and pruning
  - **Algorithms:** Cascades, MinCutLazy, MinCutAGat, MinCutBranch

[Guido Moerkotte, Thomas Neumann: Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products. **VLDB 2006**]

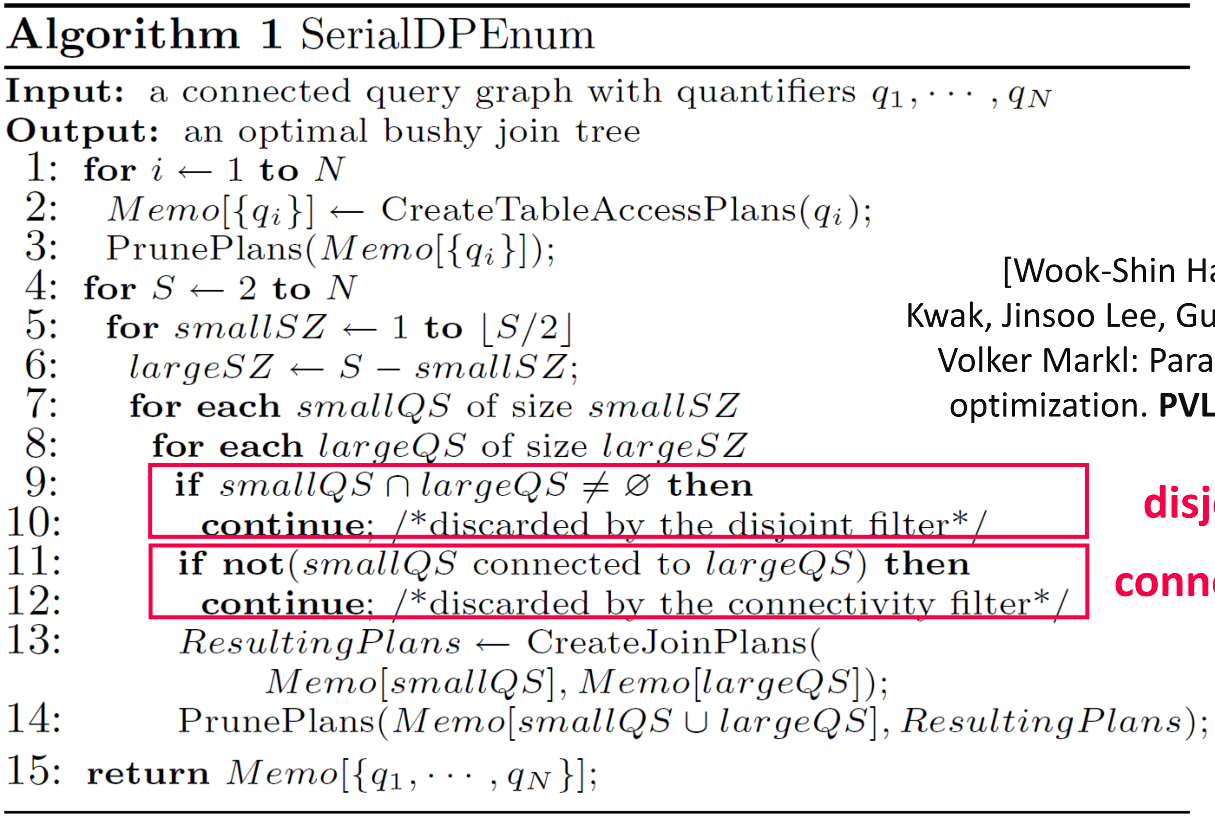
[Goetz Graefe: The Cascades Framework for Query Optimization. **IEEE Data Eng. Bull. 18(3) 1995**]

[Pit Fender: Algorithms for Efficient Top-Down Join Enumeration. **PhD Thesis, University of Mannheim 2014**]

Dynamic Programming Join Ordering, cont.

- **DPSize Algorithm**
  - Pioneered by Pat Selinger et al.
  - Implemented in IBM DB2, Postgres, etc

[Patricia G. Selinger et al.: Access Path Selection in a Relational Database Management System. **SIGMOD 1979**]



```
Algorithm 1 SerialDPEnum
Input: a connected query graph with quantifiers $q_1, \cdots, q_N$
Output: an optimal bushy join tree
for $i \leftarrow 1$ to $N$
  $Memo[\{q_i\}] \leftarrow$ CreateTableAccessPlans($q_i$);
  PrunePlans($Memo[\{q_i\}]$);
for $S \leftarrow 2$ to $N$
  for $smallSZ \leftarrow 1$ to $\lfloor S/2 \rfloor$
    $largeSZ \leftarrow S - smallSZ$;
    for each $smallQS$ of size $smallSZ$
      for each $largeQS$ of size $largeSZ$
        if $smallQS \cap largeQS \neq \emptyset$ then
          continue; /*discarded by the disjoint filter*/
        if not($smallQS$ connected to $largeQS$) then
          continue; /*discarded by the connectivity filter*/
        $ResultingPlans \leftarrow$ CreateJoinPlans(
            $Memo[smallQS], Memo[largeQS]$);
        PrunePlans($Memo[smallQS \cup largeQS], ResultingPlans$);
return $Memo[\{q_1, \cdots, q_N\}]$;
```

disjoint

connected

[Wook-Shin Han, Wooseong Kwak, Jinsoo Lee, Guy M. Lohman, Volker Markl: Parallelizing query optimization. **PVLDB 1(1) 2008**]

Dynamic Programming Join Ordering, cont.

- **DPSize Example**
  - Simplified: no interesting properties

Q1	Plan
{C}	Tbl, ~~IX~~
{D}	~~Tbl~~, IX
{L}	…
{P}	…
{S}	…

Q1+Q1

Q2	Plan
{C,L}	L⋈C, ~~C⋈L~~
{D,L}	L⋈D, ~~D⋈L~~
{L,P}	~~L⋈P~~, P⋈L
{L,S}	~~L⋈S~~, S⋈L
~~{C,D}~~	~~N/A~~
…	…

Q1+Q2, Q2+Q1

Q3	Plan
{C,D,L}	(L⋈C)⋈D, ~~D⋈(L⋈C)~~, ~~(L⋈D)⋈C~~, ~~C⋈(L⋈D)~~
{C,L,P}	~~(L⋈C)⋈P~~, P⋈(L⋈C), ~~(P⋈L)⋈C~~, ~~C⋈(P⋈L)~~
{C,L,S}	…
{D,L,P}	…
{D,L,S}	…
{L,P,S}	…

Q1+Q3, Q2+Q2, Q3+Q1

Q4	Plan
{C,D,L,P}	~~((L⋈C)⋈D)⋈P~~, P⋈((L⋈C)⋈D)
{C,D,L,S}	…
{C,L,P,S}	…
{D,L,P,S}	…

Q1+Q4, Q2+Q3, Q3+Q2, Q4+Q1

Q5	Plan
{C,D,L,P,S}	…

Graceful Degradation

- **Problem Bottom-Up**
  - Until end of optimization no valid full QEP created (**no anytime algorithm**)
  - **Fallback:** resort to heuristic if ran out of memory / time budget

- **#1 Query Simplification**
  - Simplify query with heuristics until solvable via dynamic programming
  - Choose plans to avoid (restrictions), not to join

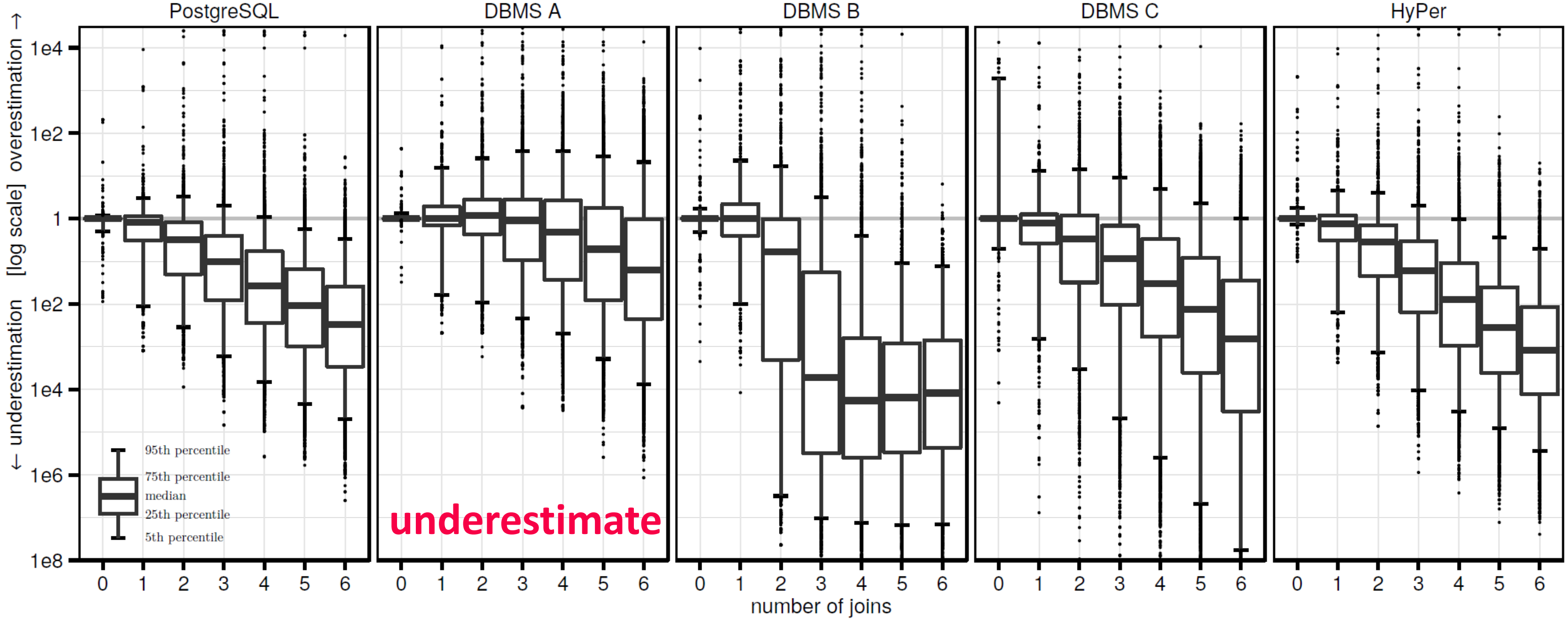
[Thomas Neumann: Query simplification: graceful degradation for join-order optimization. **SIGMOD 2009**]

- **#2 Search Space Linearization**
  - **Small queries:** count connected subgraphs, optimized exactly **DP**
  - **Medium queries** (<100): restrict **$O(n^3)$** algorithm to consider connected sub-chains of linear relation ordering
  - **Large queries:** greedy algorithm, then **Medium** on sub-trees of size K

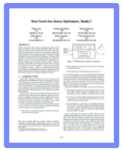
[Thomas Neumann, Bernhard Radke: Adaptive Optimization of Very Large Join Queries. **SIGMOD 2018**]

Join Order Benchmark (JOB)

- **Data:** Internet Movie Data Bases (IMDB)
- **Workload:** 33 query templates, 2-6 variants / 3-16 joins per query



underestimate



[Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, Thomas Neumann: How Good Are Query Optimizers, Really? **PVLDB 9(3) 2015**]

AQP Fundamentals

Motivation

[Zachary G. Ives, Amol Deshpande, Vijayshankar Raman: Adaptive query processing: Why, How, When, and What Next? **VLDB 2007**, http://www.cs.umd.edu/~amol/talks/VLDB07-AQP-Tutorial.pdf]

- **Recap: Success of SQL/Relational Model**
  - **Declarative:** what not how
  - **Flexibility:** closure property → composition
  - **Automatic optimization**
  - **Physical data independence**

[Guy Lohman: Query Optimization: Are We There Yet?, **BTW 2017**]

Query Optimization:
Are We There Yet?

Guy Lohman
IBM Almaden Research Center (retired)
Guy_Lohman@alumni.pomona.edu

- **Problems**
  - **Unknown statistics** (e.g., unreliable cost model assumptions uniformity, independence)
  - **Changing data / environment characteristics** (e.g., data integration, streams)

$$\frac{\Delta app}{\Delta t} \ll \frac{\Delta env}{\Delta t}$$

- **Adaptivity**
  - Query optimization/processing adapts implementation to runtime conditions

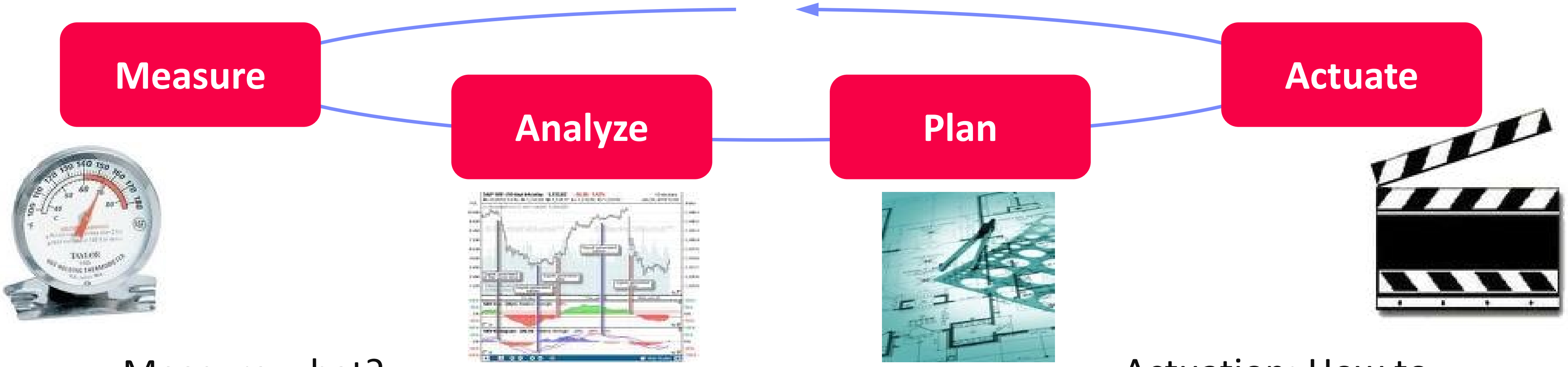
→ **Adaptive query processing** for fine-grained adaptivity

AQP Control Loop

[IBM: An architectural blueprint for autonomic computing, Technical Report, **2005**]

[Zachary G. Ives, Amol Deshpande, Vijayshankar Raman: Adaptive query processing: Why, How, When, and What Next? **VLDB 2007**]

- **MAPE** (Measure/Monitor, Analyze, Plan, Execute/Actuate)



Measure what?
Measure when?
Measurement overhead?

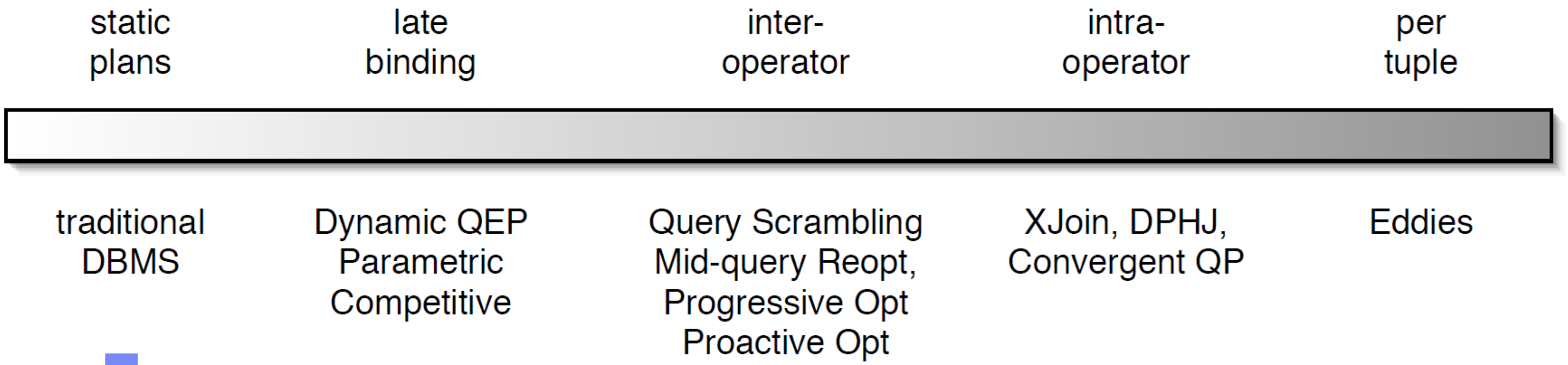
Analyze / plan what?
Analyze / plan when?
Plan how far ahead?
Planning overhead?

Actuation: How to switch plans / routing?
Actuation overhead?

Classification of AQP Techniques

- **Spectrum of Adaptivity**
  - Temporal classification

[Amol Deshpande, Joseph M. Hellerstein, Vijayshankar Raman: Adaptive query processing: why, how, when, what next. **SIGMOD 2006**]



static plans — late binding — inter-operator — intra-operator — per tuple

traditional DBMS — Dynamic QEP Parametric Competitive — Query Scrambling Mid-query Reopt, Progressive Opt Proactive Opt — XJoin, DPHJ, Convergent QP — Eddies

- **#1 Inter-Query Optimization**
  - As established in System R
  - Update statistics (ANALZE, RUNSTATS) (cardinalities, histograms, index low/high keys)
  - Rewrites, join ordering, pruning, etc

[Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, Thomas G. Price: Access Path Selection in a Relational Database Management System. **SIGMOD 1979**]

Recap: ANALYZE and EXPLAIN

- **Step 1:**
```
EXPLAIN SELECT * FROM Participant AS R, Locale AS S
        WHERE R.LID=S.LID;
```

```
Hash Join (.. rows=70 width=1592)
  Hash Cond:(s.lid = r.lid)
  ->  Seq Scan on locale s (.. rows=140 width=520)
  ->  Hash (.. rows=70 width=1072)
        ->  Seq Scan on participant r (.. rows=70 width=1072)
```

build side

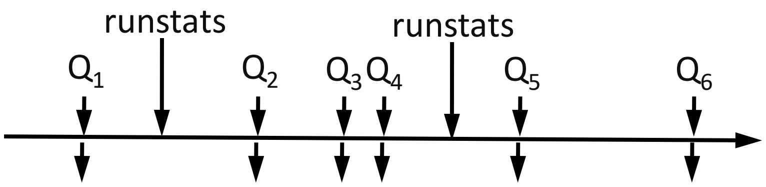
- **Step 2:**
```
ANALYZE Participant, Locale;
```

- **Step 3:**
```
EXPLAIN SELECT * FROM Participant AS R, Locale AS S
        WHERE R.LID=S.LID;
```

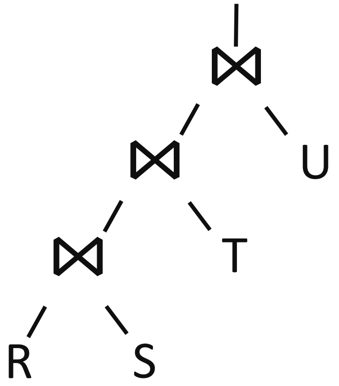
```
Hash Join (.. rows=17 width=47)
  Hash Cond:(r.lid = s.lid)
  ->  Seq Scan on participant r (.. rows=17 width=30)
  ->  Hash (.. rows=11 width=17)
        ->  Seq Scan on locale s (.. rows=11 width=17)
```

WHY?

#1 Inter-Query Optimization



```
SELECT *
  FROM R, S, T, U
  WHERE R.a=S.a
    AND R.a=T.a
    AND R.a=U.a
```

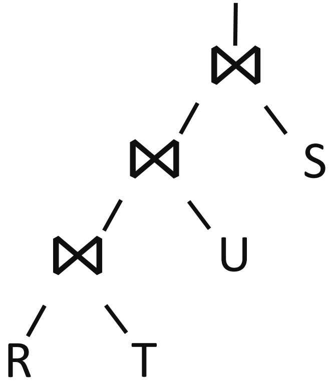


Update Statistics

```
RUNSTATS ON TABLE R
RUNSTATS ON TABLE S
RUNSTATS ON TABLE T
RUNSTATS ON TABLE U
```

Plan cache invalidation and full optimization after statistics update

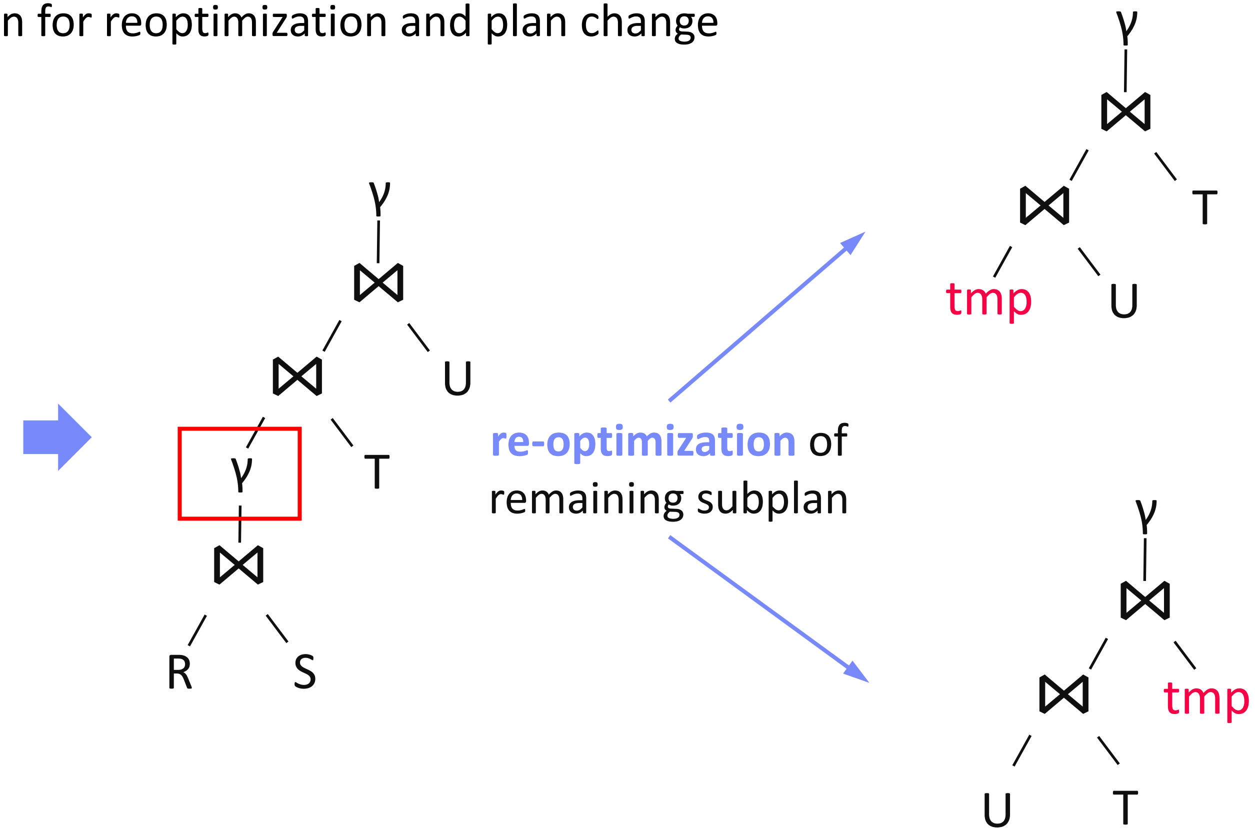
```
SELECT *
  FROM R, S, T, U
  WHERE R.a=S.a
    AND R.a=T.a
    AND R.a=U.a
```



#2 Late Binding (Staged Execution)

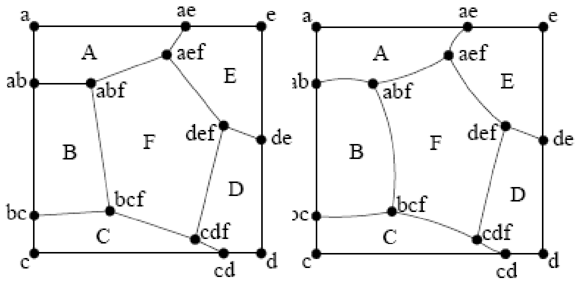
- **Basic Idea**
  - Use **natural blocking/materialization points** (sort, group-by) within a plan for reoptimization and plan change

```
SELECT *
  FROM R, S, T, U
  WHERE R.a=S.a
    AND R.a=T.a
    AND R.a=U.a
  GROUP BY R.a
```



re-optimization of remaining subplan

#2 Late Binding (Parameters)



- **Problem**
  - Unknown predicates at query compile-time (e.g., prepared statements)
  - Similar to unknown or misestimated statistics
  - Re-optimization for each query (Optimize-Always) causes unnecessary overhead
- **Basic Idea: Parametic Query Optimization**
  - Proactively optimize a query into a set of candidate plans
  - Each candidate is optimal for some region of the parameter space
  - Pick appropriate plan during query execution when parameters known
- **Approaches**
  - Progressive PQO (pay-as-you-go)
  - PQO for linear cost functions
  - AniPQO for non-linear cost functions

[Pedro Bizarro, Nicolas Bruno, David J. DeWitt: Progressive Parametric Query Optimization. **IEEE Trans. Knowl. Data Eng. 21(4) 2009**]

Learned Cardinalities

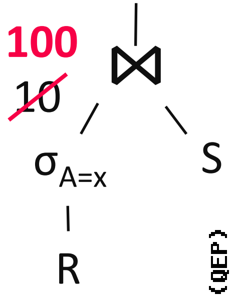
Cardinality Estimation Problems

- **Motivation**
  - Assumptions: **uniformity** of value distributions, and **independence**
  - Multi-dimensional histograms: too expensive, unclear bucket boundaries

- **Sources of Estimation Errors**
  - Correlated attributes (predicates, joins)
  - Redundant predicates

[Guy Lohman: Query Optimization: Are We There Yet?, **BTW 2017**]

- **Plan Sensitivity**
  - Recap: Plan Diagrams
  - **Example**

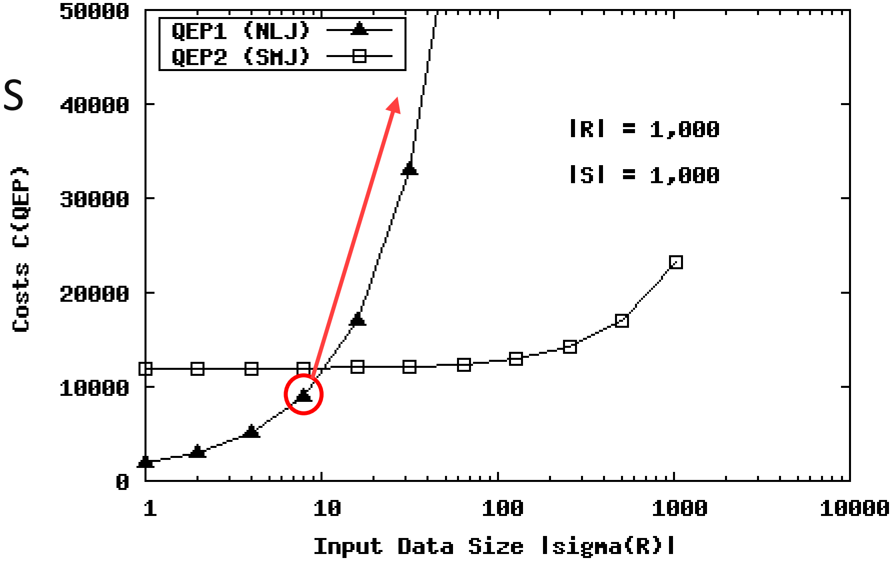


QEP1 (NLJ)

$C(NLJ) = |\sigma(R)| + |\sigma(R)| \cdot |S|$

QEP2 (SMJ)

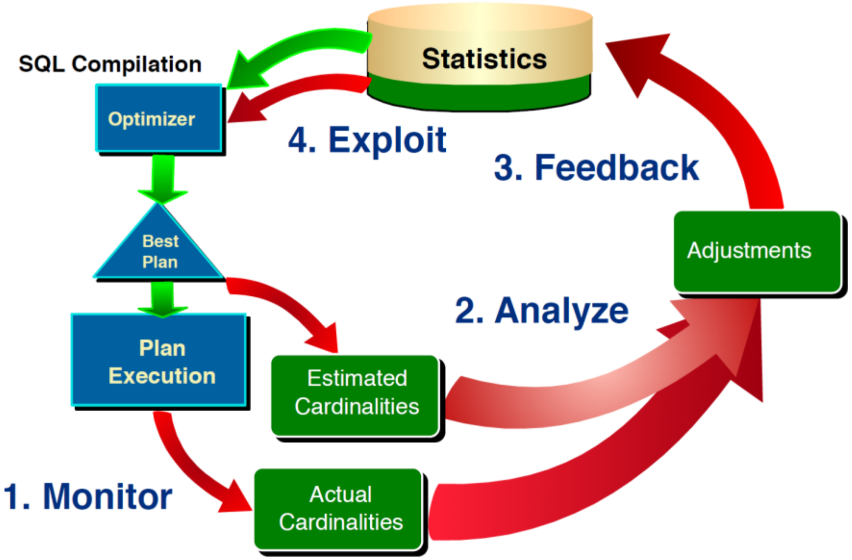
$C(SMJ) = |\sigma(R)| \cdot \log_2|\sigma(R)|$
$\quad + |S| \cdot \log_2|S| + |\sigma(R)| + |S|$



Monitoring Actual Cardinalities

- **LEO: DB2 Learning Optimizer**
  - Monitor and compare estimated and true cardinalites
  - Consistently adjust statistics (e.g., for correlations)

[Michael Stillger, Guy M. Lohman, Volker Markl, Mokhtar Kandil: LEO - DB2's LEarning Optimizer. **VLDB 2001**]



- **ASE: Adaptive Selectivity Estimation**
  - Approximate real attribute value distribution by curve-fitting function on query feedback

[Chung-Min Chen, Nick Roussopoulos: Adaptive Selectivity Estimation Using Query Feedback. **SIGMOD 1994**]

- **SIT: Statistics on Query Expressions**
  - Exploit statistics for intermediates in order to avoid the propagation of errors

[Nicolas Bruno, Surajit Chaudhuri: Exploiting statistics on query expressions for optimization. **SIGMOD 2002**]

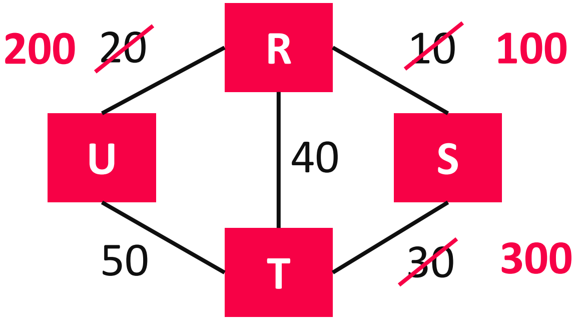
Sparse Monitored Information

- **Danger w/ Actuals**
  - **“Fleeing from Knowledge to Ignorance”**
  - Statistics of correlated attributes usually adjusted to larger cardinalities
  - Plan selection favors small, non-adjusted estimates (independence)

[Guy Lohman: Query Optimization: Are We There Yet?, **BTW 2017**]

Query Optimization: Are We There Yet?

- **Example**
  - Plan 1: (((R ⋈ S) ⋈ U) ⋈ T)
  - Plan 2: ((R ⋈ U) ⋈ (T ⋈ S))
  - Plan 3: (((R ⋈ T) ⋈ U) ⋈ S)



- **Relation to Search Space**
  - Intermediates (not concrete plan) relevant for cardinalities
  - Example: ((R ⋈ S) ⋈ T) and (R ⋈ (S ⋈ T)) produce the same results
  - **Still exponential** (power set)

ML For Cardinality Estimation

- **Common Approach**
  - Featurization of attributes, tables, and joins
  - Concatenation/aggregation of sub-models
  - Augmentation by samples for training and/or corrections

- **Examples**

[Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, Alfons Kemper: **Learned Cardinalities: Estimating Correlated Joins with Deep Learning**. **CIDR 2019**]

[Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, Surajit Chaudhuri: **Selectivity Estimation for Range Predicates using Lightweight Models**. **PVLDB 12(9) 2019**]

[Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Peter Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, Ion Stoica: **Deep Unsupervised Cardinality Estimation**. **PVLDB 13(3) 2019**]

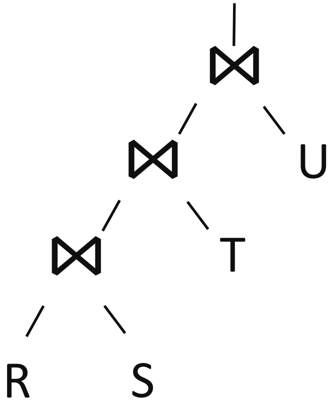
Intra-Query Adaptivity

#3 Inter-Operator Re-optimization

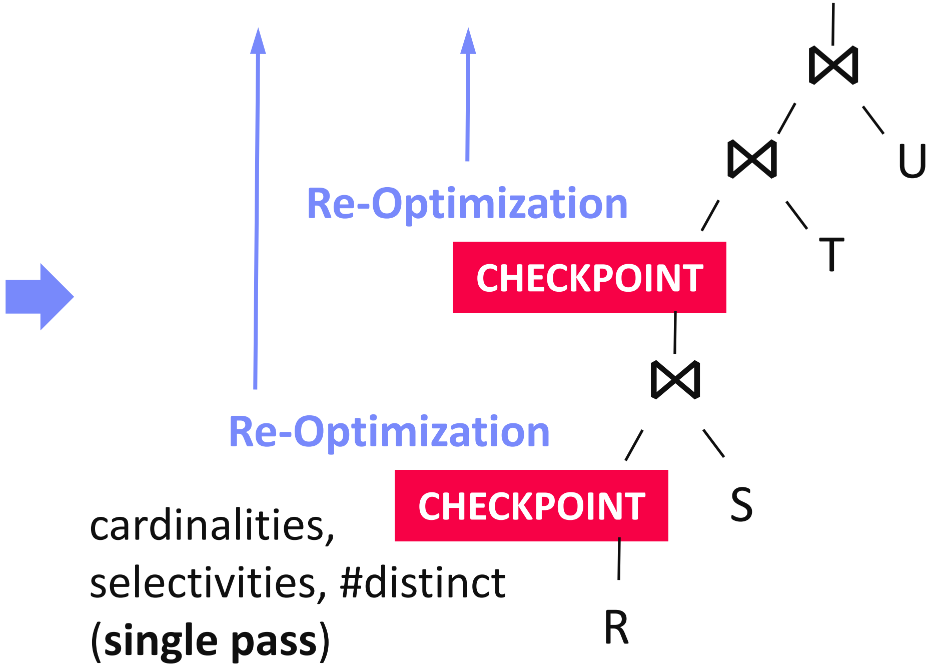
[Navin Kabra, David J. DeWitt: Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. **SIGMOD 1998**]

- **Basic Idea**
  - Insert artificial materialization points for reoptimization at arbitrary points between plan operators

```
SELECT *
  FROM R,S,T,U
  WHERE R.a=S.a
    AND R.a=T.a
    AND R.a=U.a
```



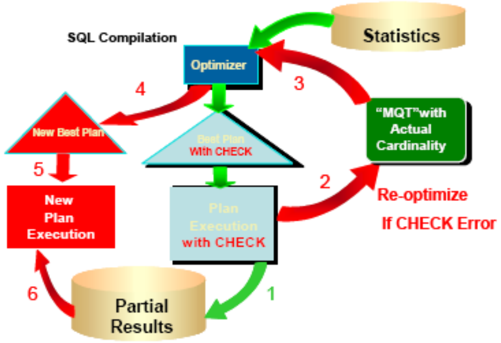
Optimal remaining sub-plans and resource allocation



cardinalities, selectivities, #distinct (**single pass**)

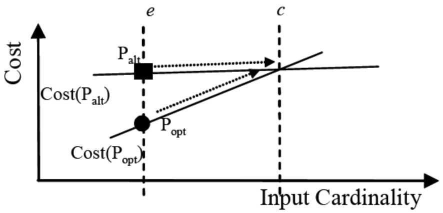
#3 Inter-Operator Re-optimization, cont.

- **Problem**
  - Expensive checkpointing / wasted intermediates
  - Trigger re-optimization only if violated validity ranges



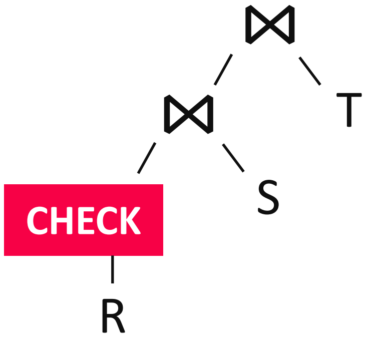
Reactive Reoptimization

1) Compute Validity Range (BLACK BOX)



2) Place Checkpoint operators

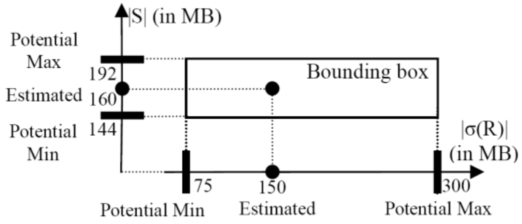
3) Re-optimization on CHECK error



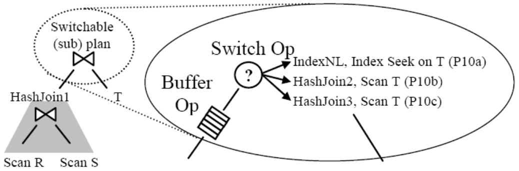
[Volker Markl, V. Raman, D. E. Simmen, G. M. Lohman, H. Pirahesh: Robust Query Processing through Progressive Optimization. **SIGMOD 2004**]

Proactive Reoptimization

1) Bounding box around estimates



2) Use bounding boxes to compute a switchable plan for (lo, est, hi)



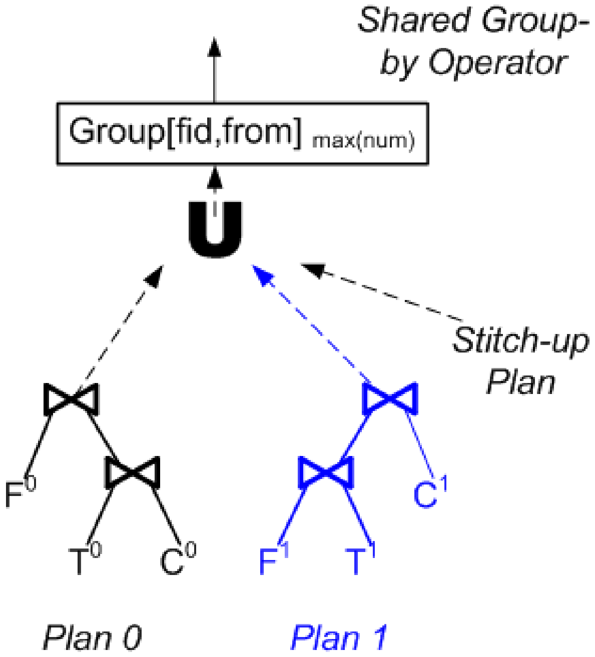
[Shivnath Babu, Pedro Bizarro, David J. DeWitt: Proactive Re-optimization. **SIGMOD 2005**]

#4 Intra-Operator Adaptivity

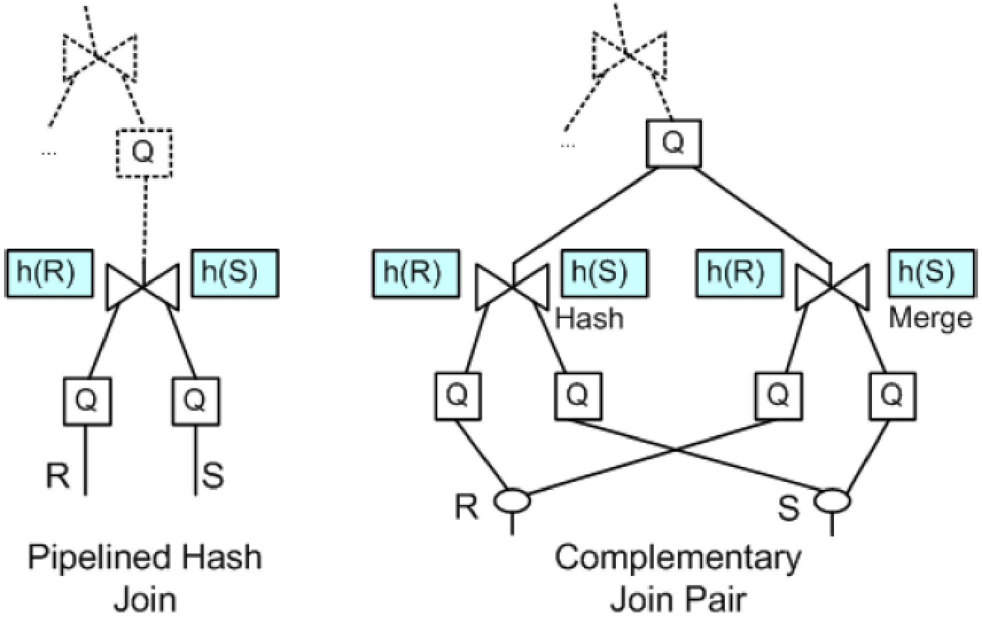
[Zachary G. Ives, Alon Y. Halevy, Daniel S. Weld: Adapting to Source Properties in Processing Data Integration Queries. **SIGMOD 2004**]

- **Basic Idea** (Corrective Query Processing)
  - Use different plans for different partitions of the data
  - Combine the results of subplans in stich-up-phases

(a) aggregation/join query as combined results of two plans

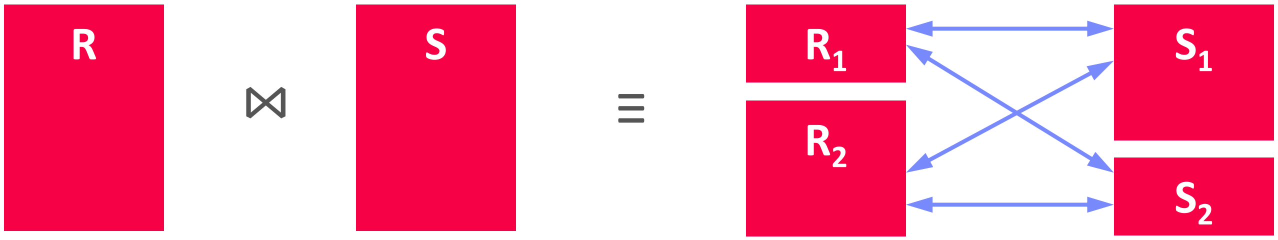


(b) Complementary Join Pair
(Generalized of Pipelined Hash Join)



#4 Intra-Operator Adaptivity, cont.

- **Algebraic Key Property**
  - Distribute relational UNION through PSJ operations
  - Joins $$R_1 \bowtie \ldots \bowtie R_m = \bigcup_{1 \le c_1 \le n, \ldots, 1 \le c_m \le n} (R_1^{c1} \bowtie \ldots \bowtie R_m^{c_m})$$
- **Example**
  - If $R = R_1 \cup R_2$, (horizontal partitioning)
    $S = S_1 \cup S_2$
  - then: $R \bowtie S \equiv (R_1 \cup R_2) \bowtie (S_1 \cup S_2)$
    $\equiv (R_1 \bowtie S_1) \cup (R_1 \bowtie S_2) \cup (R_2 \bowtie S_1) \cup (R_2 \bowtie S_2)$

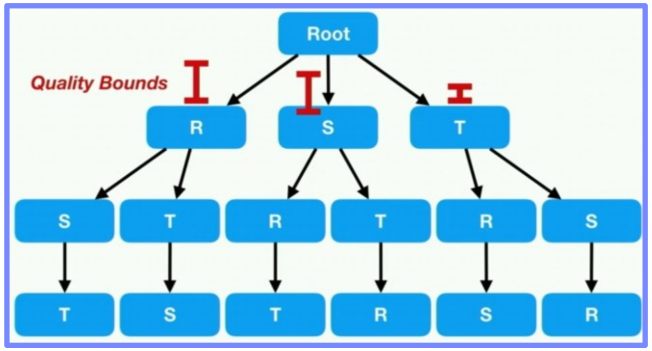


Intra-Query Learning

[Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, Joseph Antonakakis: SkinnerDB: Regret-Bounded Query Evaluation via Reinforcement Learning. **SIGMOD 2019**]

- **Basic Idea**
  - Micro-episodes (**time slices**), run plans with different join orders, evaluate reward, **stitch-up partial results** (no cost model, good for UDFs)
  - Exploitation vs exploration via **reinforcement learning**

- **UCT for Join Ordering**
  - Build tree of join orders gradually from root to bottom
  - Store statistics in nodes of tree
  - Pick next best order via UCT algorithm (w/ guarantees on cumulative regret)



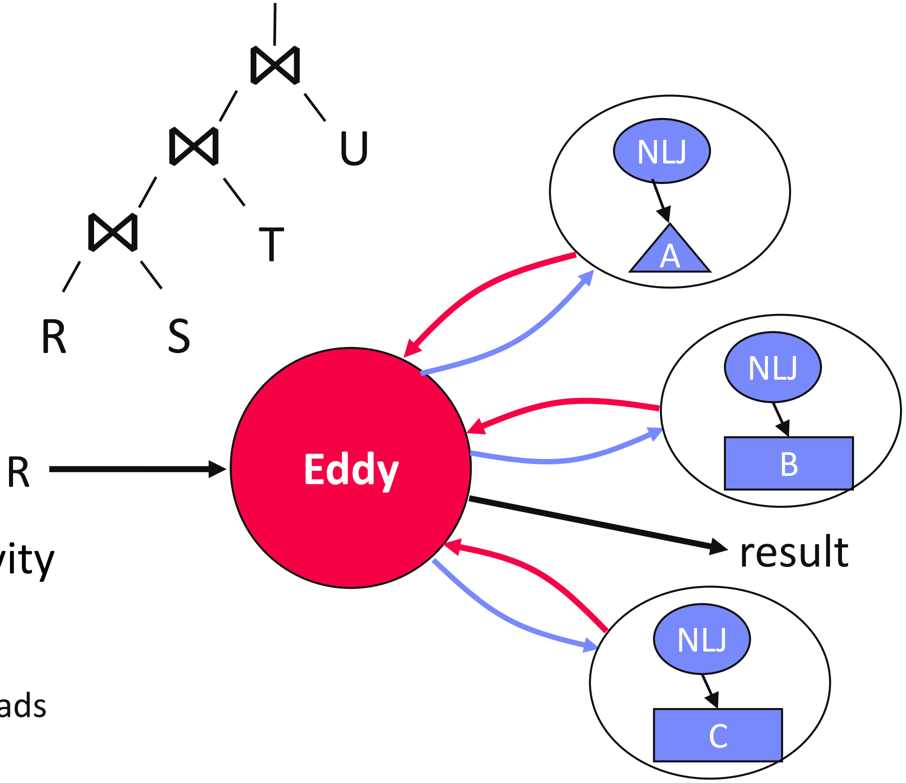
- **Multi-way Joins**
  - Evaluate entire join order in given time slices
  - Reuse previous state (e.g., hash tables)

#5 Tuple Routing (Eddies)

[Ron Avnur, Joseph M. Hellerstein: Eddies: Continuously Adaptive Query Processing, **SIGMOD 2000**]

- **Basic Idea**
  - No plan (no fixed execution order)
  - Tuples routed to relevant operators using routing policies
- **Query Execution via Tuple Routing**
  - Eddy operator routes tuples to applicable operators
  - Read/done bit vectors

a	b	c	…	ready	done
15	10	ABC	…	**111**	**000**



  - Encapsulates all aspects of adaptivity in a “standard” dataflow operator

[Amol Deshpande: An initial study of overheads of eddies. **SIGMOD Rec. 33(1) 2004**]

#5 Tuple Routing (Eddies) – Routing Policies

- **Deterministic**
  - Monitor **costs & selectivities** continuously
  - Re-optimize periodically using rank ordering (or A-Greedy for correlated predicates)

[Remzi H. Arpaci-Dusseau: Run-time adaptation in River. ACM Trans. Comput. Syst. **TOCS 2003**]

- **Lottery Scheduling**
  - Operators run in threads with input queue
  - Tickets assigned according to input/output
  - Route tuple to next eligible operator with room in queue, **based on #tickets and backpressure**

[Ron Avnur, Joseph M. Hellerstein: Eddies: Continuously Adaptive Query Processing, **SIGMOD 2000**]

- **Content-based Routing**
  - **Different routes for different data**
  - Based on attribute values (i.e., correlation)

[Pedro Bizarro, Shivnath Babu, David J. DeWitt, Jennifer Widom: Content-Based Routing: Different Plans for Different Data. **VLDB 2005**]

Summary and Q&A

- Recap: **Join Enumeration / Ordering**
- **AQP Fundamentals**
- **Learned Cardinalities**
- **Intra-Query Adaptivity**

[Surajit Chaudhuri: Query optimizers: time to rethink the contract? **SIGMOD 2009**] (constraints, directives, anytime algorithms, adaptivity, new environments)

[Marianne Winslett: Pat Selinger Speaks Out. **SIGMOD Rec. 32(4) 2003** https://sigmod.org/publications/interview/pat-selinger/]

"Query optimizers have been 25 years in development, with enhancements of the cost-based query model and the optimization that goes with it, and a richer and richer variety of execution techniques that the optimizer chooses from. We just have to keep working on this. It's a never-ending quest for an increasingly better model and repertoire of optimization and execution techniques. So the more the model can predict what's really happening in the data and how the data is really organized, the closer and closer we will come [to the ideal system]"

- **Next Lectures** (Part C)
  - **10 Cloud Database Systems** [Dec 07, 1pm]
  - ~~**11 Modern Concurrency Control**~~
  - **12 Modern Storage and HW Accelerators** [Dec 07, 3pm]