8 Adaptive Query Processing

Data independence facilitates modern DBMS technology

- Separates specification ("what") from implementation ("how")
- Optimizer maps declarative query → algebraic operations
- Platforms, conditions are constantly changing:
  $\Delta app / \Delta t \ll \Delta env / \Delta t$

- Query processing **adapts** implementation to runtime conditions
  - Static applications → dynamic environments

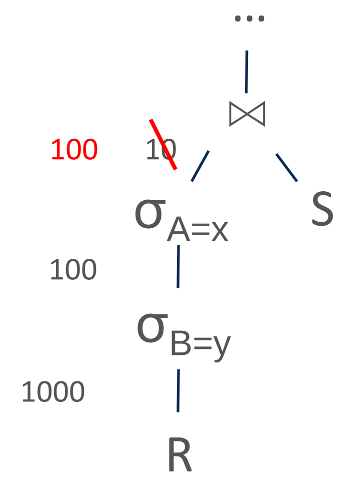
Traditional Optimization Is Breaking

- In traditional settings
  - Queries over many tables
  - Unreliability of traditional cost estimation
  - Success & maturity make problems more apparent, critical
- In new environments
  - e.g. data integration, web services, streams, P2P, sensor nets, hosting
  - Unknown and dynamic characteristics for data and runtime
  - Increasingly aggressive sharing of resources and computation
  - Interactivity in query processing

> Motivation: Problem Correlation

Example Query: $\sigma_{A=x \wedge B=y}(R) \bowtie S$

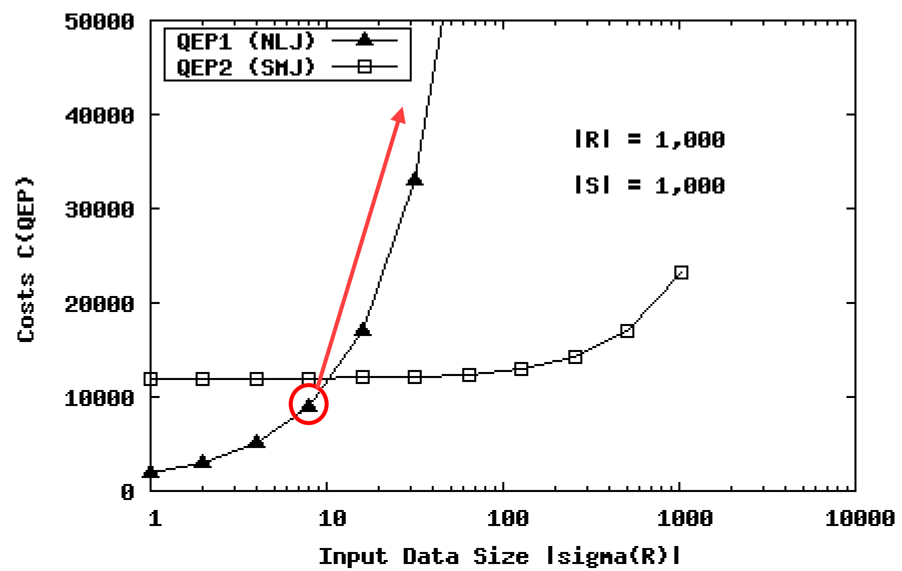
- $f(\sigma_{A=x}(R)) = 1/10$
- $f(\sigma_{B=y}(R)) = 1/10$
- $f(\sigma_{A=x \wedge B=y}(R)) = 1/10 \cdot 1/10 = 1/100$
  (We assumed the predicates
  were independent by multiplying!)
- Problem 1: Correlation (functional dependency)
  ➔ Problem: **underestimation** (leads to aggressive plans, e.g. NLJ)



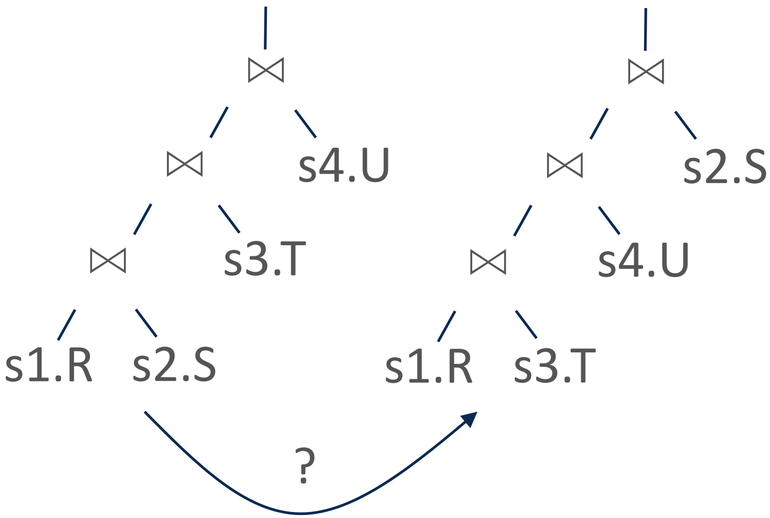
Robustness of Plans
(Insensitivity to input statistics)

- **QEP1 (NLJ)**
  $C(NLJ) = |\sigma(R)| + |\sigma(R)| \cdot |S|$

- **QEP2 (SMJ)**
  $C(SMJ) = |\sigma(R)| \cdot \log_2|\sigma(R)| + |S| \cdot \log_2|S| + |\sigma(R)| + |S|$



> Motivation: Information Integration



Information Integration

- Query processing over heterogeneous systems and applications

Problem 1: Changing Workload Characteristics

- Unpredictable workload of external systems
- Infrastructural properties (network traffic, bandwidth)

Problem 2: Missing Knowledge about Statistics of External Systems

- Cardinalities, Selectivities, Ordering, Bandwidth
- No access to statistical information
- Estimation error increases exponentially in the size of the query

➔ Incremental statistic maintenance and re-optimization

All these problems lead to the same conclusion of ***Adaptive Query Processing***

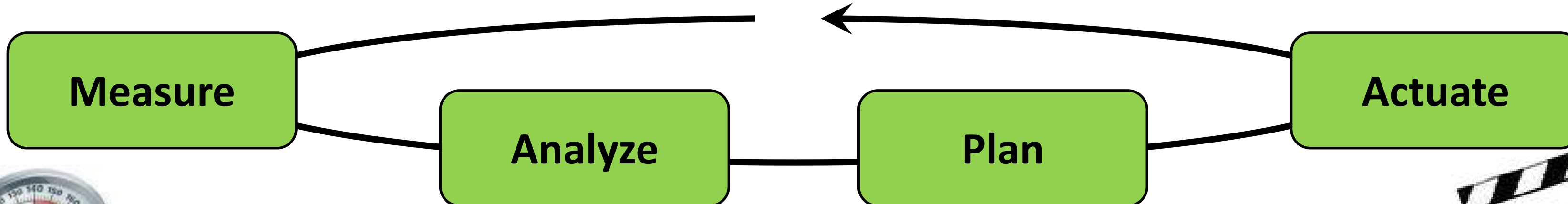
- Changing workload characteristics
- Unknown/uncertain statistics (external systems or correlation)

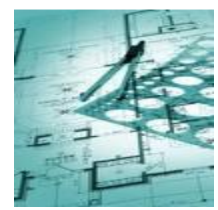
> Generic Feedback Loop

MAPE (Monitor, Analyze, Plan, Execute) /
MAPA (Measure, Analyze, Plan, Actuate)

[IBM. An architectural blueprint for autonomic computing. Technical report, IBM, 2005.]

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





- *Measure what?*
- *Measure when?*
- *Measurement overhead?*

- *Analyze/replan what decisions ?*
- *Analyze / replan when ?*
- *Plan how far ahead ?*
- *Planning overhead ?*

- *Actuation: How do they switch to the new plan/new routing strategy ?*
- *Actuation overhead ?*

> Outline

- Classifications of Adaptive Query Processing
  - Inter-Query Adaptation (static, late binding)
  - Inter-Operator Adaptation
  - Intra-Operator Adaptation
  - Tuple Routing
  - Rewriting-Based Classification
  - Reactive vs. Proactive Reoptimization
  - Synchronous vs. Asynchronous Reoptimization
- Adaptive Query Processing (in Action)
  - Plan-Based Adaptation
  - Continuous-Query-Based Adaptation
  - Routing-Based Adaptation

Classifications of Adaptive Query Processing

> Temporal Classification

Spectrum of Adaptivity

[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

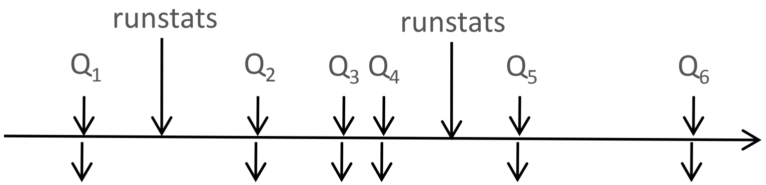
Inter-Query Optimization

- As established in System R

[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].

- UPDATE STATISTICS (cardinalities, histograms, index low/high keys)
- Dynamic Programming + Pruning Techniques

> Temporal Classification (1)

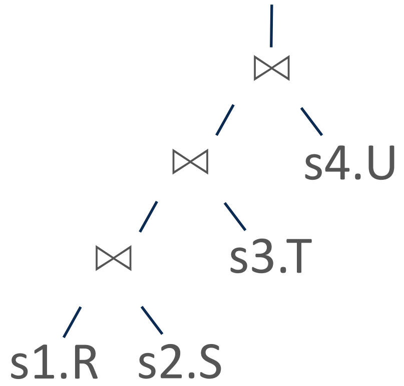


*Example **Inter-Query** Adaptivity*

- Opt for each query
- Opt for batch of queries

Query 1:
SELECT *
FROM R,S,T,U
WHERE R.a=S.a AND
R.a=T.a AND
R.a=U.a

Full Optimization

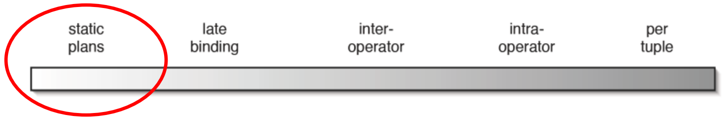


Update Statistics:
RUNSTATS ON TABLE R ON ALL COLUMNS

RUNSTATS ON TABLE S ON ALL COLUMNS

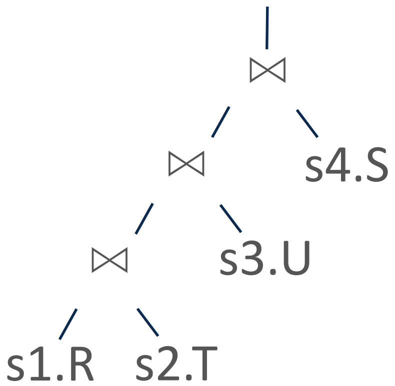
RUNSTATS ON TABLE T ON ALL COLUMNS

RUNSTATS ON TABLE U ON ALL COLUMNS



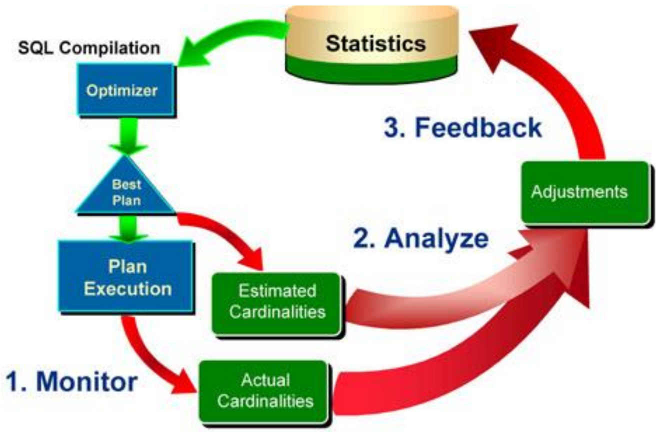
Query 2:
SELECT *
FROM R,S,T,U
WHERE R.a=S.a AND
R.a=T.a AND
R.a=U.a

Full Optimization



> Temporal Classification (1b)

Inter-Query *Adaptivity*



LEO: DB2 Learning Optimizer

- E.g., correlation adjustments with maximum entropy approach

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

ASE: Adaptive Selectivity Estimation

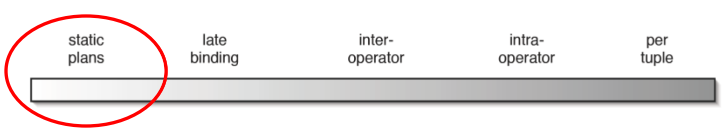
[Chung-Min Chen, Nick Roussopoulos: Adaptive Selectivity Estimation Using Query Feedback. SIGMOD 1994]

- Real attribute value distribution approximated by curve-fitting function using a query feedback mechansism

SIT: Statistics on query expressions

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

- Exploit statistics build on expressions corresponding to intermediate nodes of a query plan rather than only base table statistics
- Problem of many relevant candidate statistics → workload-driven technique

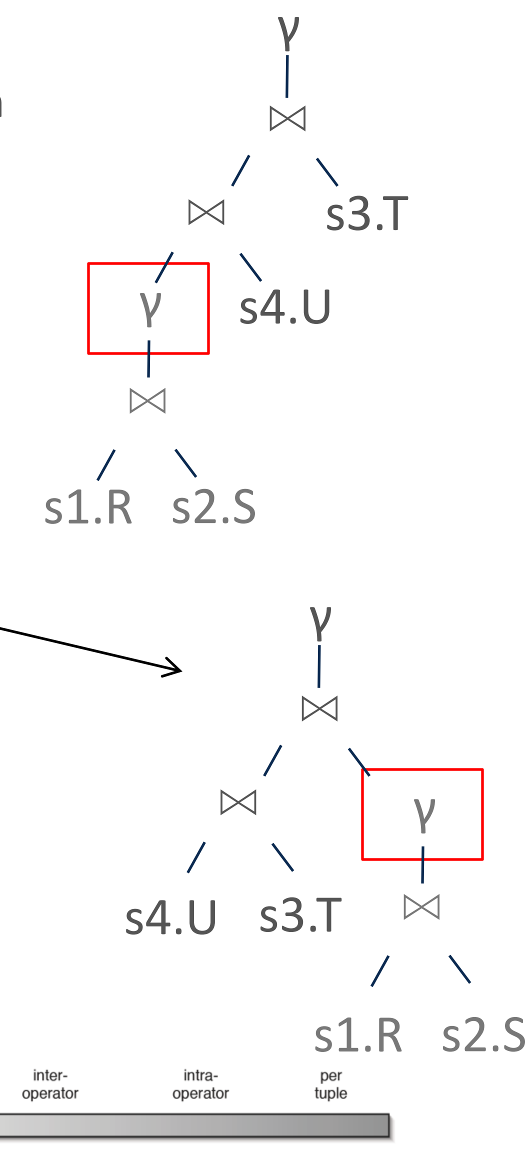


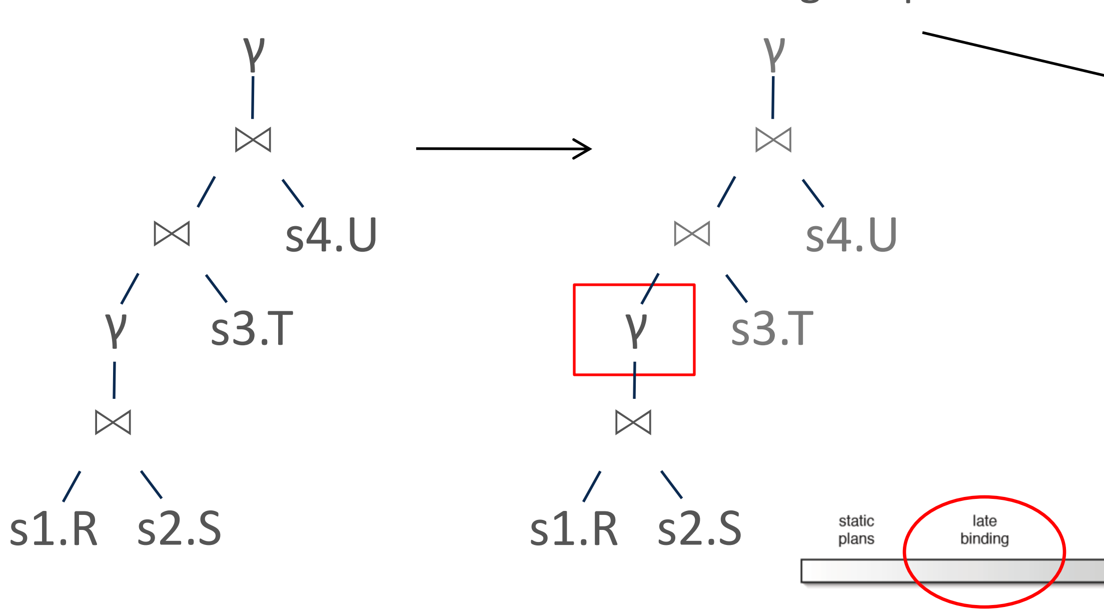
*Example **Late Binding***

- Use natural blocking points within a plan for reoptimization

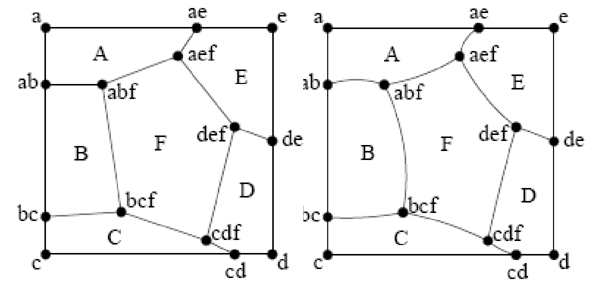
Query:
SELECT *
  FROM R,S,T,U
  WHERE R.a=S.a AND
      R.a=T.a AND
      R.a=U.a
  GROUP BY R.a

Re-Optimization of remaining subplan





> Temporal Classification (2b)



Late Binding *(Parametric Query Optimization)*

Problem

- Unknown predicates at query compile-time (e.g., prepared statements)
- Similar to unknown or misestimated statistics
- Re-optimization for each incoming query may produce redundant work
- *Optimize-Once* and *Optimize-Always* both have disadvantages

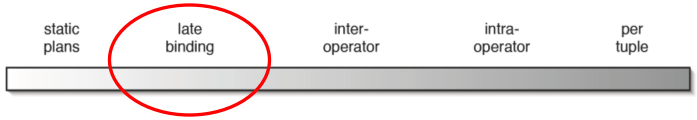
Core Idea

- Optimize a query into a number of candidate plans (initial effort)
- Each candidate is optimal for some region of the parameter space
- Chooses an appropriate plan during query execution when the actual parameter values are known (can be much faster than re-optimization)

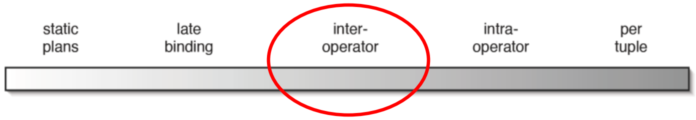
Approaches

- Progressive PQO (pay-as-you-go creation of cadindates)
- 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., TKDE 2009]



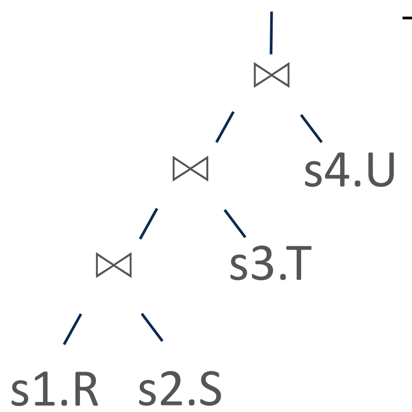
> Temporal Classification (3)

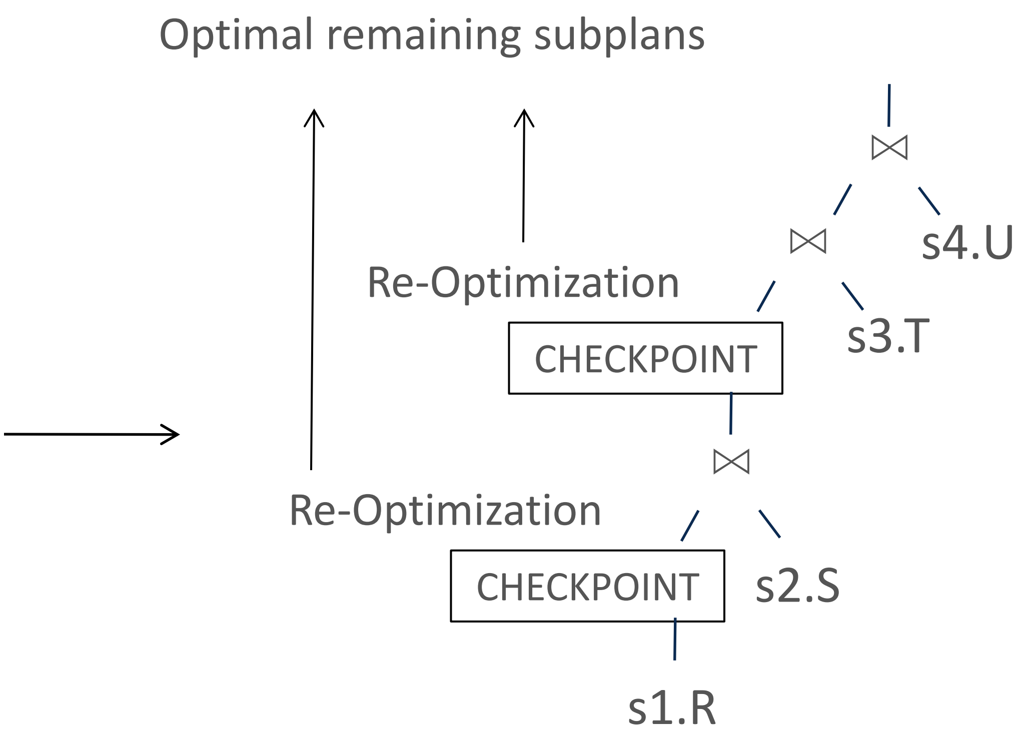


*Example **Inter-Operator***

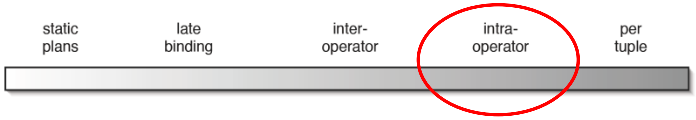
- Insert artificial materialization points for reoptimization at arbitrary points between operators of a plan

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





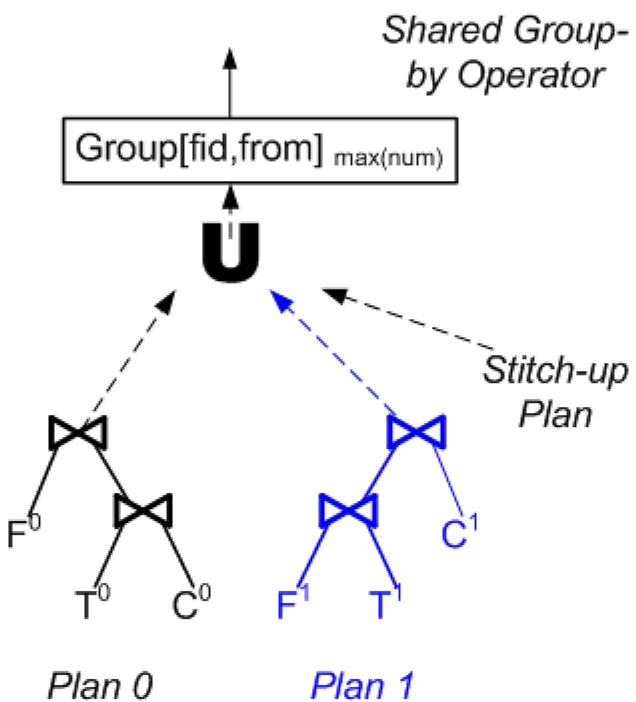
> Temporal Classification (4)



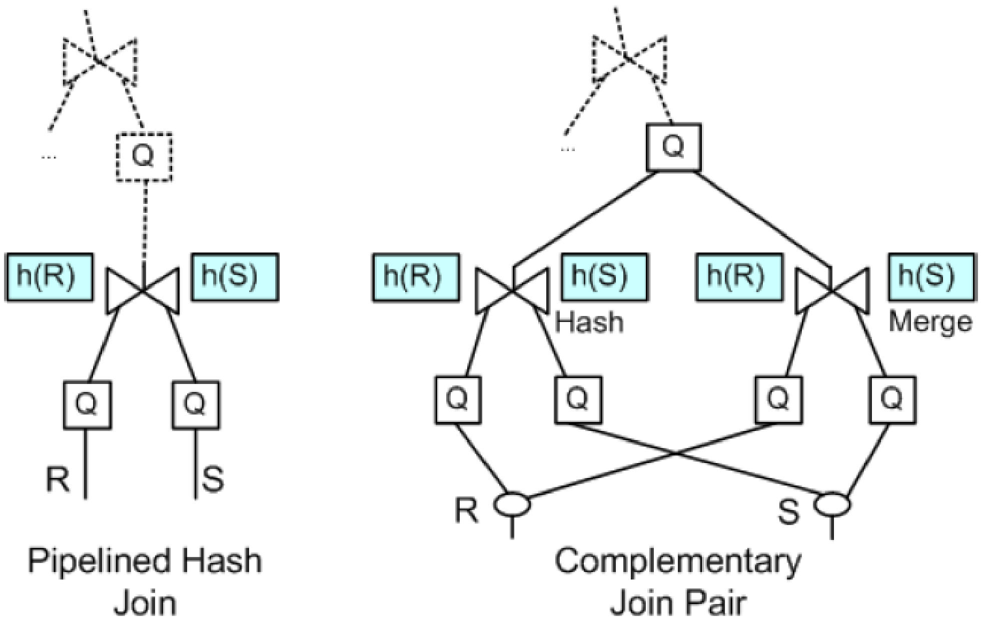
Example ***Intra-Operator***

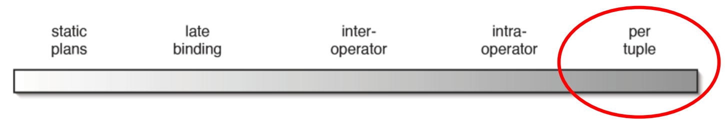
- Use different plans for different partitions of the data
- Combine the results of subplans in stich-up-phases

(a) an aggregation/join query as a combination of two plans plus a final stitch-up-phase



(b) a generalization: „Pipelined Hash Join“ and „Complementary Join Pair“



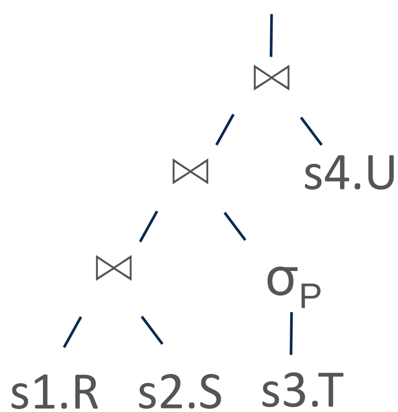


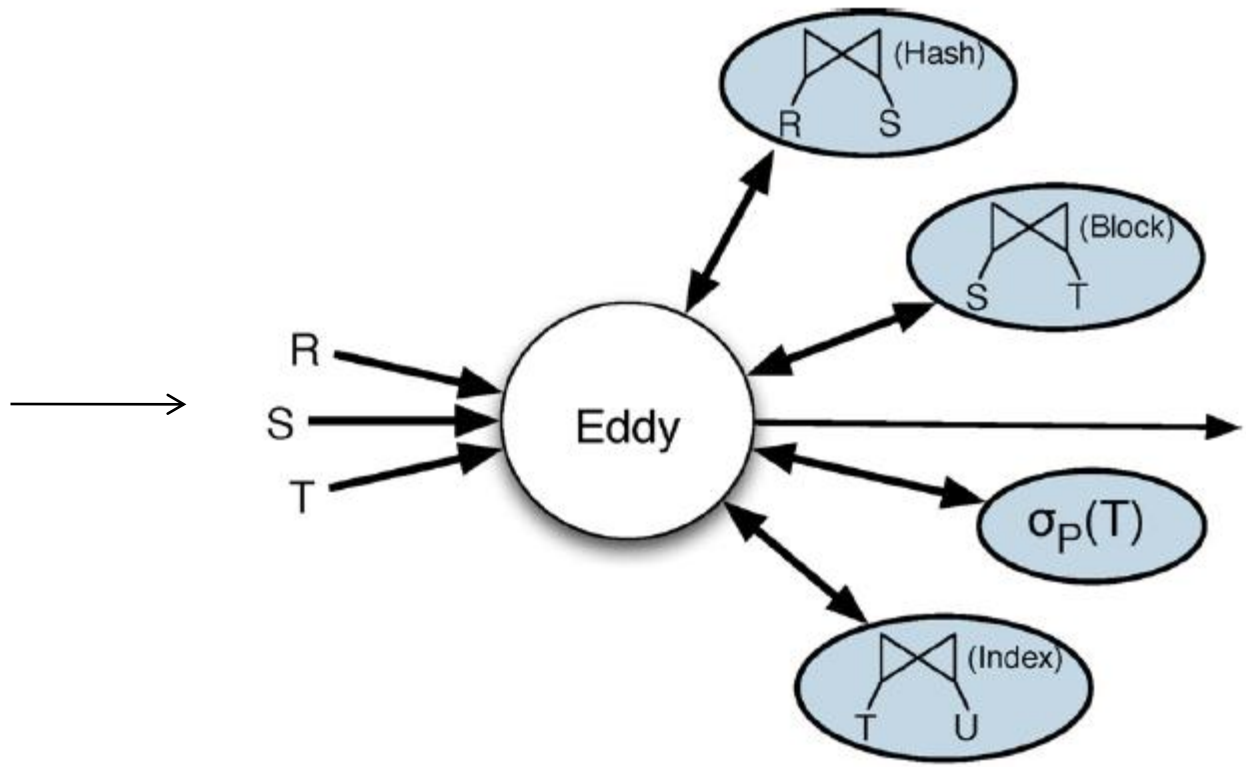
Example ***Tuple Routing***

- No plan (no fixed execution order of operators)
- Tuples are explicitely routed across relevant operators using routing policies

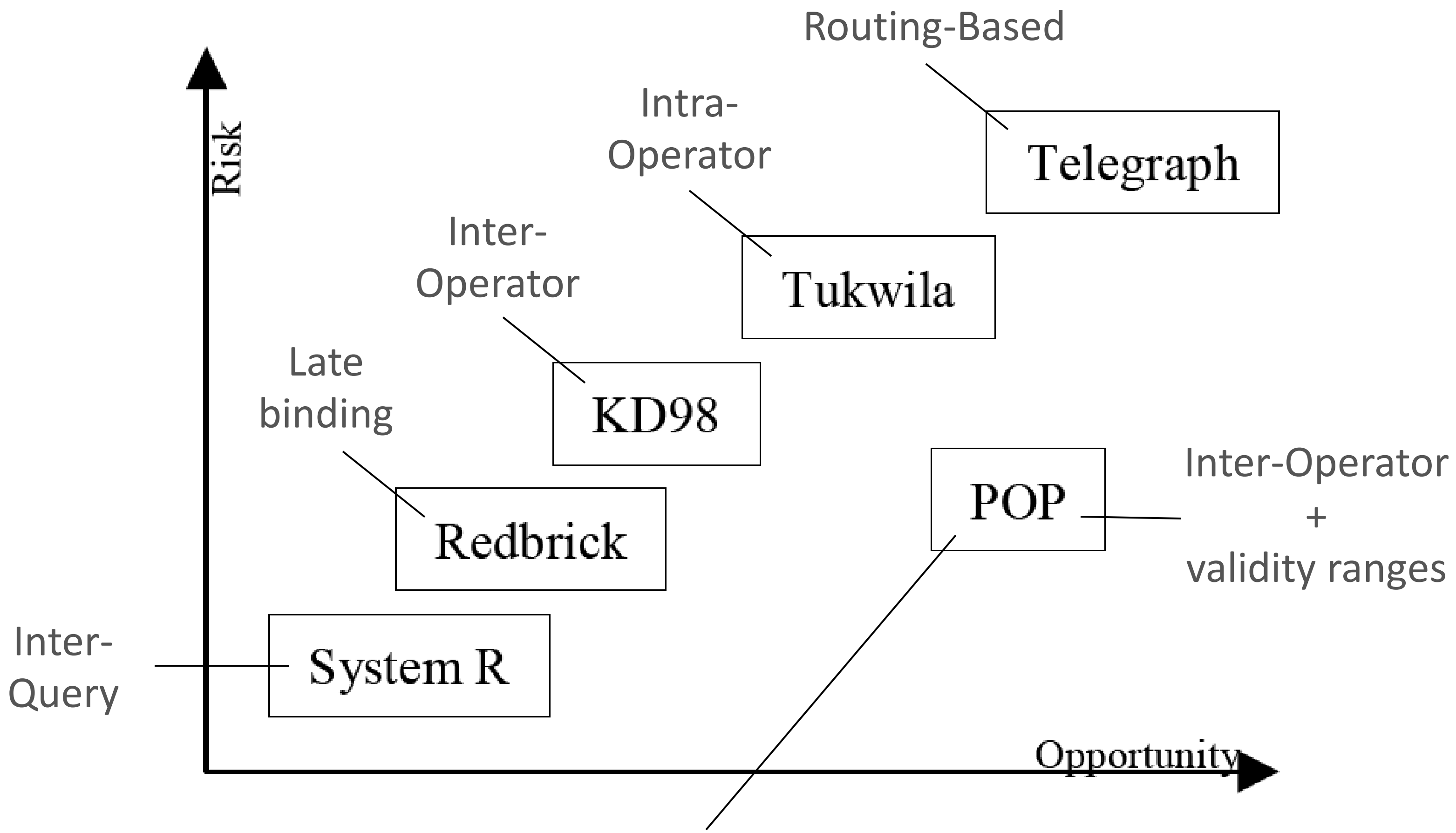
Query:

SELECT *
  FROM R,S,T,U
  WHERE σ_P(T) AND
    R.a=S.a AND
    S.b=T.b AND
    T.c=U.c





Trade-off between Risk and Re-Optimization Opportunity



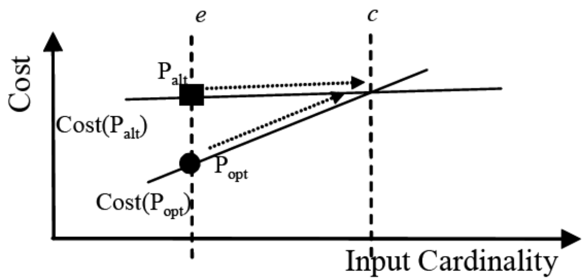
[Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, Hamid Pirahesh: Robust Query Processing through Progressive Optimization. SIGMOD 2004:659-670]

> Reactive vs. Proactive Classification

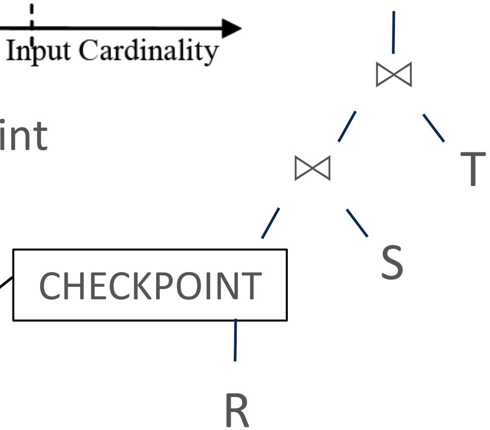
Both use validity ranges to detect suboptimalities but initiated differently

Reactive Reoptimization

1) Compute Validity Range (BLACK BOX)



2) Place Checkpoint operators

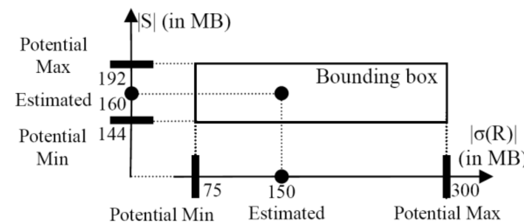


3) Trigger Re-Optimization if validity range is violated at checkpoint operator

Reactive

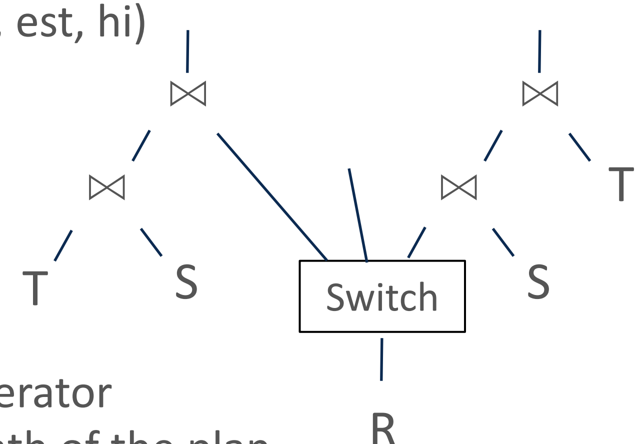
Proactive Reoptimization

1) Bounding box around all estimates



Proactive

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



3) Switch Operator chooses path of the plan

> Rewriting-Oriented Classification

Plan-Based Adaptation (DBMS)

- Reoptimization scope: current plan only (future queries will not benefit)

Continuous-Query-Based Adaptation (DSMS)

- Reoptimization scope: continuous query (future incoming tuples benefit)
- Temporal and state mangement aspects

Routing-Based Adaptation (DBMS, DSMS)

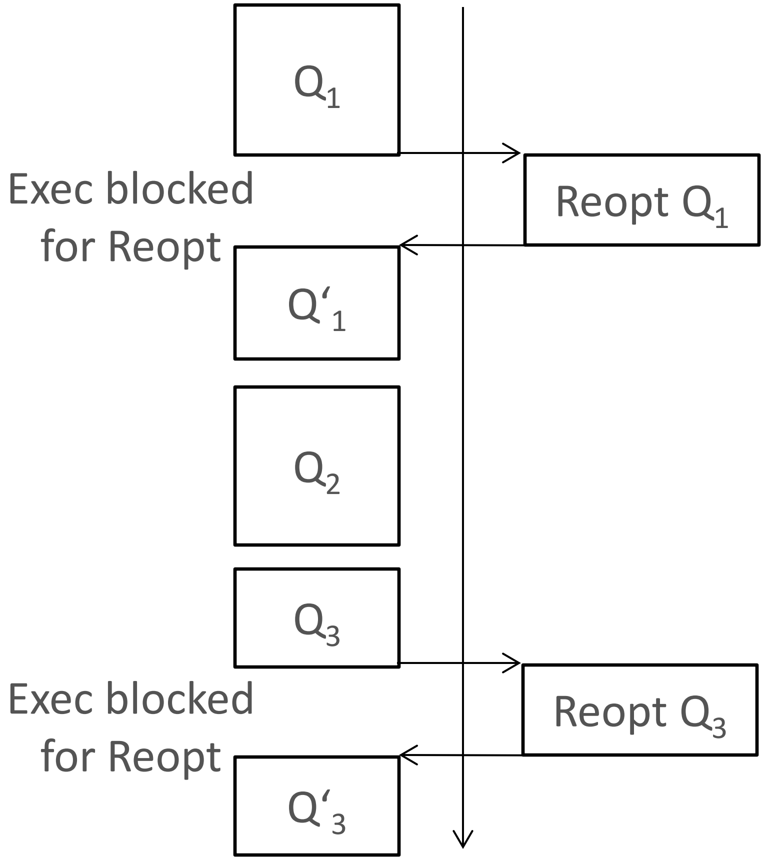
- Query processing and reoptimization as routing of tuples along different paths

Integration-Flow Optimization (IS)

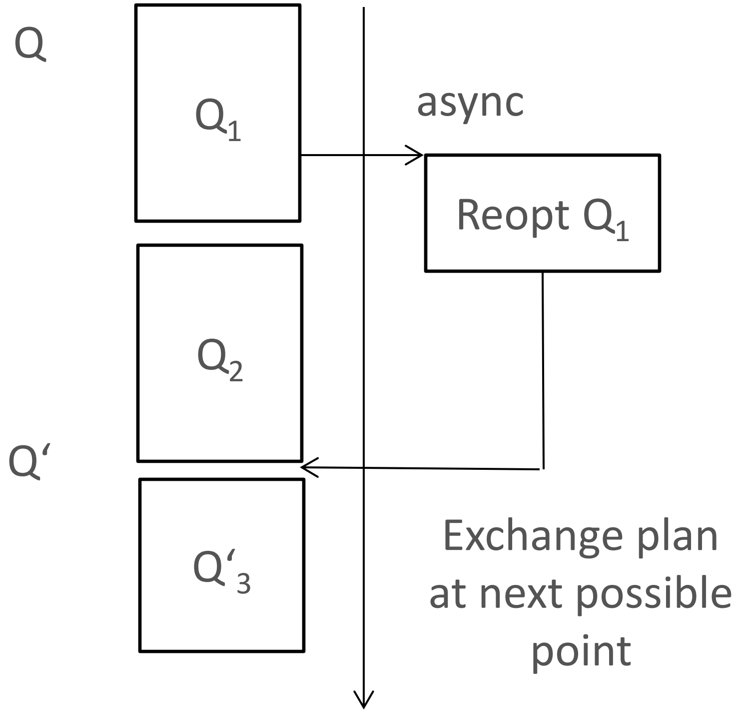
- Many independent instances (deployed once, executed many times)
- Periodical reoptimization (independent of executing a single instance)
- On-demand reoptimization
  - Model optimality conditions of a plan and continuously check if optimality conditions are violated

Synchronous versus Asynchronous Re-Optimization

Synchronous Reopt
(e.g., mid-query optimization)



Asynchronous Reopt
(e.g., reoptimization in DSMS/IS)



Typical granularity: inter-query

> Discussion

Lesson Learned

- Temporal Classification
  (Inter-Query, Late Binding, Inter-Operator, Intra-Operator, Routing -Based)
- Reactive versus Proactive Re-Optimization
- Rewriting-Oriented Classification
- Synchronous versus Asynchronous Re-Optimization

Advantages

- Reoptimization Opportunity: Adaptation to misestimated or unknown properties

→ Huge execution time improvement

Disadvantages

- Risk of Overhead: Overhead for evaluating optimality, re-optimization and reuse of intermediate results

→ Overhead might not be amortized by execution time improvement

> Outline

- **Classifications of Adaptive Query Processing**
  - Inter-Query Adaptation (static, late binding)
  - Inter-Operator Adaptation
  - Intra-Operator Adaptation
  - Tuple Routing
  - Rewriting-Based Classification
  - Reactive vs. Proactive Reoptimization
  - Synchronous vs. Asynchronous Reoptimization

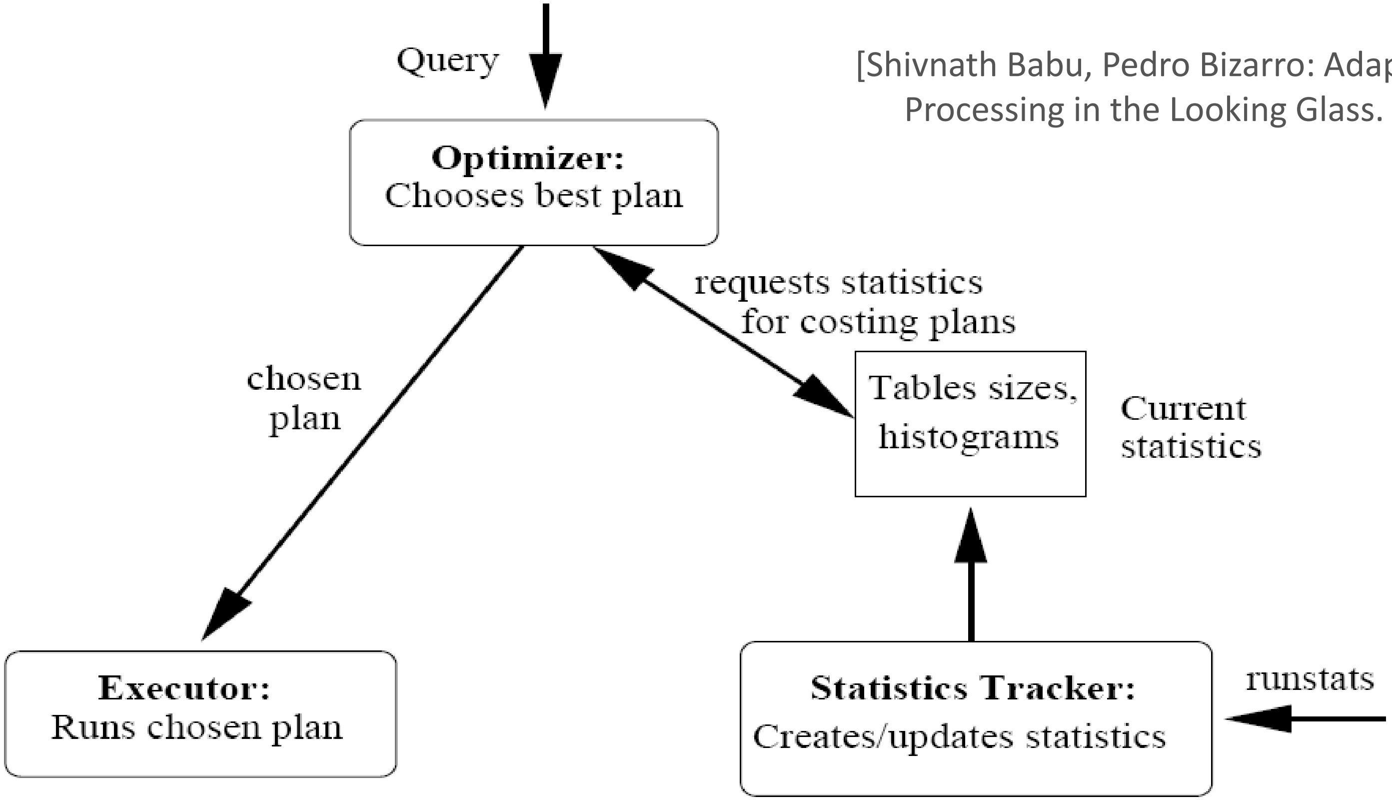
- **Adaptive Query Processing (in Action)**
  - Plan-Based Adaptation
  - Continuous-Query-Based Adaptation
  - Routing-Based Adaptation

Adaptive Query Processing (in Action)

> Conceptual System Architecture

Starting Point: Traditional DBMS (inter-query)

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

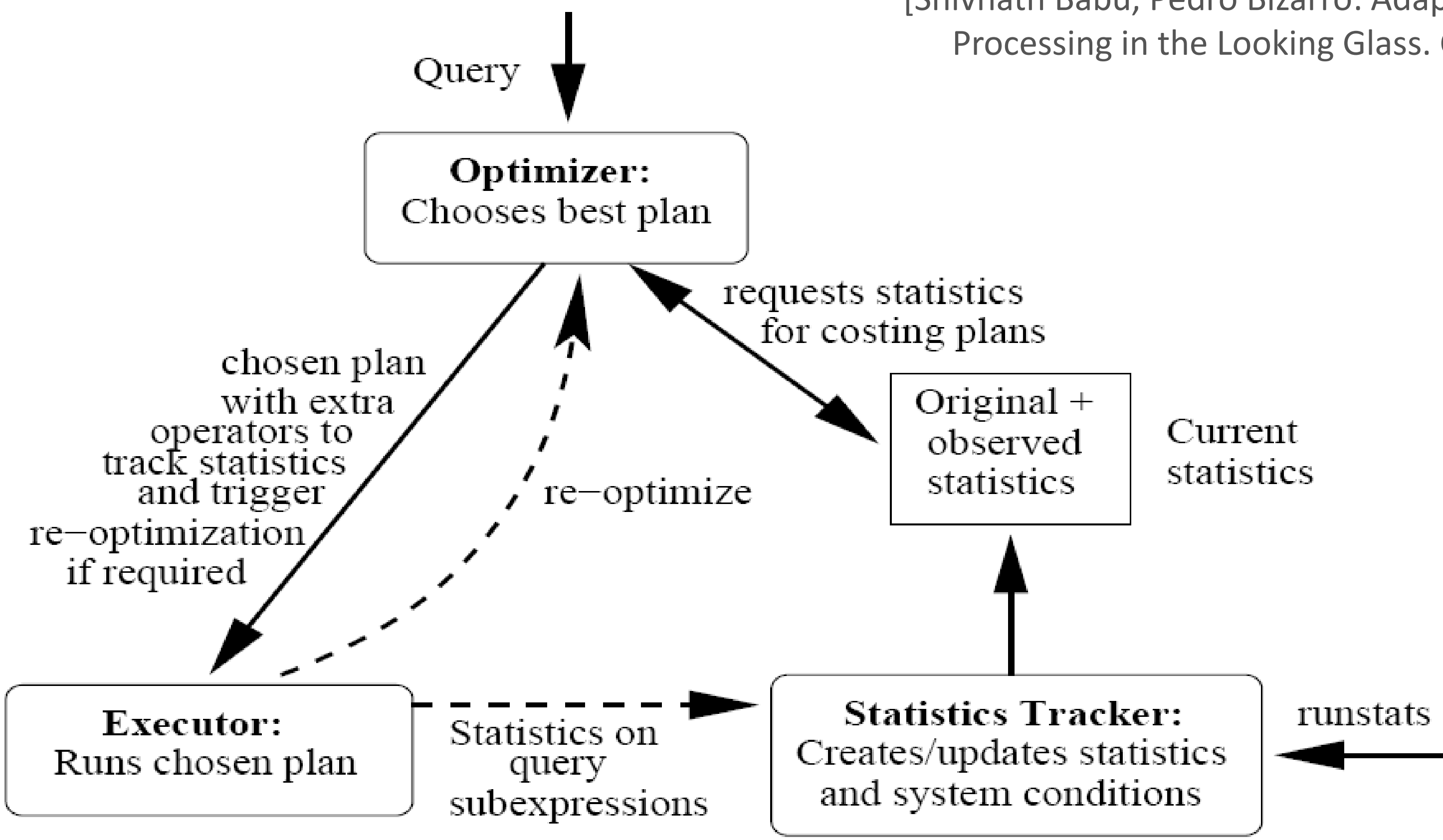


Interesting remark: [Surajit Chaudhuri: Query optimizers: time to rethink the contract? SIGMOD 2009]

Plan-Based Adaptation

Reoptimize the current plan during execution

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



> Approach Overview

Important plan-based approaches

- All **inter-operator** approaches are **synchronous**, while CQP uses **asynchronous** reoptimization

	Inter-Operator	Intra-Operator
Proactive	Rio (Babu, Bizarro, DeWitt)	–
Reactive	POP (Markl, Lohman) ReOpt (DeWitt)	CQP (Ives, Halevy)

> Approach Overview

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

Important plan-based approaches

- All **inter-operator** approaches are **synchronous**, while CQP uses **asynchronous** reoptimization

	Inter-Operator	Intra-Operator
Proactive	Rio (Babu, Bizarro, DeWitt)	–
Reactive	POP (Markl, Lohman) ReOpt (DeWitt)	CQP (Ives, Halevy)

> Statistic Collection

Solution Overview

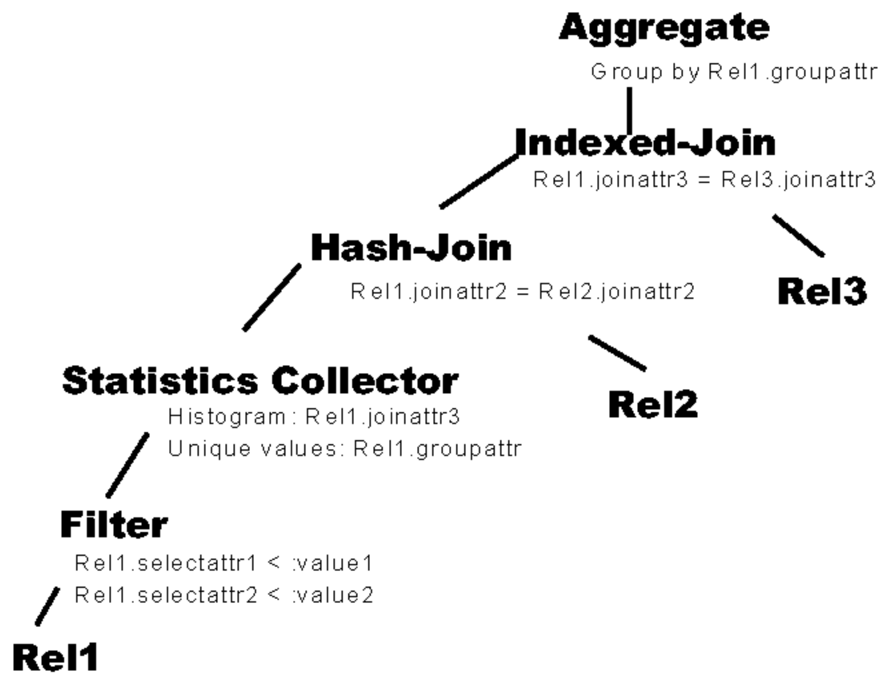
- monitor how the query is doing at key points, and consider dynamically re-optimizing those portions of the query which have not yet been started
- Using System-R optimizer

Core Aspects

- Annotated Query Execution Plans
- Runtime Collection of Statistics
- Dynamic Resource Re-allocation
- Query Plan Modification

Annotated Query Plans

- Statistic monitoring
  - Sizes and cardinalities, selectivities of predicates
  - Estimates of number of groups to be aggregated
- Add into tree statistic collectors
  - Must be collectable in a single pass
  - Will only help with portions of query "beyond" the current pipeline



> Statistic Collection Heuristics

Low-Overhead Statistics

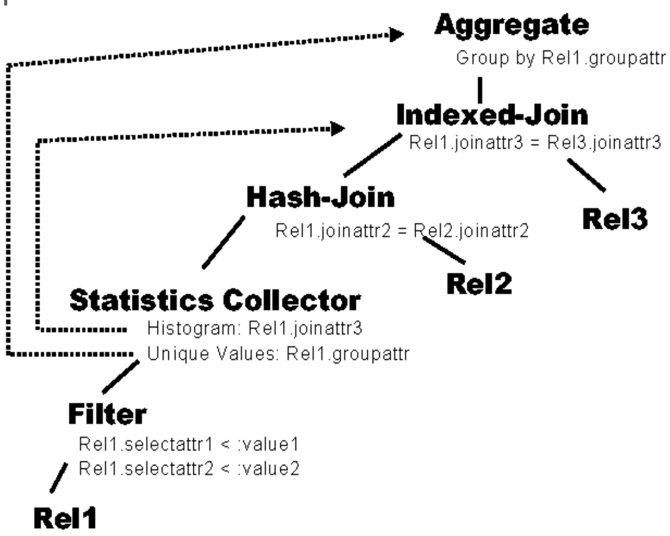
- Want to find "most effective" statistics
  - Don't want to gather statistics for "simple" queries
  - Want to limit effect of algorithm to maximum overhead ratio, μ
  - Factors: Probability of inaccuracy, Fraction of query affected

Inaccuracy Potentials

- The following heuristics are used:
  - Inaccuracy potential = low, medium, high
  - Lower if we have more information on table value distribution
  - 1+max of inputs for multiple-input selection
  - Always high for user-defined methods
  - Always high for non-equijoins
  - For most other operators, same as worst of inputs

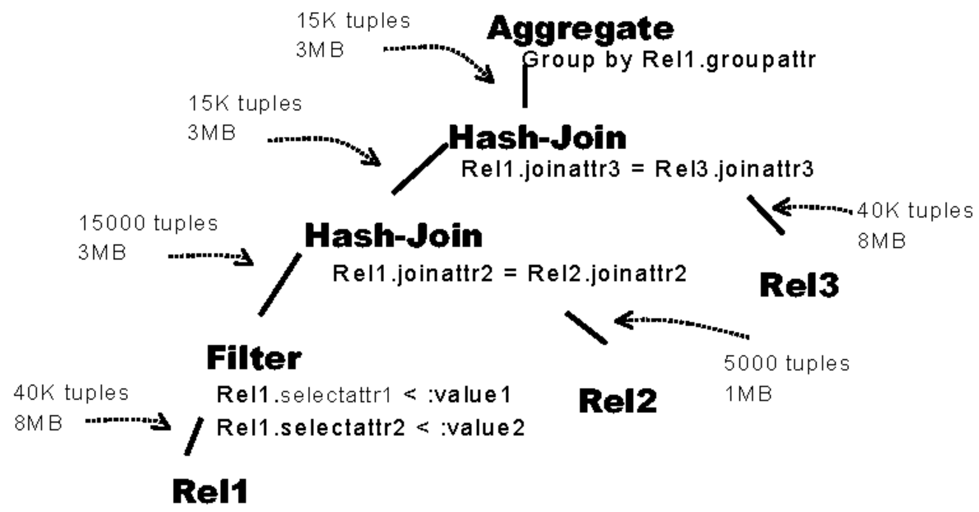
More Heuristics

- Check fraction of query affected
  - Check how many other operators use the same statistic
- The winner:
  - Higher inaccuracy potentials first
  - Then, if a tie, the one affecting the larger portion of the plan



Based on improved estimates,
we can modify the memory allocated
to each operation

- Results: less I/O, better performance
- Only for operations that have not yet begun executing



> Plan Modification

1) Only re-optimize part not begun

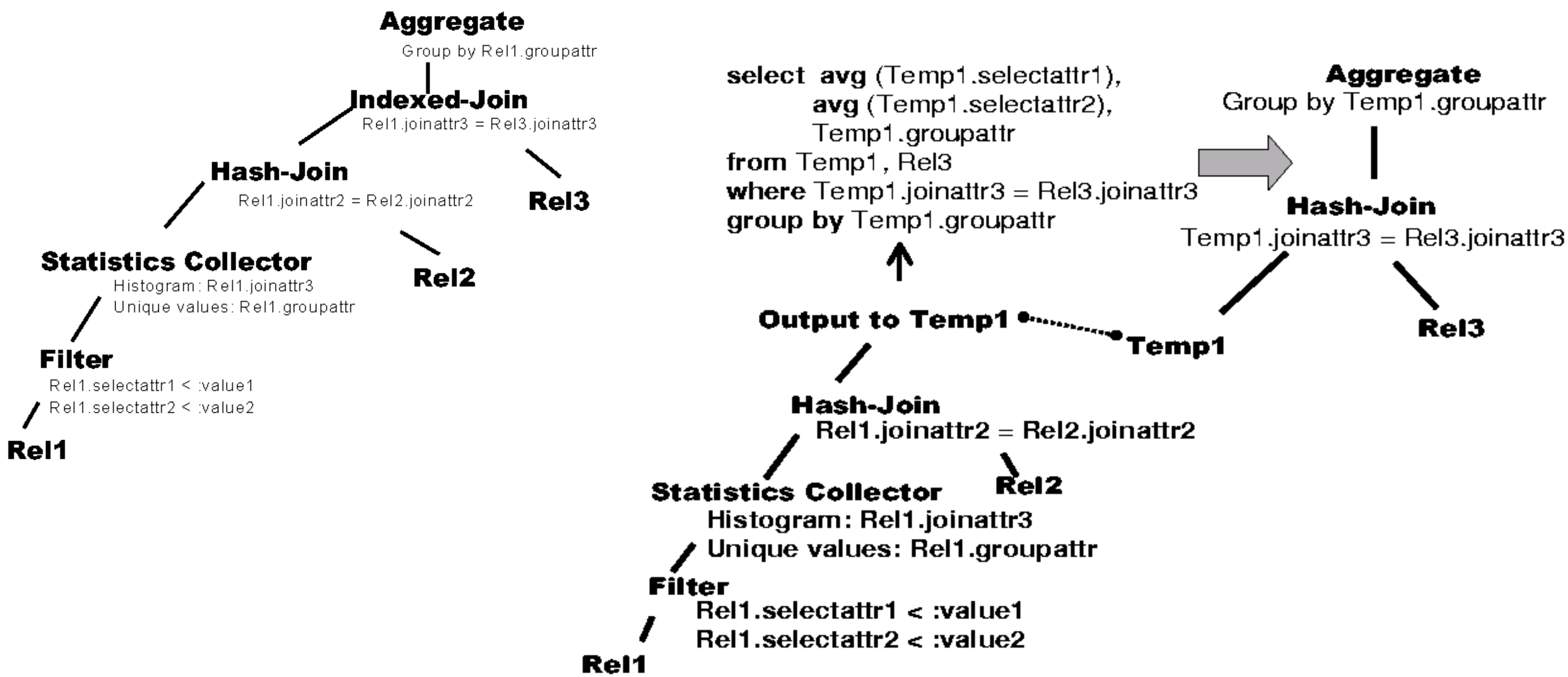
2) Suspend query, save intermediate in temp file

3) Create new plan for remainder, treating temp as an input

Discussion:
ad-hoc thresholds are a bad idea

➔ E.g. even a 100x error on very small relation may not make a difference in optimal plan

➔ Many unnecessary re-optimization steps



Approach Overview

[Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, Hamid Pirahesh: Robust Query Processing through Progressive Optimization, SIGMOD 2004]

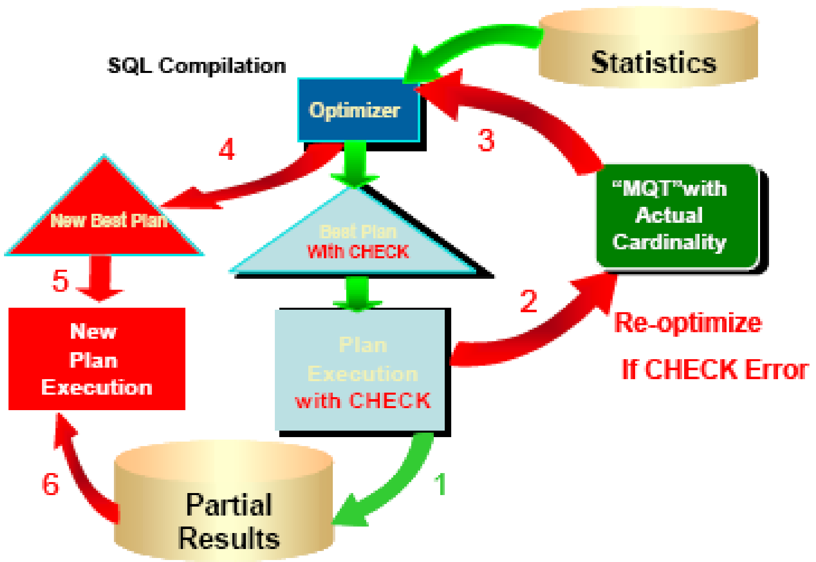
Important plan-based approaches

- All **inter-operator** approaches are **synchronous**, while CQP uses **asynchronous** reoptimization

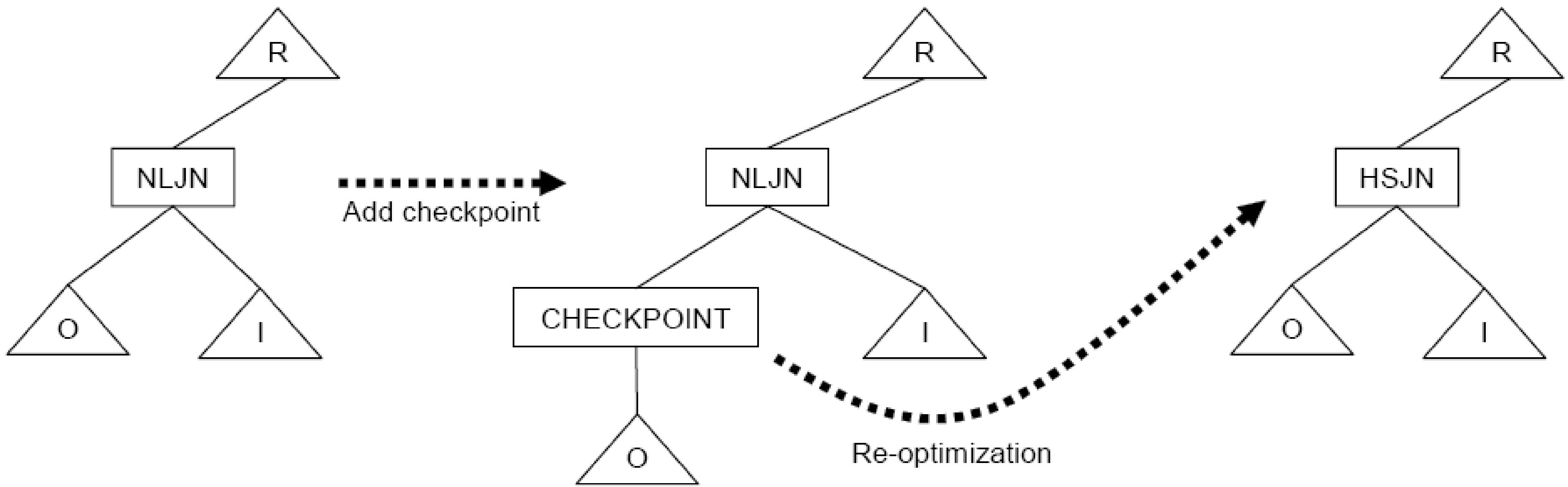
	Inter-Operator	Intra-Operator
Proactive	Rio (Babu, Bizarro, DeWitt)	–
Reactive	POP (Markl, Lohman) ReOpt (DeWitt)	CQP (Ives, Halevy)

Solution Overview

- Lazily trigger reoptimization during execution if cardinality counts indicate current plan is suboptimal
- introduces checkpoint (CHECK) operator to compare actual vs estimated cardinality
- precompute optimal cardinality ranges



Risk vs. Opportunity



CHECK operator to find if a plan is suboptimal

- At optimization time, find out cardinality range
(at CHECK location) for which plan is optimal
- At run time, ensure cardinality within [l,u]
- If violated, stop plan execution and reoptimize

Location of CHECKs

Re-optimize

- taking observed cardinality into account, and
- exploiting intermediate results where beneficial
- Heuristic: limit number of reoptimizations (default: 3)

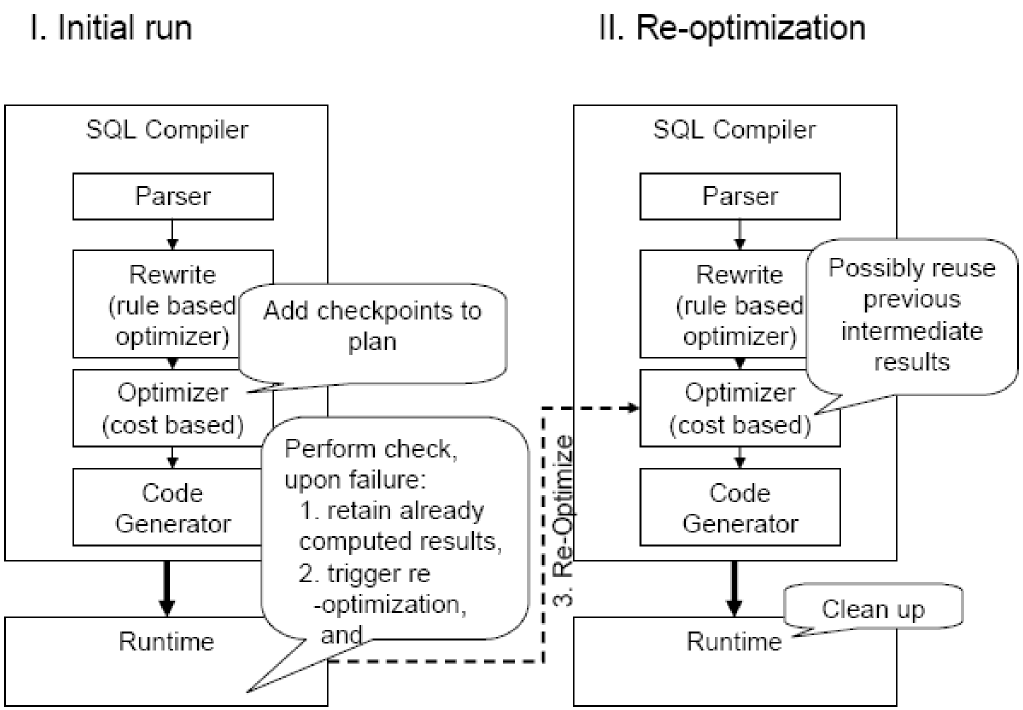


Figure 1: Progressive Optimization architecture

Consider a plan edge e that flows rows into operator o,

- let P be the subplan rooted at *o.*
- The ***validity range*** for *e* is an upper and lower bound on the number of rows flowing through *e,* such that if the range is violated at runtime, we can guarantee P is suboptimal

Finding Optimality Ranges

- Plan P_{opt} with root operator o_{opt} is being compared with another plan P_{alt} different only in the root operator o_{alt}.
- Need to solve
  - *cost(P_{alt} , c) – cost(P_{opt} , c) = 0*
  - where c is the cardinality on edge *e*
- Cost functions can be complex/ non-linear/non-continuous
- Using Newton-Raphson Iteration

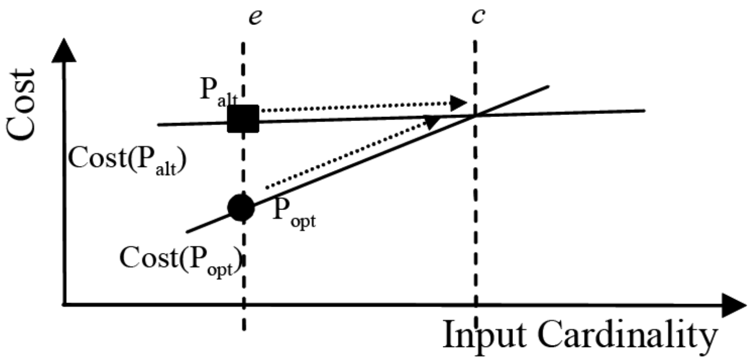


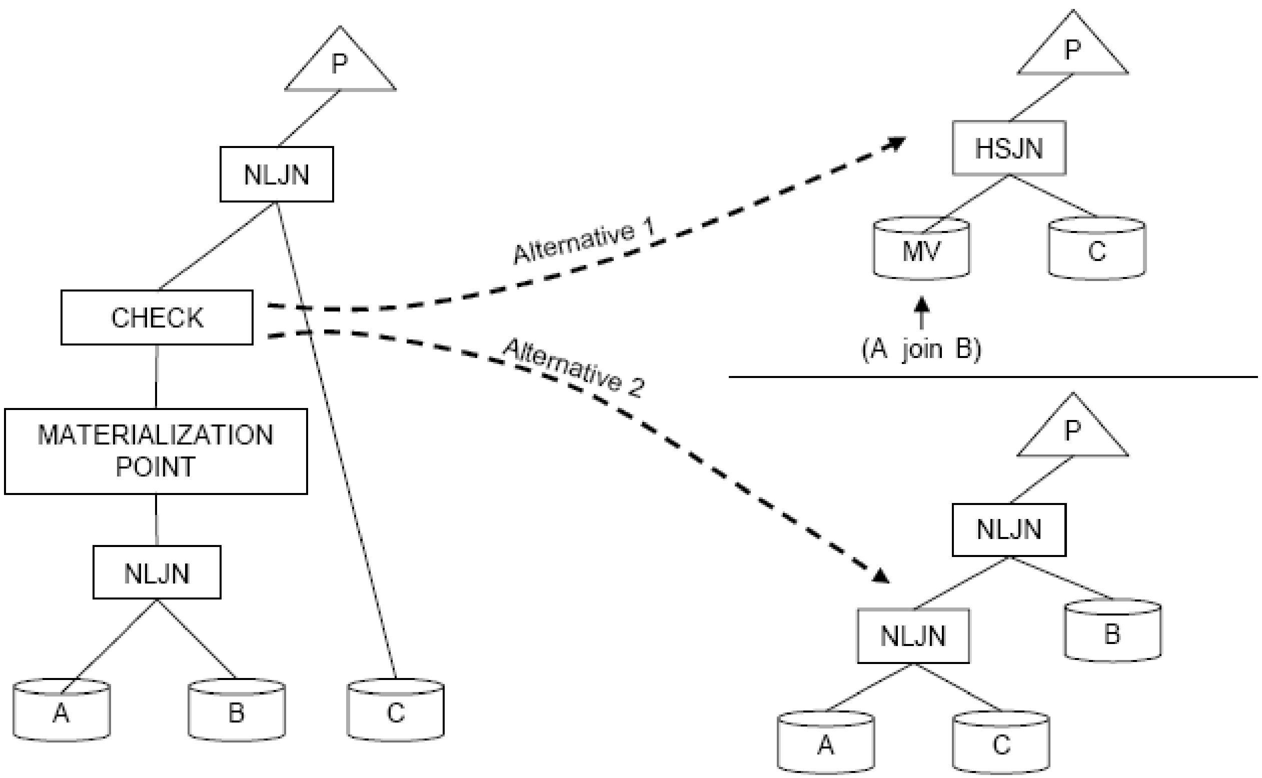
Figure 4: Computing the Upper Bound of a Validity Range

Conservative Detection of Suboptmalities

Discussion:
black-box approach for the sub query plan
➔ No optimality conditions
➔ No directed reoptimization
➔Validity ranges not combined for multiple sub graphs?

Optional Use of Materialized Views

- But all intermediate results are stored as temporary MVs
- Optimizer decides to use them or not (trashing or reusing as optimization problem)

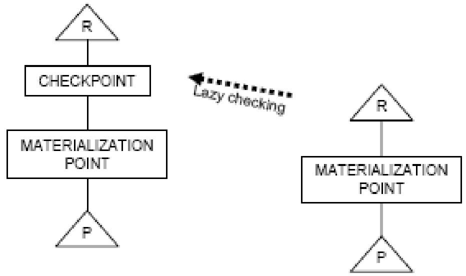


> Variants of CHECK

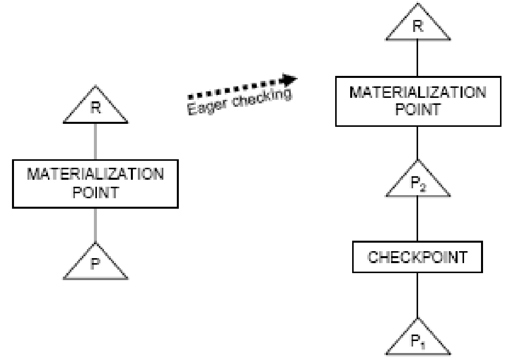
Variants applicable in different cases, trade off risk for opportunity

Variants

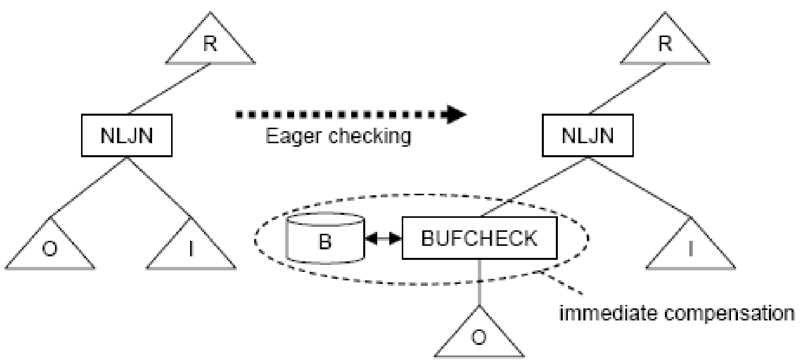
- Lazy checking
  - Adding CHECKs above a materialization point (SORT, TEMP etc)



- Lazy checking with eager materialization
  - Can insert materialization point if it does not exists already
- Eager checking without compensation
  - CHECK is pushed down the materialization point, into pipeline



- Eager checking with buffering
  - Delayed pipelining
- Eager checking with deferred compensation
  - Anti-join all rows returned to the user with the new result stream



> Approach Overview

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

Important plan-based approaches

- All **inter-operator** approaches are **synchronous**, while CQP uses **asynchronous** reoptimization

	Inter-Operator	Intra-Operator
Proactive	Rio (Babu, Bizarro, DeWitt)	–
Reactive	POP (Markl, Lohman) ReOpt (DeWitt)	CQP (Ives, Halevy)

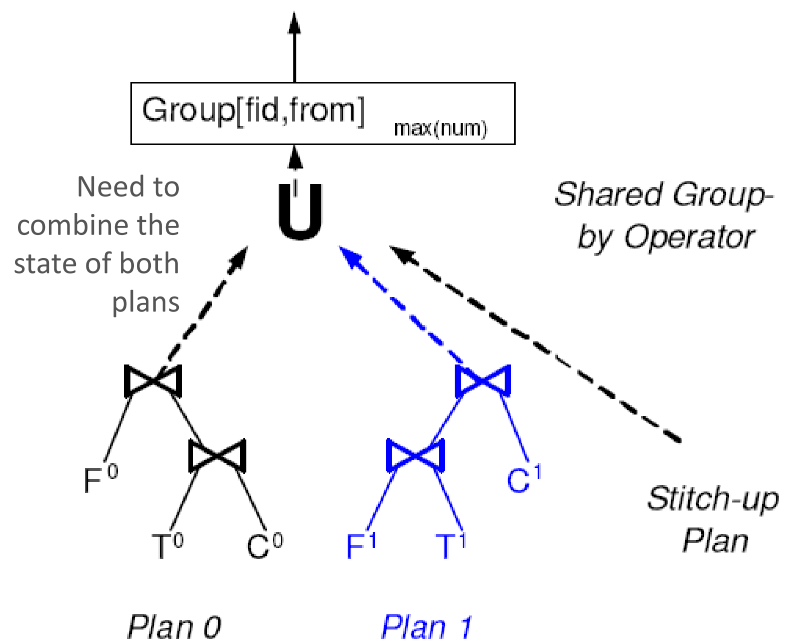
Drawbacks of Inter-Operator Re-Optimization

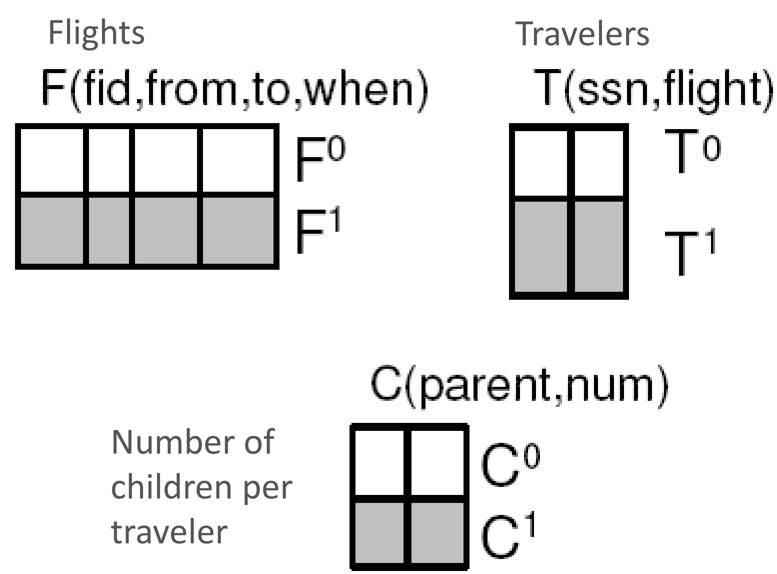
- Possibly trashing of intermediate results
- Use of materialization points might be too coarse-grained

History:
Convergent Query Processing
renamed to
Corrective Query Processing

Core Idea (within the data integration platform Tukwila)

- Data partitioning – use new plans for new data only
- Combine resulting data partitions in so-called stitch-up phases





> Pipelined Hash Join

Pipelined Hash Join

- symmetric
- non-blocking
- memory-consuming

- foreach arriving tupel (P1 | P2)
  - 1) Probing
  - 2) Building

fid	from	to	when	ssn
1007	Berlin	Chicago	3456712	70001
1007	Berlin	Chicago	3456712	70002
1001	Dresden	LA	1234567	70003
1001	Dresden	LA	1234567	70004

Hashtable 1

1001
1007

$\bowtie$ fid=flight

Hashtable 2

1007
1007
1001
1001

Flight

fid	from	to	when
1001	Dresden	LA	1234567
1007	Berlin	Chicago	3456712

P1

Travelers

ssn	flight
70001	1007
70002	1007
70003	1001
70004	1001

P2

> Adaptive Data Partitioning

Algebraic key property

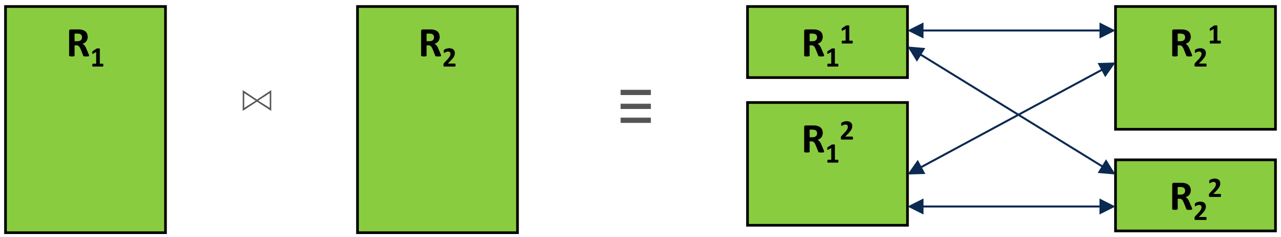
- Distribute relational UNION through PSJ operations
- For joins; in general: $$R_1 \bowtie \ldots \bowtie R_m = \bigcup_{1 \le c_1 \le n, \ldots, 1 \le c_m \le n} (R_1^{c1} \bowtie \ldots \bowtie R_m^{c_m})$$
- Example:

  If $R_1 = R_1^1 \cup R_1^2$, (horizontal partitioning)
  $R_2 = R_2^1 \cup R_2^2$

  then: $R_1 \bowtie R_2$

  $\equiv (R_1^1 \cup R_1^2) \bowtie (R_2^1 \cup R_2^2)$

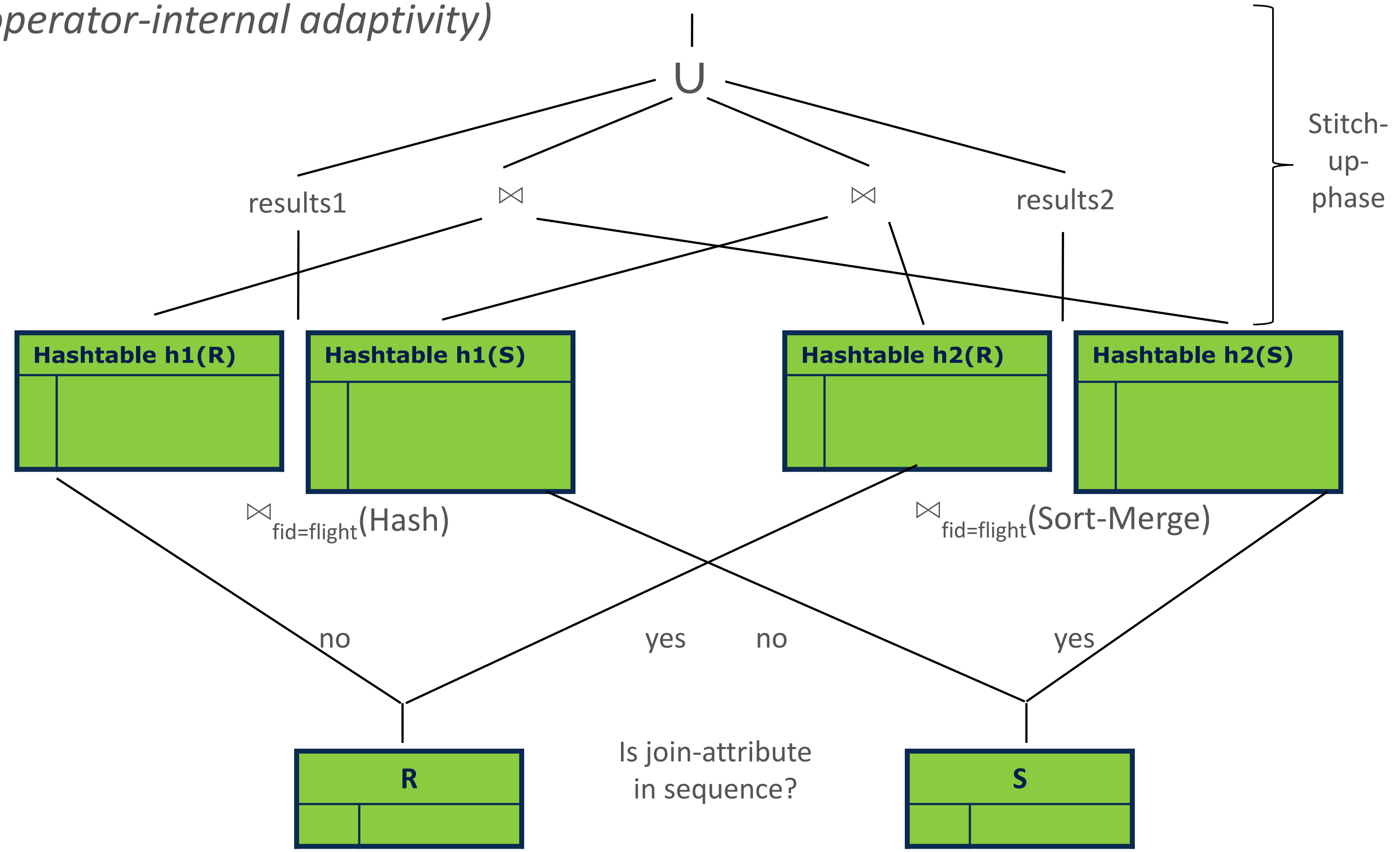
  $\equiv (R_1^1 \bowtie R_2^1) \cup (R_1^2 \bowtie R_2^2) \cup (R_1^1 \bowtie R_2^2) \cup (R_1^2 \bowtie R_2^1)$



Union over all join combinations of partitions

(state in terms of results and hash maps)

Complementary Join Pair (operator-internal adaptivity)



> Preaggregation (Partial Eager Group By)

Eager Group By

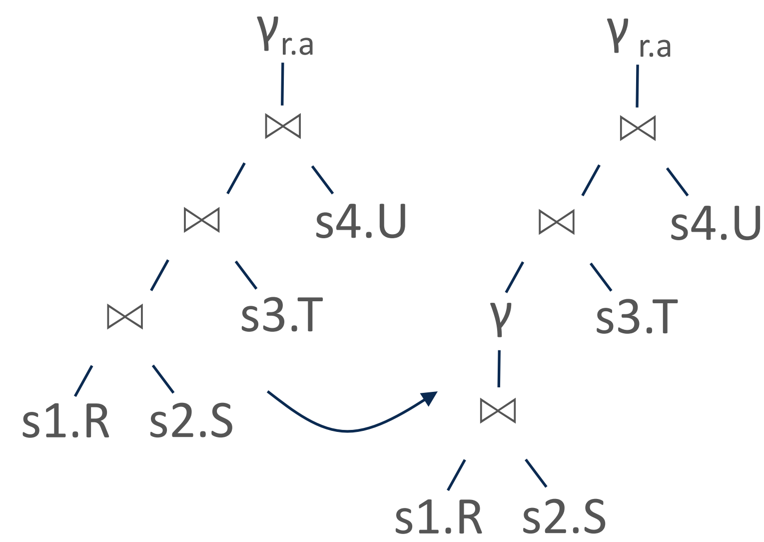
- Cost-based rewriting decision

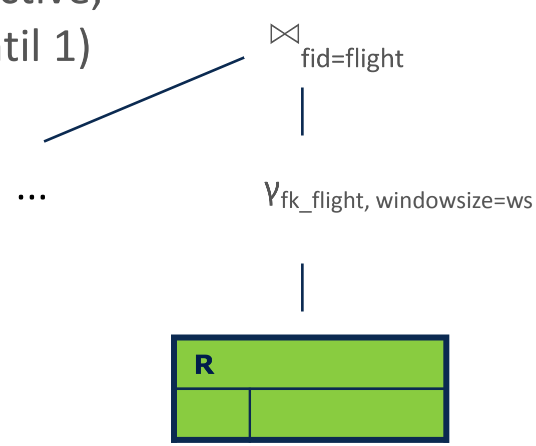
Adjustable-window pre-aggregation

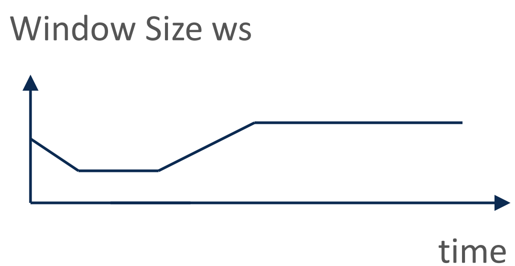
- Pipelined pre-aggregation
- Window required for pipeline character

Control Strategy

- if current window is effective, increase next window size
- if current window is not effective, reduce next window size (until 1)
- used before
  - Join
  - Final-Aggregation







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

Important plan-based approaches

- All **inter-operator** approaches are **synchronous**, while CQP uses **asynchronous** reoptimization

	Inter-Operator	Intra-Operator
Proactive	Rio (Babu, Bizarro, DeWitt)	–
Reactive	POP (Markl, Lohman) ReOpt (DeWitt)	CQP (Ives, Halevy)

Current Re-Optimizers (reactive)

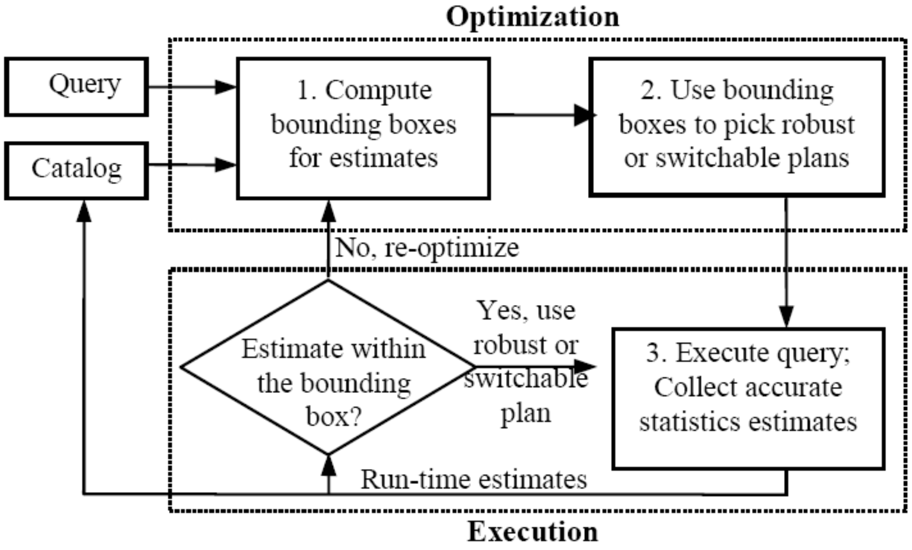
- use traditional optimizers to pick a plan and
- if violations are detected, they stop execution, and re-optimize

Shortcomings of the use of traditional query optimizers (TRADs)

- Pr1: pick plans depending on uncertain statistics, making *re-optimization very likely*
- Pr2: if TRAD chooses a new plan, *work* until estimation error is found *will be lost*
- Pr3: limited statistics collection ability *leading to new mistakes*

Construction of proactive re-optimizer (Rio)

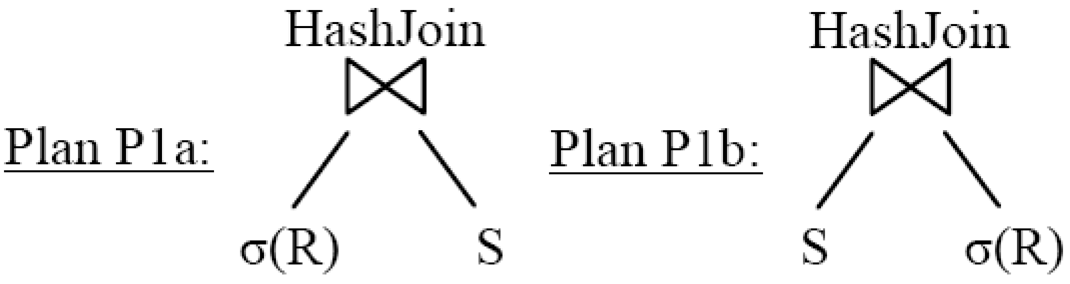
- *bounding boxes* (intervals instead of single-point),
- generate *robust* and *switchable* plans
- minimizing re-optimization
- Reuse intermediate in case of plan change
- using *Random Sample processing*

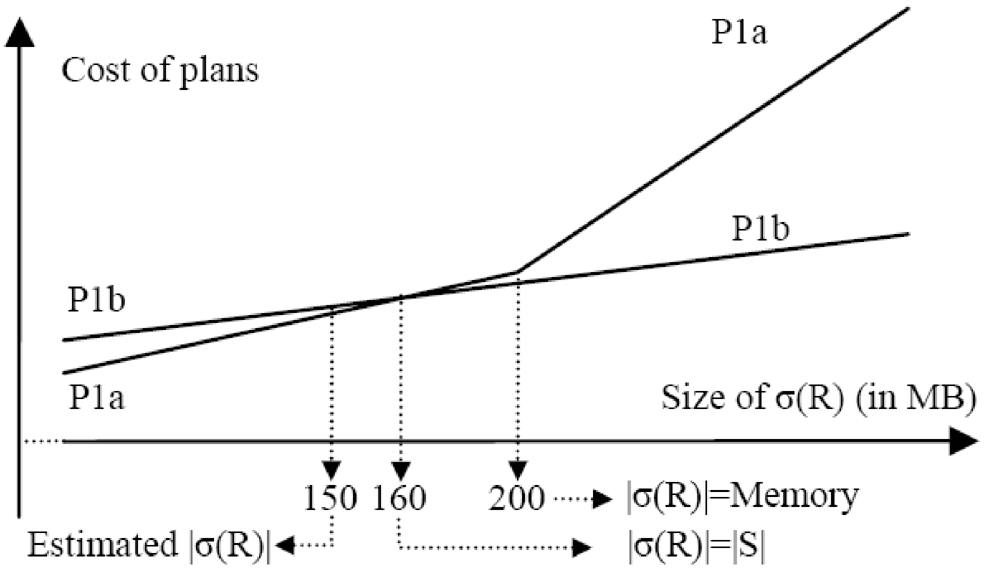


Example 1

- DB-cache=200MB, |R|=500MB, |S|=160MB, and |σ(R)|=300MB
- optimizer estimates |σ(R)|=150MB

Query:
SELECT *
FROM R, S
WHERE R.a=S.a AND
R.b>K1 AND
R.c>K2



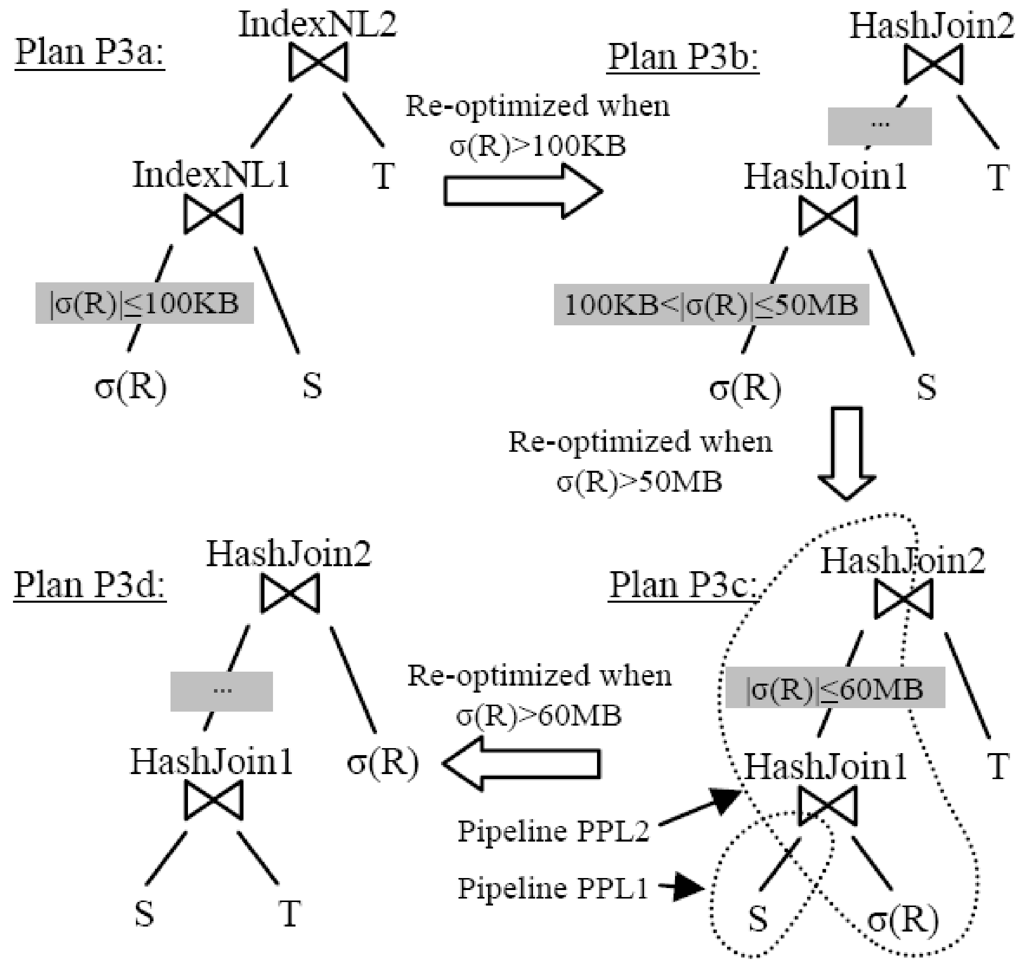


- TRAD will choose P1a resulting in making two passes over R and S (P1b needs only one)
- Reactive re-optimizer (VRO) will choose P1a computing a validity range for P1a 100KB≤|σ(R)|≤160MB

Example 2

Query:
SELECT *
FROM R,S,T
WHERE R.a=S.a AND
S.b=T.b AND
R.c>K1 AND R.d=K2

- Assume sizes of the tables are known accurately to be |R|=200MB, |S|=50MB, and |T|=60MB
- Further assume |σ(R)|=80MB
- optimizer underestimates as 40KB

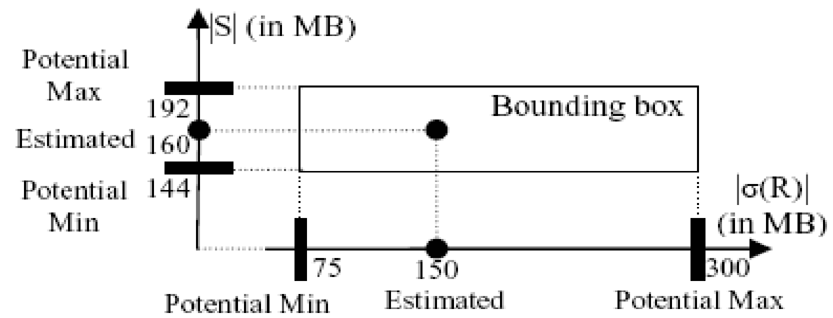


- Based on these statistics, the TRAD chooses Plan P3a
- Due to limited information of the actual size of |σ(R)|, a reactive re-optimizer will trigger re-optimization step-by-step three times moving from P3a to P3d

> Optimization with Bounding Boxes

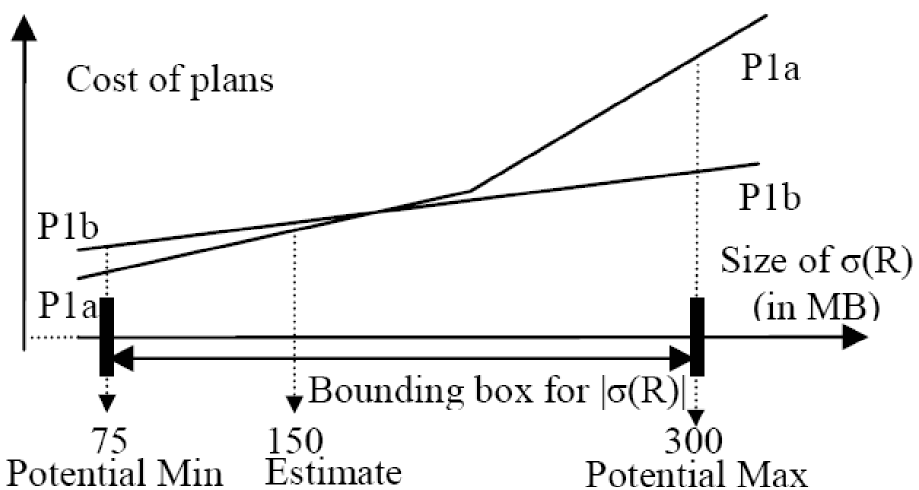
Bounding Boxes and Plans: four cases

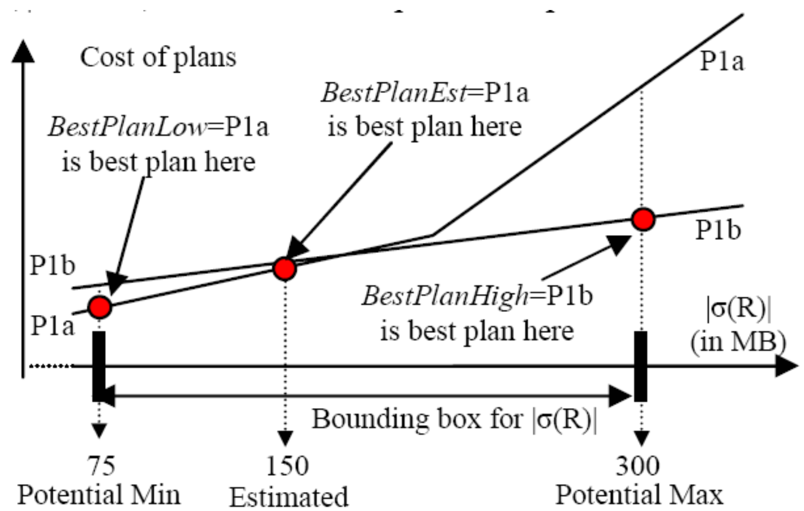
- Single optimal plan
- Single robust plan
- Switchable plan (set S of plans p)
- None of the above



Example

- A) robust plan P1b
- B) switchable plans (always 3 plans)

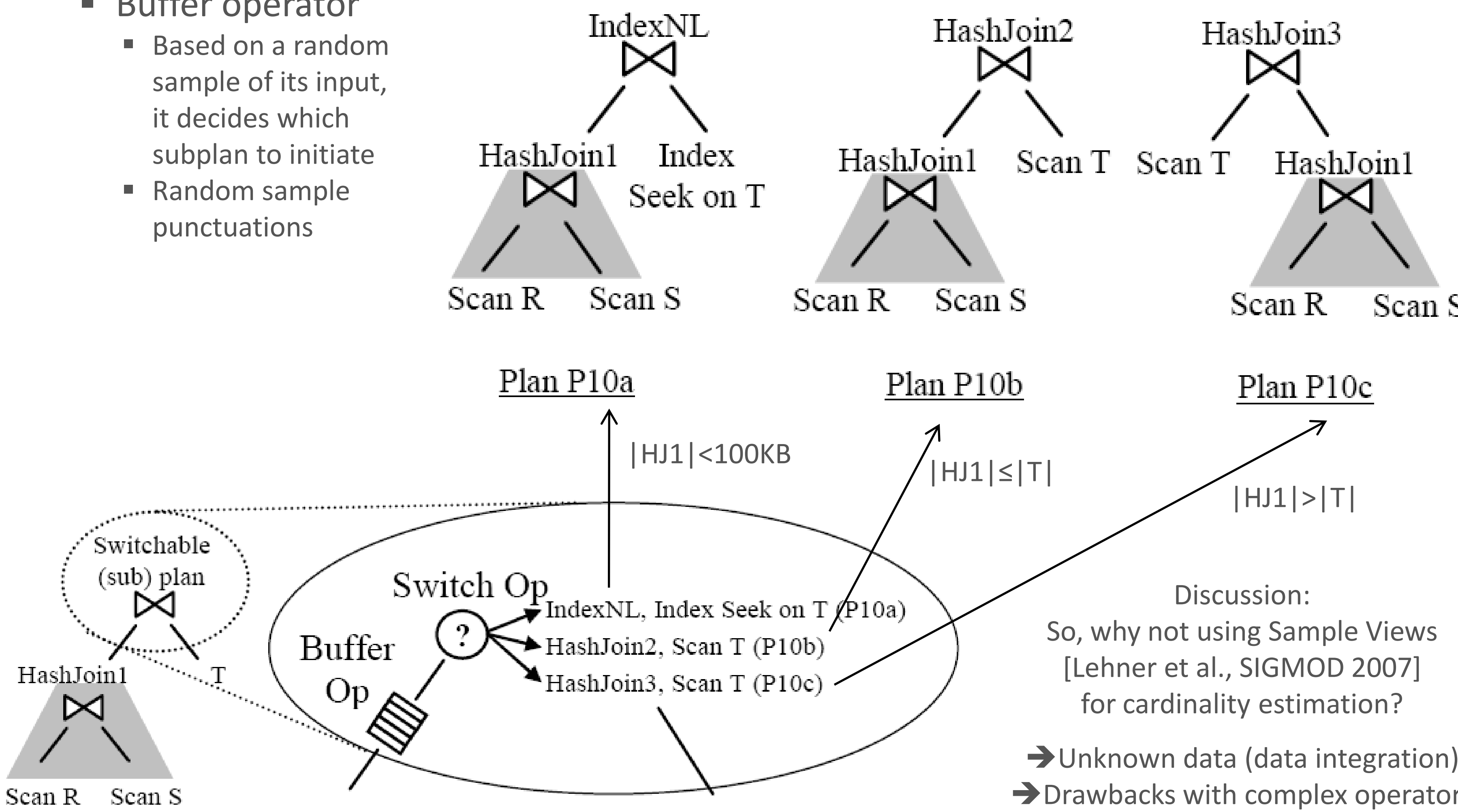




> Re-optimization using Switch-Operator

Switchable Plans

- Reuse intermediate results (no trashing)
- Buffer operator
  - Based on a random sample of its input, it decides which subplan to initiate
  - Random sample punctuations



Discussion:
So, why not using Sample Views [Lehner et al., SIGMOD 2007] for cardinality estimation?

➔Unknown data (data integration)
➔Drawbacks with complex operators

> Discussion

Lesson Learned

- Reactive, inter-operator reoptimization
- Reactive, inter-operator reoptimization with validity ranges
- Reactive, intra-operator reoptimization
- Proactive, inter-operator reoptimization with bounding boxes

Advantages

- Fast adaptation during runtime of a single query
- Huge opportunities for optimization (unknowns, changing workload)

Disadvantages

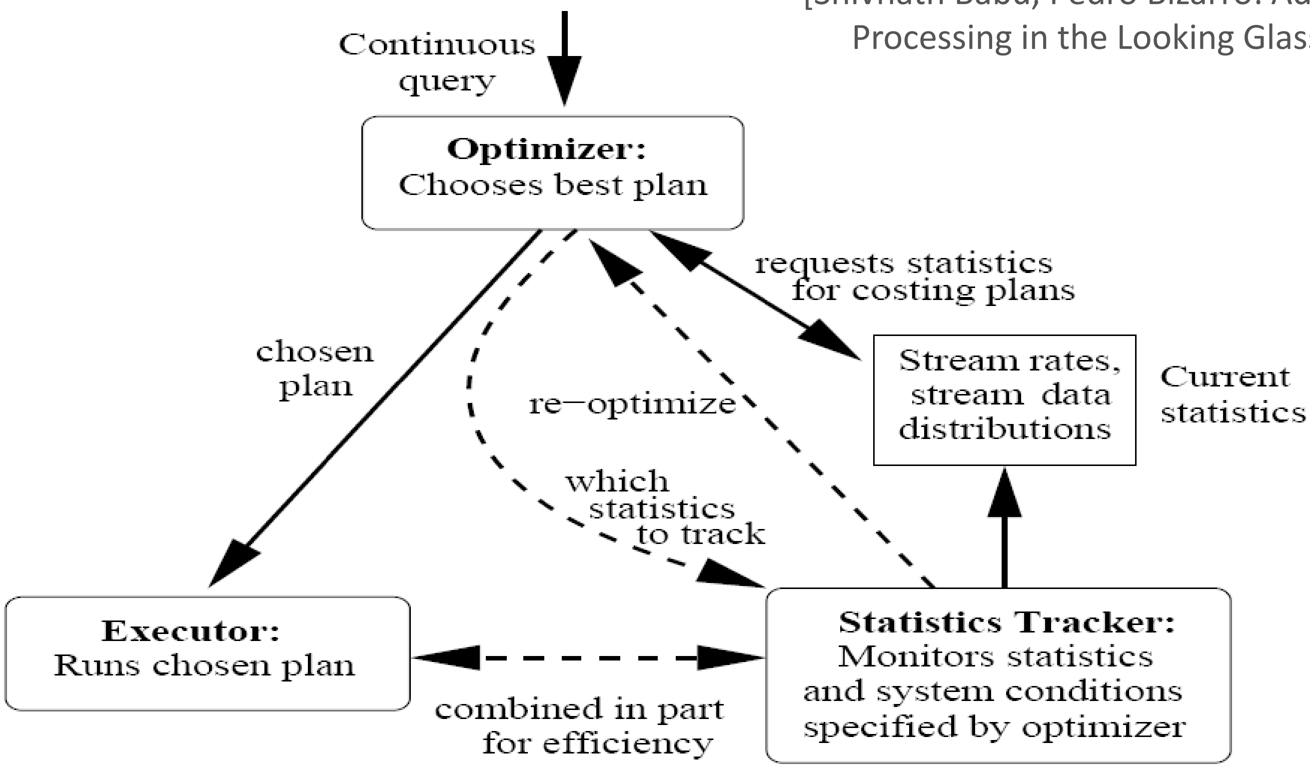
- Partly trashing of intermediate results
- Reoptimization does not guarantee to find a better plan
- Overhead for combining/reusing intermediate results

Continuous-Query-Based Adaptation

> Conceptual System Architecture

Reoptimize continuous query during runtime

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



Fundamental Differences to Mid-Query Re-Optimization

- Larger optimization scope: optimize the continuous query rather than only the current query plan
- Higher optimization potential: all kinds of operators
  - E.g., selection reordering not applicable for plan-based reoptimization but for CQ-based adaptation
- Need for incremental statistics maintenance (no local data)
- Monitor arbitrary changes in stream characteristics and systems conditions during runtime

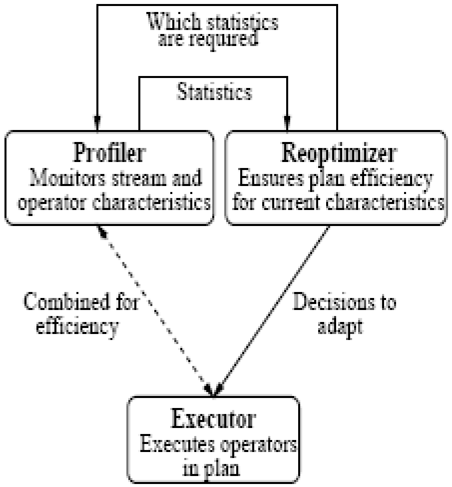
Example Systems

- CAPE (Worcester)
- NiagaraCQ (Wisconsin-Madison)
- StreaMon (Stanford)

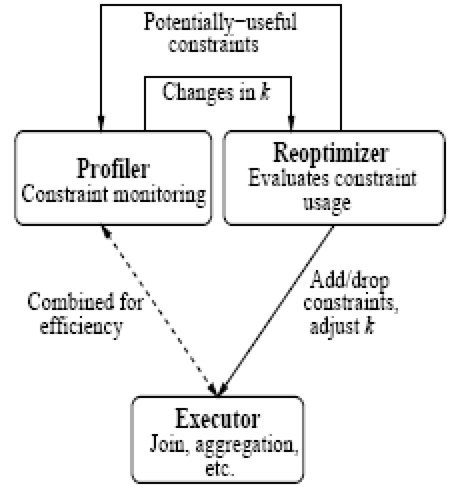
Basic Concepts

- Track only relevant statistics
- Sampling-based techniques
- Combine statistics tracking and query execution whenever possible

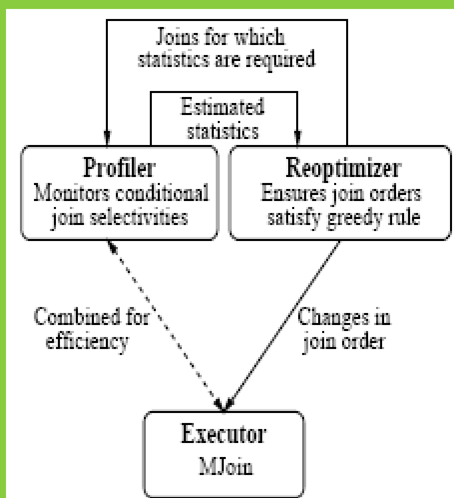
StreaMon Statistic Tracking



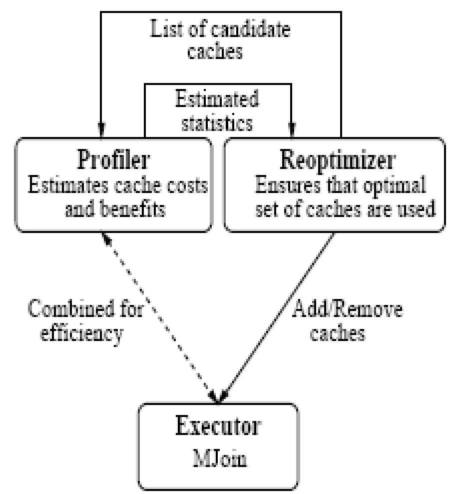
Adaptive Memory Minimization



Adaptive Join Ordering



Adaptive Caching for Joins

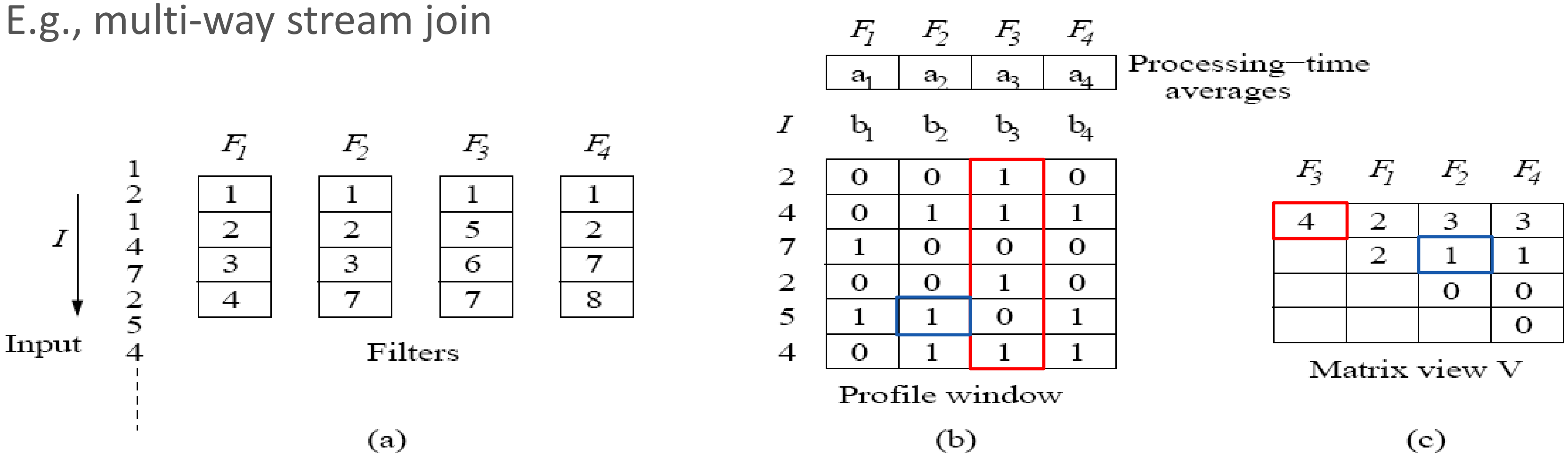


[Shivnath Babu, Jennifer Widom: StreaMon: An Adaptive Engine for Stream Query Processing. SIGMOD Conference 2004]

[Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, Jennifer Widom: Adaptive Ordering of Pipelined Stream Filters. SIGMOD 2004]

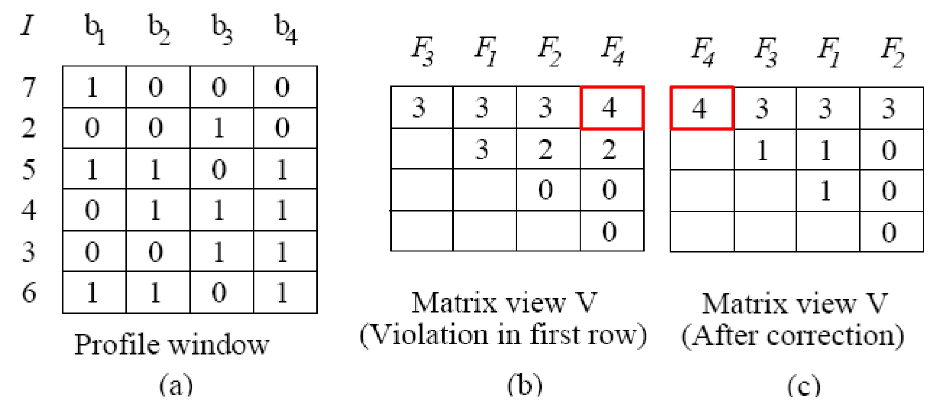
Pipelined Filters

- Stream of tuples is processed by a set of commutative filters
- E.g., multi-way stream join



A-Greedy Algorithm (Adaptive Greedy)

- A-Greedy Profiler: maintains a matrix view over a window of profile tuples
- A-Greedy Re-Optimizer: Checks for violations of greedy invariant and reoptimizes if necessary



Lesson Learned

- Adaptation by Dynamic Exploration of Stream Characteristics
- Requirement of Dynamic Plan Migration (Moving/Recomputing State)

[Yali Zhu, Elke A. Rundensteiner, George T. Heineman: Dynamic Plan Migration for Continuous Queries Over Data Streams. SIGMOD 2004]

Advantages

- Huge optimization potential (scope not limited to current query)
- Incremental statistics maintenance required anyway (missing knowledge)

Disadvantages

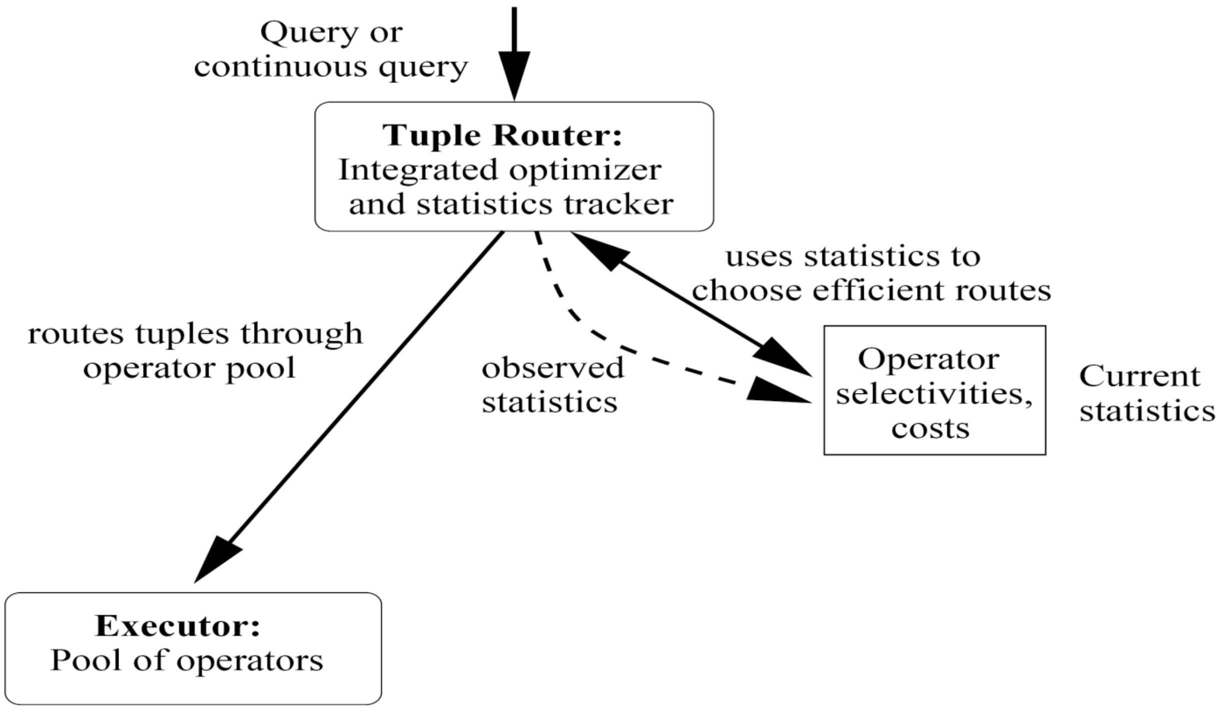
- Possibly large overhead for full exploration (profiling)
- No evaluation of optimization benefit vs. costs for dynamic plan migration

Routing-Based Adaptation

> Conceptual System Architecture

No optimizer component used (optimization by choosing different routes)

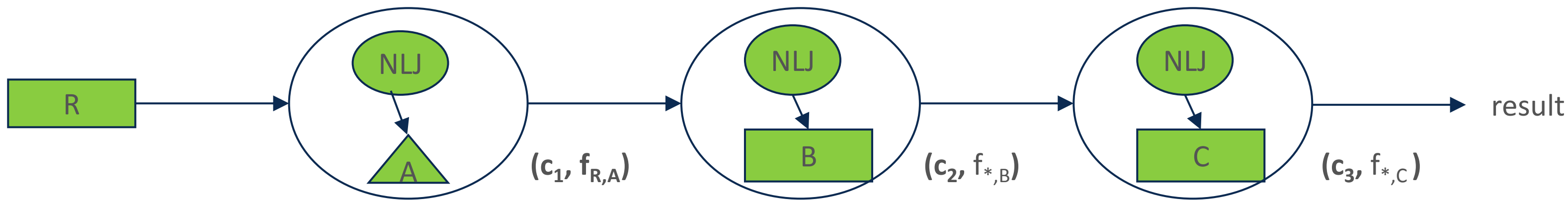
[Shivnath Babu, Pedro Bizarro: Adaptive Query Processing in the Looking Glass. CIDR 2005]



> Adapting the Join Order using an Eddy

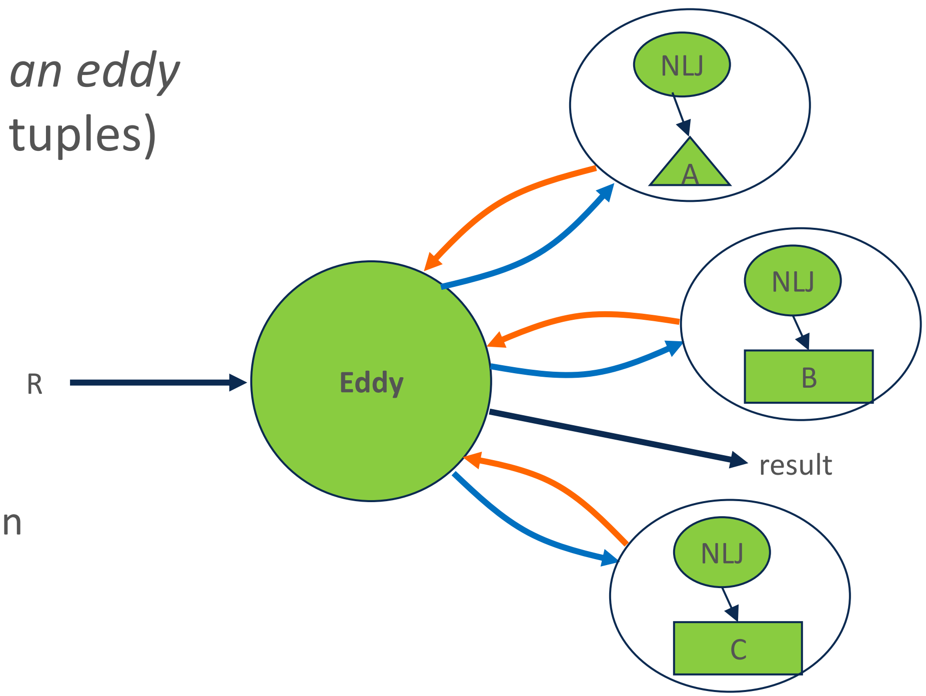
Ron Avnur, Joseph M. Hellerstein: Eddies: Continuously Adaptive Query Processing, SIGMOD 2000

A traditional query plan



Pipelined query execution using an eddy
(Query processing as routing of tuples)

- An eddy operator
  - Intercepts tuples from sources and output tuples from operators
  - Executes query by routing source tuples through operators



Encapsulates all aspects of adaptivity in a “standard” dataflow operator: measure, model, plan and actuate.

> Eddies – State Management

Used to decide validity and need of applying operators

An R Tuple: r1

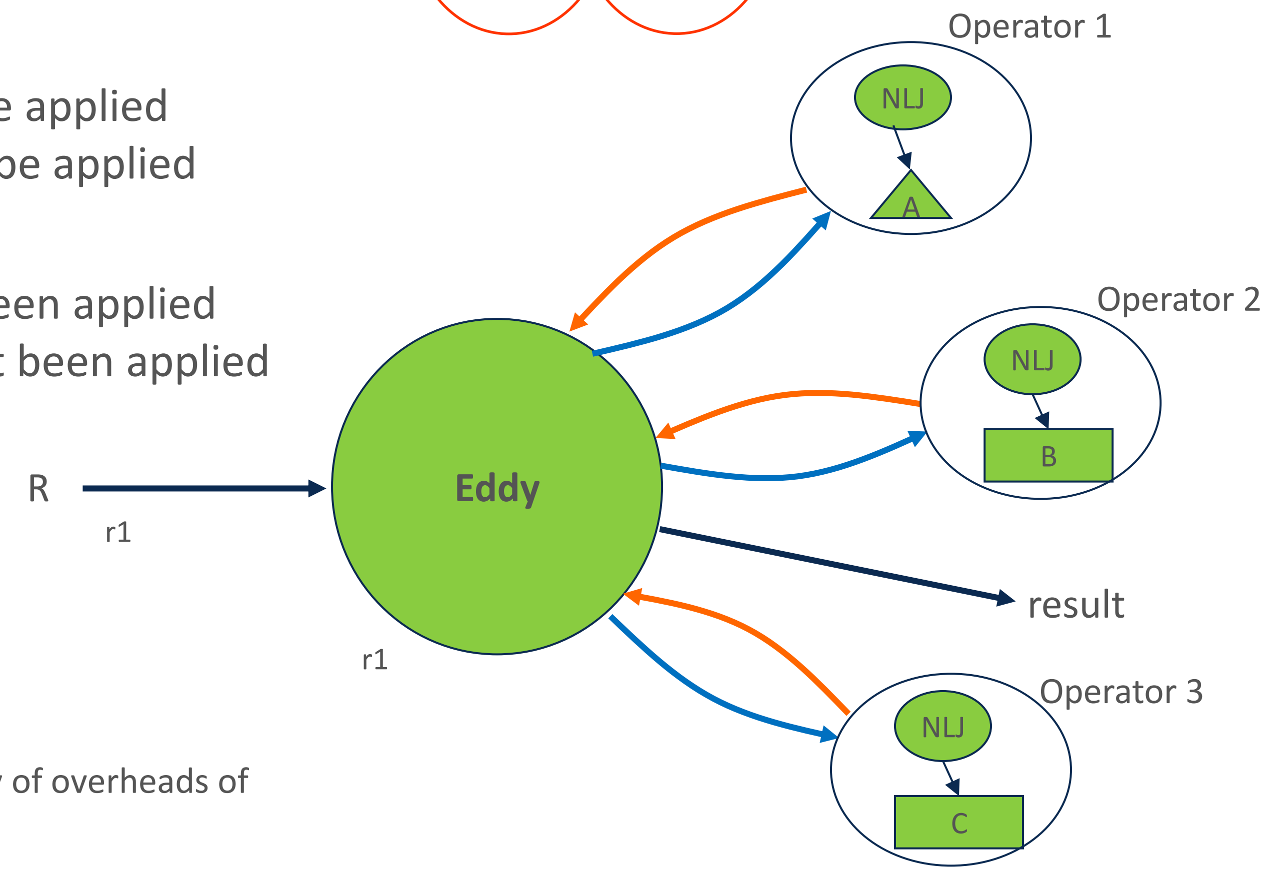
a	b	c	…	ready	done
15	10	AnameA	…	111	000

ready bit i:

- 1 → operator i can be applied
- 0 → operator i can't be applied

done bit i:

- 1 → operator i has been applied
- 0 → operator i hasn't been applied



[Amol Deshpande: An initial study of overheads of eddies. SIGMOD Record, 2004]

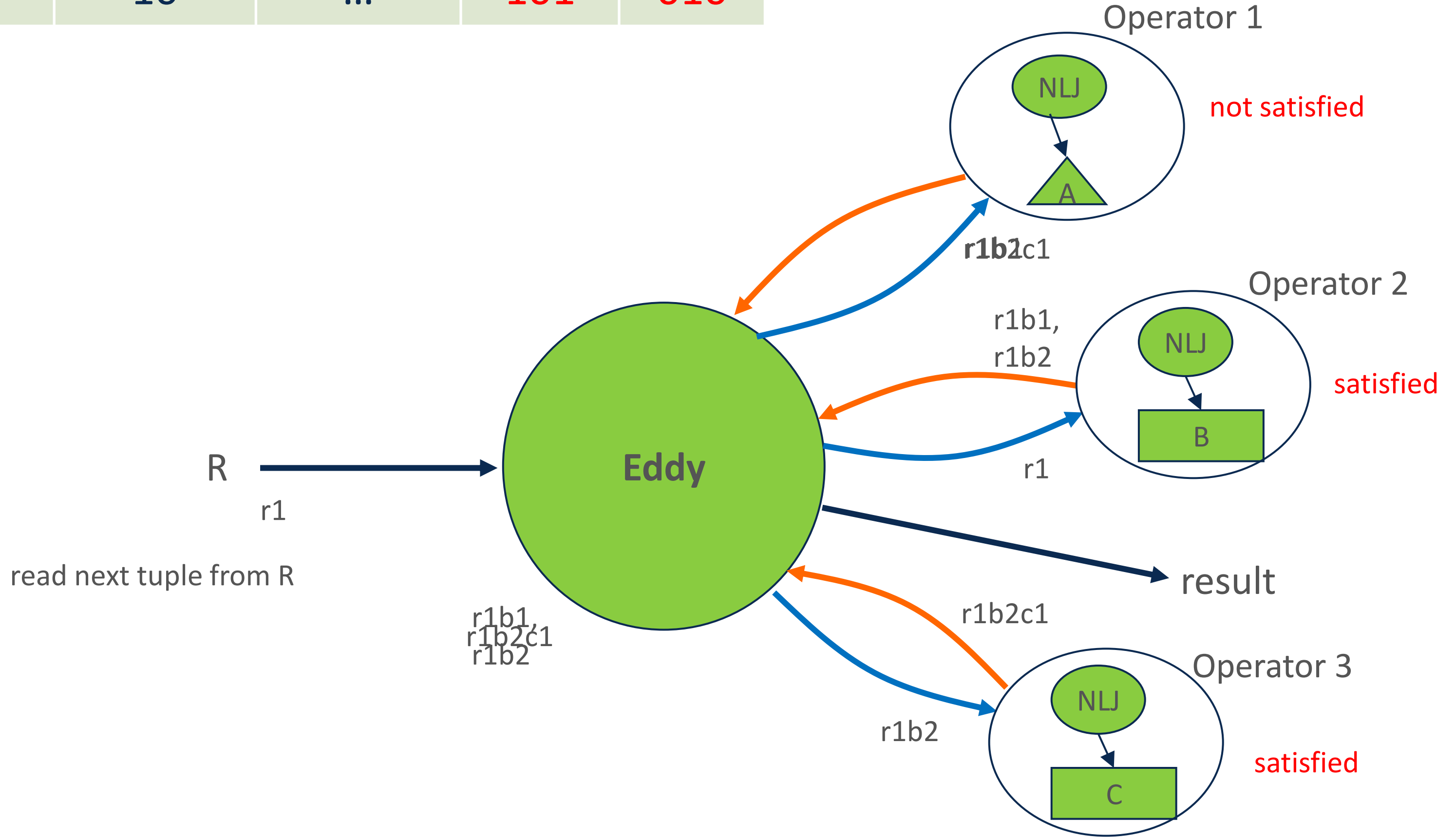
> Eddies – Query Execution

r1b1

a	b	…	ready	done
15	10	…	100	011

r1b2

a	b	…	ready	done
15	10	…	101	010



> Eddies – Query Execution (2)

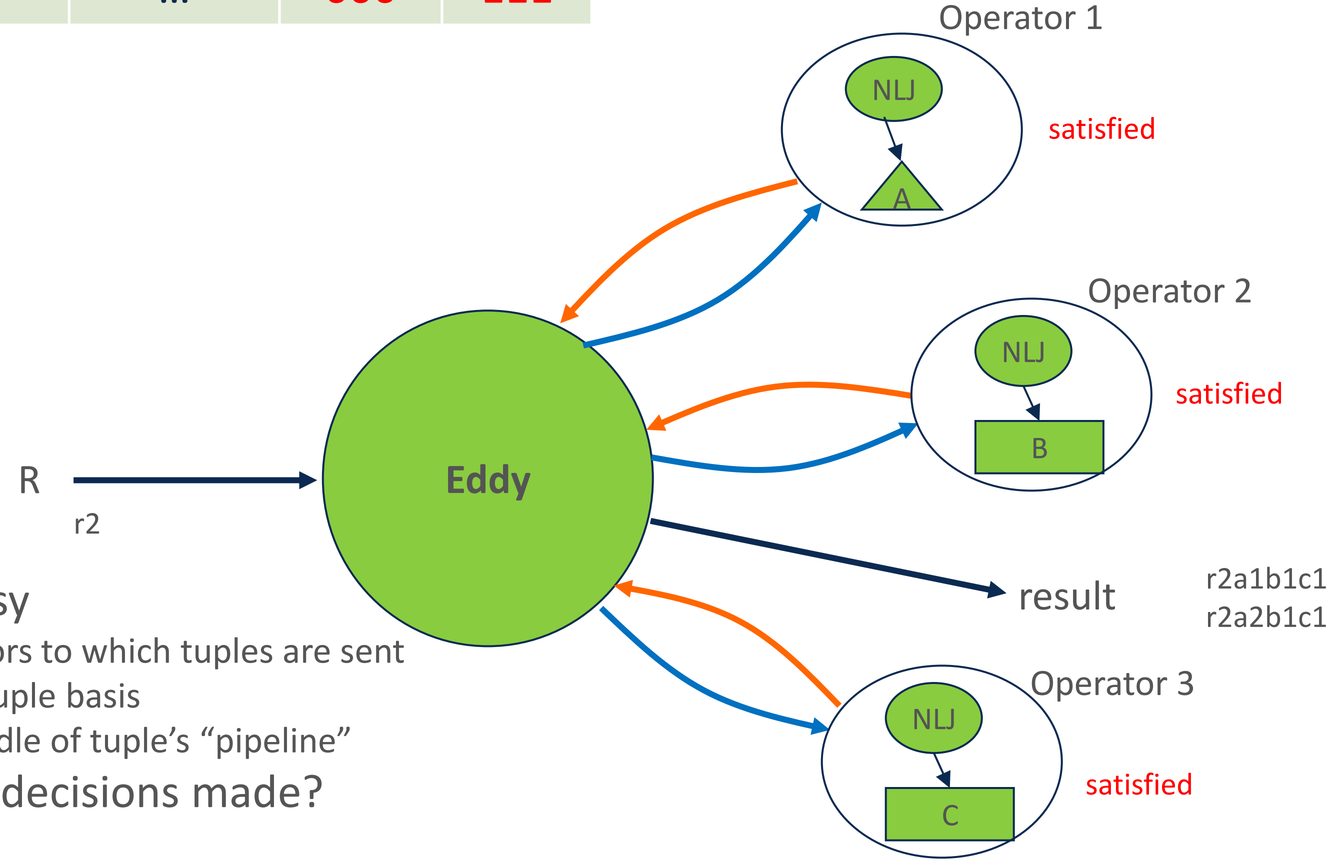
r2a1b1c1

a	b	…	ready	done
15	10	…	000	111

r2a2b1c1

a	b	…	ready	done
15	10	…	000	111

if done == 111,
send to output



Discussion

- Adapting order is easy
  - Just change the operators to which tuples are sent
  - Can be done on a per-tuple basis
  - Can be done in the middle of tuple's "pipeline"
- How are the routing decisions made?
  - Using a routing policy

> Routing Policies

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

Deterministic

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

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

Lottery scheduling

- Each operator runs in thread with an input queue
- “Tickets” assigned according to tuples input / output
- Route tuple to next eligible operator with room in queue, based on number of “tickets” and “backpressure”

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

Content-based routing

- **Different routes for different plans based on attribute values**

Lesson Learned

- Eddies – query processing as routing of tuples
- Self-Tuning Query Mesh

[Rimma V. Nehme, Elke A. Rundensteiner, Elisa Bertino: Self-tuning query mesh for adaptive multi-route query processing. EDBT 2009]

Advantages

- Query execution, statistic monitoring and optimization combined in one operator
- Lowest granularity of adaptation (high adaptation sensibility)
- Simple operator reordering by routing policies
- Execution model inherently enables load balancing

Disadvantages

- Routing overhead (ready and done states)
- Additional overhead for test tuples of alternative paths
- Eddy operator is the serial fraction of whole query plan (restricts the speedup according to Amdahl‘s law)

Summary and Conclusions

> Summary

Problems

- Changing workload characteristics
- Unknown/uncertain statistics (external systems or correlation)

➔ **Adaptive Query Processing**

Summary

- How to classify approaches of adaptive query processing
- Adaptive Query Processing (in action)
  - Plan-Based Adaptation
  - Continous Query Adaptation
  - Routing-Based Adaptation

[Amol Deshpande, Zachary G. Ives, Vijayshankar Raman: Adaptive Query Processing. Foundations and Trends in Databases (FTDB) 1(1):1-140 (2007)]

Conclusions

- Trade-off Risk (Runtime Overhead) and Opportunity (Performance Improvement)
- Specific application areas require tailor-made re-optimization approaches
- Many sophisticated approaches for long running queries
- Lots of open research issues still remain unsolved

> Summary (2)

Database Technology Group

(Selected) Complementary Research Directions to AQP

- Separation: AQP searches for the OPTIMAL plan at different granularities

Plan Robustness (insensitiv to input statistics)

- [M. Abhirama, Sourjya Bhaumik, Atreyee Dey, Harsh Shrimal, Jayant R. Haritsa: On the Stability of Plan Costs and the Costs of Plan Stability. PVLDB 2010]
- [Harish D., Pooja N. Darera, Jayant R. Haritsa: Identifying robust plans through plan diagram reduction. PVLDB 2008]

Online Design Tuning and Database Cracking (storage level)

- [Stratos Idreos, Martin L. Kersten, Stefan Manegold: Self-organizing tuple reconstruction in column-stores. SIGMOD 2009]
- [Stratos Idreos, Martin L. Kersten, Stefan Manegold: Updating a cracked database. SIGMOD 2007]
- [Stratos Idreos, Martin L. Kersten, Stefan Manegold: Database Cracking. CIDR 2007]
- [Martin Lühring, Kai-Uwe Sattler, Eike Schallehn, Karsten Schmidt: Autonomes Index Tuning - DBMS-integrierte Verwaltung von Soft Indexen. BTW 2007]

Multi Query Optimization and Scan Sharing

- [Subi Arumugam, Alin Dobra, Christopher M. Jermaine, Niketan Pansare, Luis Leopoldo Perez: The DataPath system: a data-centric analytic processing engine for large data warehouses. SIGMOD 2010]
- [Philipp Unterbrunner, Georgios Giannikis, Gustavo Alonso, Dietmar Fauser, Donald Kossmann: Predictable Performance for Unpredictable Workloads. PVLDB 2009]
- [Vijayshankar Raman, Garret Swart, Lin Qiao, Frederick Reiss, Vijay Dialani, Donald Kossmann, Inderpal Narang, Richard Sidle: Constant-Time Query Processing. ICDE 2008]
- [Yu Cao, Gopal C. Das, Chee Yong Chan, Kian-Lee Tan: Optimizing complex queries with multiple relation instances. SIGMOD 2008]