Lecture #17

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Optimizer Implementation (Part 2)

Andy Pavlo // 15-721 // Spring 2023

ADMINISTRIVIA

Project #2:

→ Feedback Submission: **Saturday April 1st**
→ Final Submission: **Monday May 1st**

Project #3

→ Status Update Presentation: **Wednesday April 5th**
→ Final Presentations: **Friday May 5th @ 5:30pm**
→ Please email me if you want to discuss your project!

QUERY OPTIMIZATION

For a given query, find a **correct** execution plan that has the lowest "cost".

This is the part of a DBMS that is the hardest to implement well (proven to be NP-Complete).

No optimizer truly produces the "optimal" plan
→ Use estimation techniques to guess real plan cost.
→ Use heuristics to limit the search space.

QUERY OPTIMIZATION STRATEGIES

Choice #1: Heuristics
→ INGRES, Oracle (until mid 1990s)

Choice #2: Heuristics + Cost-based Join Search
→ System R, early IBM DB2, most open-source DBMSs

Choice #3: Randomized Search
→ Academics in the 1980s, current Postgres

Choice #4: Stratified Search
→ IBM's STARBURST (late 1980s), now IBM DB2 + Oracle

Choice #5: Unified Search
→ Volcano/Cascades in 1990s, now MSSQL + Greenplum

STRATIFIED SEARCH

First rewrite the logical query plan using transformation rules.

→ The engine checks whether the transformation is allowed before it can be applied.

→ Cost is never considered in this step.

Then perform a cost-based search to map the logical plan to a physical plan.

UNIFIED SEARCH

Unify the notion of both logical→logical and logical→physical transformations.

→ No need for separate stages because everything is transformations.

This approach generates many transformations, so it makes heavy use of memoization to reduce redundant work.

TOP-DOWN VS. BOTTOM-UP

Top-down Optimization

→ Start with the outcome that the query wants, and then work down the tree to find the optimal plan that gets you to that goal.
→ **Examples**: Volcano, Cascades

Bottom-up Optimization

→ Start with nothing and then build up the plan to get to the outcome that you want.
→ **Examples**: System R, Starburst

TODAY'S AGENDA

Logical Query Optimization

Cascades

Real-World Implementations

LOGICAL QUERY OPTIMIZATION

Transform a logical plan into an equivalent logical plan using pattern matching rules.

The goal is to increase the likelihood of enumerating the optimal plan in the search.

Cannot compare plans because there is no cost model but can "direct" a transformation to a preferred side.

LOGICAL QUERY OPTIMIZATION

Split Conjunctive Predicates

Predicate Pushdown

Replace Cartesian Products with Joins

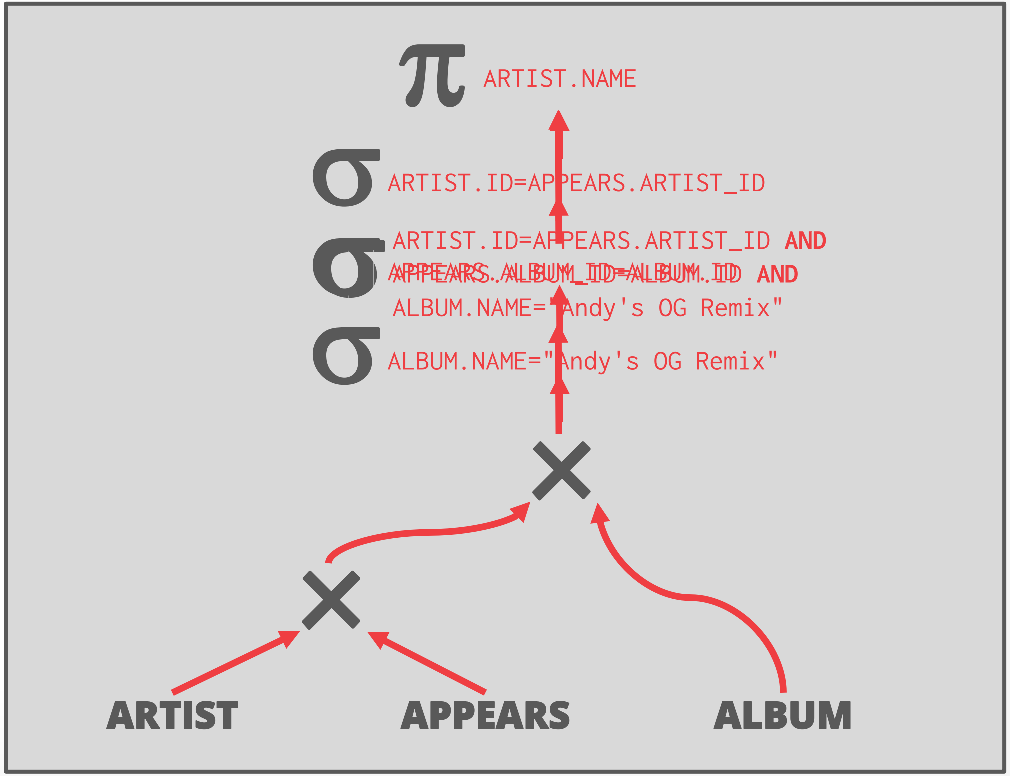
Projection Pushdown

Source: Thomas Neumann

SPLIT CONJUNCTIVE PREDICATES

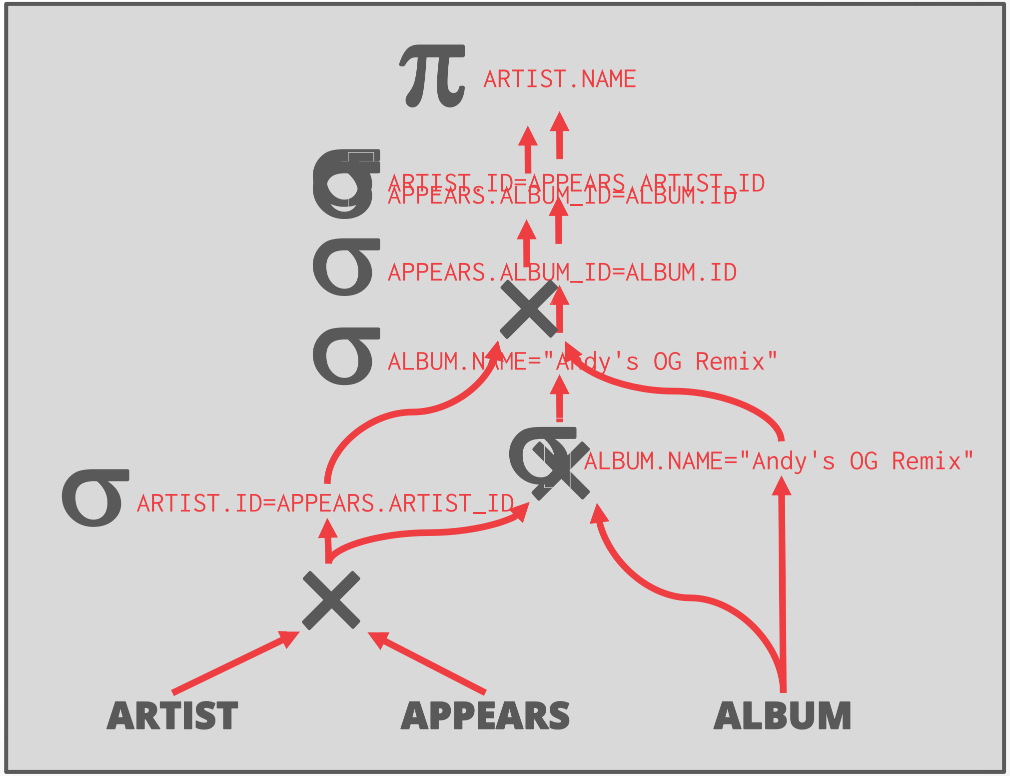
```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Andy's OG Remix"
```

Decompose predicates into their simplest forms to make it easier for the optimizer to move them around.



PREDICATE PUSHDOWN

```sql
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Andy's OG Remix"
```

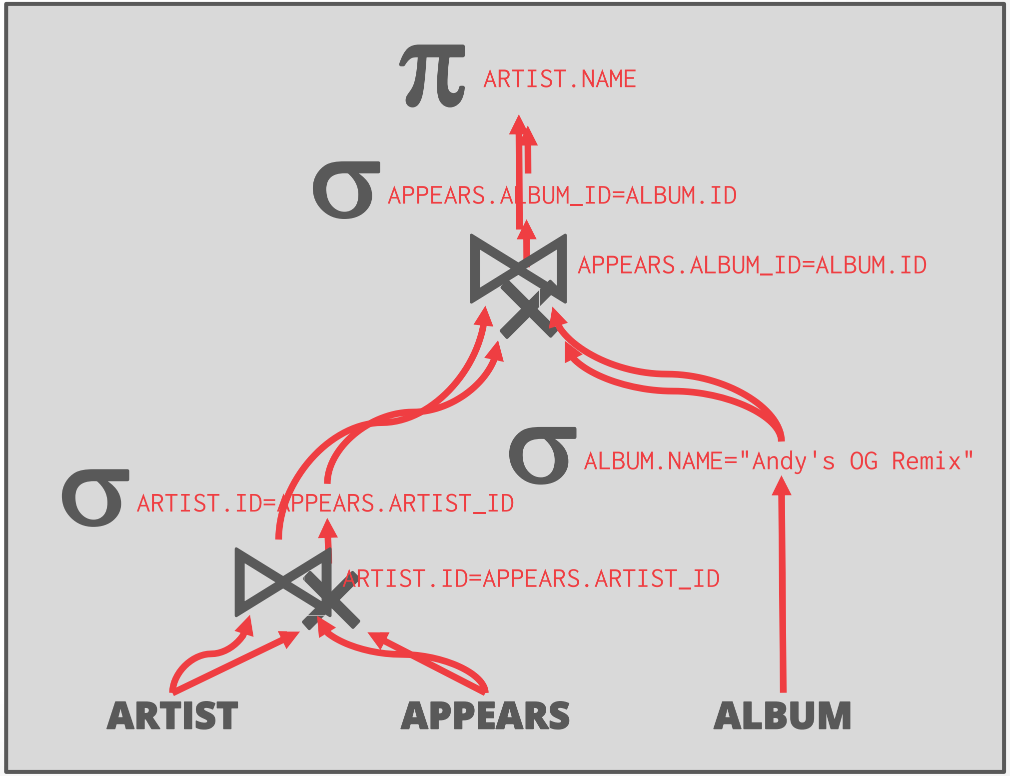


Move the predicate to the lowest point in the plan after Cartesian products.

REPLACE CARTESIAN PRODUCTS

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Andy's OG Remix"
```

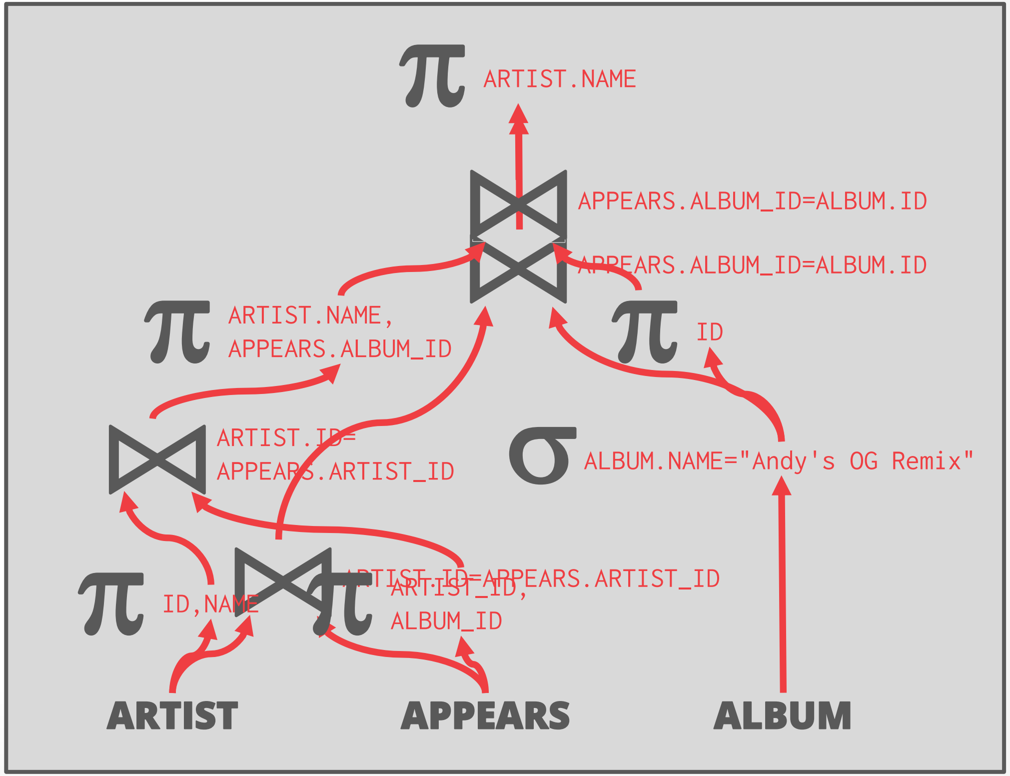
Replace all Cartesian Products with inner joins using the join predicates.



PROJECTION PUSHDOWN

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Andy's OG Remix"
```

Eliminate redundant attributes before pipeline breakers to reduce materialization cost.



PHYSICAL QUERY OPTIMIZATION

Transform a query plan's logical operators into physical operators.

→ Add more execution information
→ Select indexes / access paths
→ Choose operator implementations
→ Choose when to materialize (i.e., temp tables).

This stage must support cost model estimates.

OBSERVATION

All the queries we have looked at so far have had the following properties:

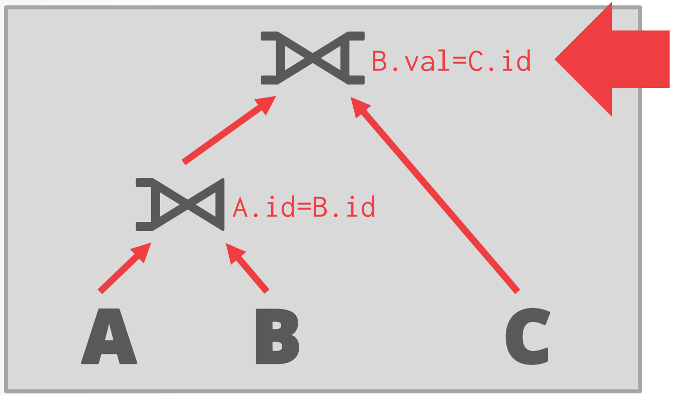
→ Equi/Inner Joins
→ Simple join predicates that reference only two tables.
→ No cross products

Real-world queries are more complex / nasty:

→ Outer Joins
→ Semi Joins
→ Anti Joins
→ Lateral Joins

REORDERING LIMITATIONS

```
SELECT * FROM A
  LEFT OUTER JOIN B
    ON A.id = B.id
  FULL OUTER JOIN C
    ON B.val = C.id;
```



No valid reordering is possible!

The A⟕B operator is not commutative with B⟗C.

→ The DBMS does not know the value of **B.val** until after computing the join with **A**.

Source: Pit Fender

PLAN ENUMERATION

Approach #1: Transformation

→ Modify some part of an existing query plan to transform it into an alternative plan that is equivalent.
→ Top-Down Search

Approach #2: Generative

→ Iteratively assemble and add building blocks to generate a query plan.
→ Bottom-Up Search

ON THE CORRECT AND COMPLETE ENUMERATION OF THE CORE SEARCH SPACE
SIGMOD 2013

DYNAMIC PROGRAMMING OPTIMIZER

Model the query as a hypergraph and then incrementally expand to enumerate new plans.

Algorithm Overview:

→ Iterate connected sub-graphs and incrementally add new edges to other nodes to complete query plan.

→ Use rules to determine which nodes the traversal is allowed to visit and expand.

DYNAMIC PROGRAMMING STRIKES BACK
SIGMOD 2008

CASCADES OPTIMIZER

Object-oriented implementation of the Volcano query optimizer.

→ **Top-down approach** (backward chaining) using branch-and-bound search.

Supports simplistic expression re-writing through a direct mapping function rather than an exhaustive search.

Graefe

THE CASCADES FRAMEWORK FOR QUERY OPTIMIZATION
IEEE DATA ENGINEERING BULLETIN 1995

EFFICIENCY IN THE COLUMBIA DATABASE QUERY OPTIMIZER
PORTLAND STATE UNIVERSITY MS THESIS 1998

CASCADES OPTIMIZER

Optimization tasks as data structures.

Rules to place property enforcers.

Ordering of moves by promise.

Predicates as logical/physical operators.

EFFICIENCY IN THE COLUMBIA DATABASE QUERY OPTIMIZER
PORTLAND STATE UNIVERSITY MS THESIS 1998

CASCADES – EXPRESSIONS

An **expression** represents some operation in the query with zero or more input expressions.

→ Optimizer needs to be able to quickly determine whether two expressions are equivalent.

```
SELECT * FROM A
  JOIN B ON A.id = B.id
  JOIN C ON C.id = A.id;
```

Logical Expression: $(A \bowtie B) \bowtie C$

Physical Expression: $(A_{Seq} \bowtie_{HJ} B_{Seq}) \bowtie_{NL} C_{Idx}$

CASCADES – GROUPS

A **group** is a set of logically equivalent logical and physical expressions that produce the same output.

→ All logical forms of an expression.

→ All physical expressions that can be derived from selecting the allowable physical operators for the corresponding logical forms.

Group

Output: [ABC]	Logical Exps	Physical Exps
	1. $(A \bowtie B) \bowtie C$	1. $(A_{Seq} \bowtie_{NL} B_{Seq}) \bowtie_{NL} C_{Seq}$
	2. $(B \bowtie C) \bowtie A$	2. $(B_{Seq} \bowtie_{NL} C_{Seq}) \bowtie_{NL} A_{Seq}$
	3. $(A \bowtie C) \bowtie B$	3. $(A_{Seq} \bowtie_{NL} C_{Seq}) \bowtie_{NL} B_{Seq}$
	4. $A \bowtie (B \bowtie C)$	4. $A_{Seq} \bowtie_{NL} (C_{Seq} \bowtie_{NL} B_{Seq})$
	⋮	⋮

Equivalent Expressions

CASCADES – MULTI-EXPRESSION

Instead of explicitly instantiating all possible expressions in a group, the optimizer implicitly represents redundant expressions in a group as a **multi-expression**.

→ This reduces the number of transformations, storage overhead, and repeated cost estimations.

Output: [ABC]	Logical Multi-Exps	Physical Multi-Exps
	1. [AB]⋈[C]	1. [AB]$\bowtie_{SM}$[C]
	2. [BC]⋈[A]	2. [AB]$\bowtie_{HJ}$[C]
	3. [AC]⋈[B]	3. [AB]$\bowtie_{NL}$[C]
	4. [A]⋈[BC]	4. [BC]$\bowtie_{SM}$[A]
	⋮	⋮

CASCADES – RULES

A **rule** is a transformation of an expression to a logically equivalent expression.

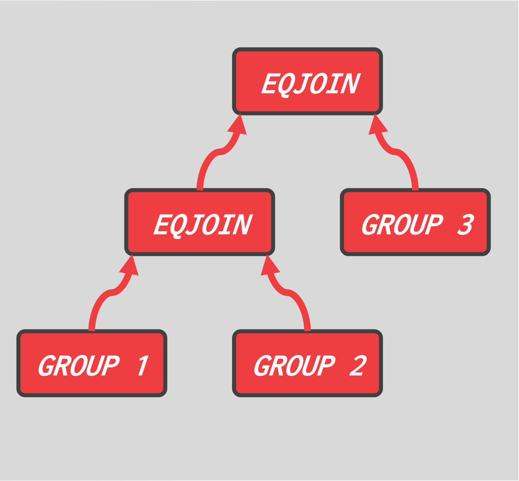
→ **Transformation Rule:** Logical to Logical
→ **Implementation Rule:** Logical to Physical

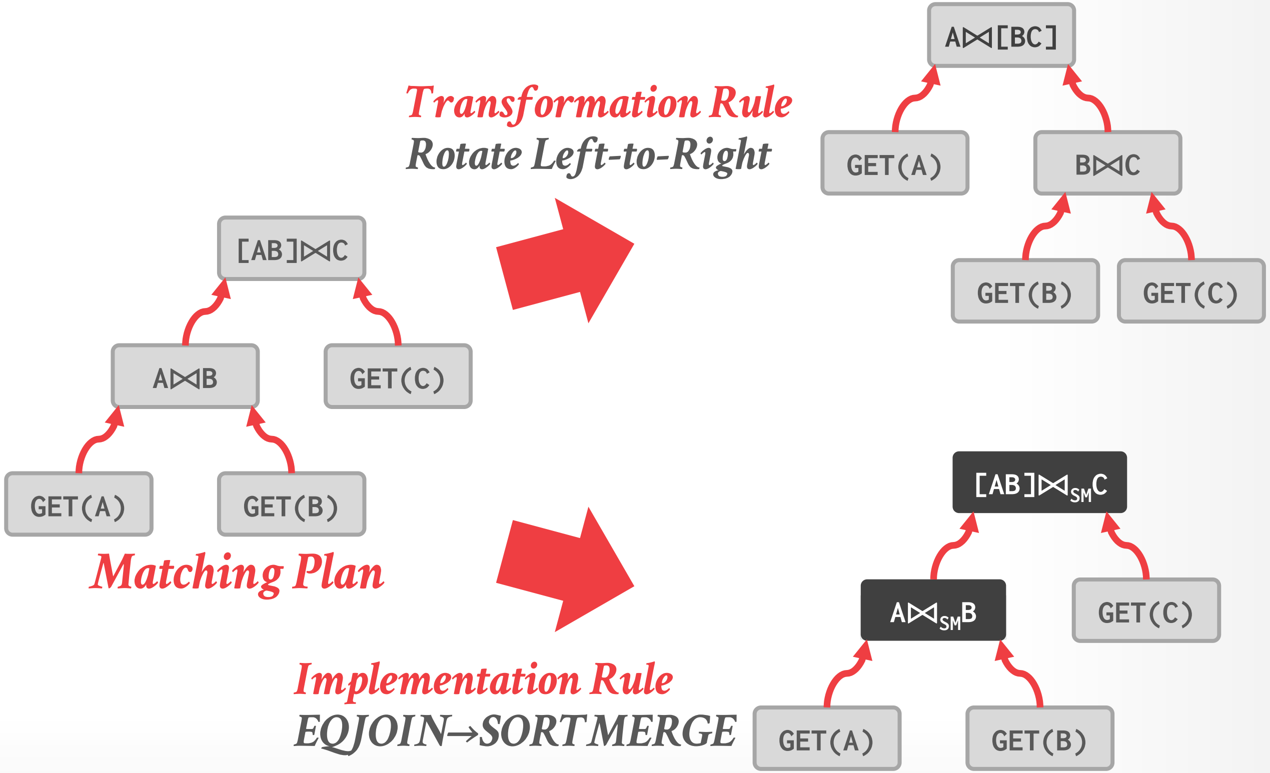
Each rule is represented as a pair of attributes:

→ **Pattern**: Defines the structure of the logical expression that can be applied to the rule.
→ **Substitute**: Defines the structure of the result after applying the rule.

CASCADES – RULES

Pattern





- *Group*
- *Logical Expr*
- *Physical Expr*

CASCADES – MEMO TABLE

Stores all previously explored alternatives in a compact graph structure / hash table.

Equivalent operator trees and their corresponding plans are stored together in groups.

Provides memoization, duplicate detection, and property + cost management.

PRINCIPLE OF OPTIMALITY

Every sub-plan of an optimal plan is itself optimal.

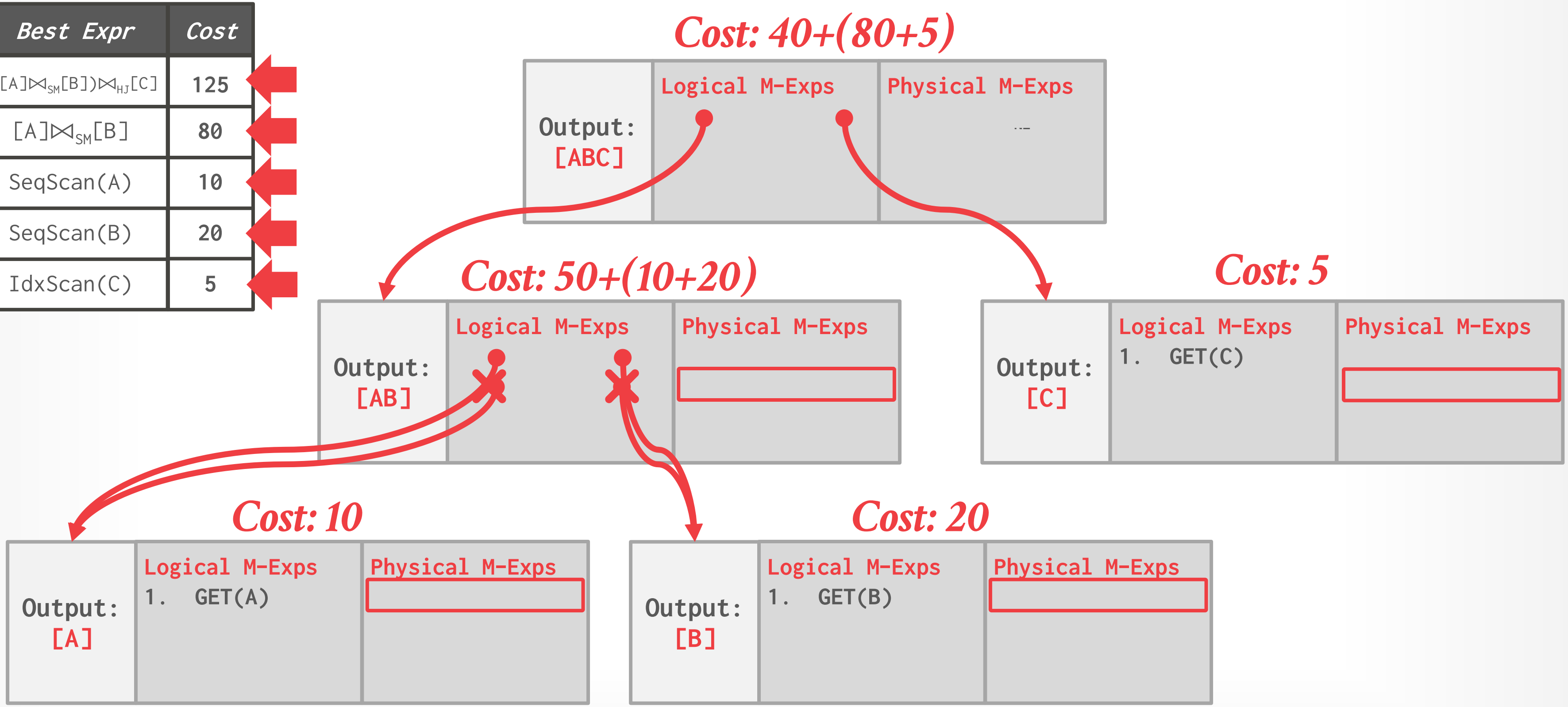
This allows the optimizer to restrict the search space to a smaller set of expressions.

→ The optimizer never has to consider a plan containing sub-plan **P1** that has a greater cost than equivalent plan **P2** with the same physical properties.

EXPLOITING UPPER AND LOWER BOUNDS IN TOP-DOWN QUERY OPTIMIZATION
IDEAS 2001

CASCADES – MEMO TABLE

	Best Expr	*Cost*
[ABC]	$([A]\bowtie_{SM}[B])\bowtie_{HJ}[C]$	125
[AB]	$[A]\bowtie_{SM}[B]$	80
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]	IdxScan(C)	5



SEARCH TERMINATION

Approach #1: Transformation Exhaustion

→ Stop when there are no more ways to transform the target plan. Usually done per group.

Approach #2: Wall-clock Time

→ Stop after the optimizer runs for some length of time.

Approach #3: Transformation Count

→ Stop after a certain number of transformations have been considered.

Approach #4: Cost Threshold

→ Stop when the optimizer finds a plan that has a lower cost than some threshold.

CASCADES IMPLEMENTATIONS

Standalone:

→ Wisconsin OPT++ (1990s)
→ Portland State Columbia (1990s)
→ Greenplum Orca (2010s)
→ Apache Calcite (2010s)

Integrated:

→ Microsoft SQL Server (1990s)
→ Tandem NonStop SQL (1990s)
→ Clustrix (2000s)
→ CockroachDB (2010s)

REAL-WORLD IMPLEMENTATIONS

Microsoft SQL Server

Apache Calcite

Greenplum Orca

CockroachDB

SingleStore

Cascades

MICROSOFT SQL SERVER

First Cascades implementation started in 1995.

- → Derivatives are used in many MSFT database products.
- → All transformations are written in C++. No DSL.
- → Scalar / expression transformations are written in procedural code and not rules.

DBMS applies transformations in multiple stages with increasing scope and complexity.

- → The goal is to leverage domain knowledge to apply transformations that you always want to do first to reduce the search space.

MICROSOFT SQL SERVER

Sub-Query Removal
Outer Joins to Inner Joins
Predicate Pushdown
Empty Result Pruning

Cost-based Search Initialization

Stage1: Trivial Plan
Stage2: Quick Plan (Parallel)
Stage3: Full Plan (Parallel)

Engine-Specific Transformations

Simplification / Normalization → ***Pre-Exploration*** → ***Exploration*** → ***Post-Optimization***

Tree-to-Tree Transformations

Trivial Plan Short-circuit
Projection Normalization
Statistics Identification/Collection
Initial Cardinality Estimates
Join Collapsing

Multi-Stage Cost-Based Search

Source: Nico Bruno + Cesar Galindo-Legaria

MICROSOFT SQL SERVER

Optimization #1: Timeouts are based on the number of transformations not wallclock time.

→ Ensures that overloaded systems do not generate different plans than under normal operations.

Optimization #2: Pre-populate the Memo Table with potentially useful join orderings.

→ Heuristics that consider relationships between tables.

→ Syntactic appearance in query.

APACHE CALCITE

Standalone extensible query optimization framework for data processing systems.

→ Support for pluggable query languages, cost models, and rules.

→ Does not distinguish between logical and physical operators. Physical properties are provided as annotations.

Originally part of LucidDB.

APACHE CALCITE: A FOUNDATIONAL FRAMEWORK FOR OPTIMIZED QUERY PROCESSING OVER HETEROGENEOUS DATA SOURCES
SIGMOD 2018

GREENPLUM ORCA

Standalone Cascades implementation in C++.
→ Originally written for Greenplum.
→ Extended to support HAWQ.

A DBMS integrates Orca by implementing API to send catalog + stats + logical plans and then retrieve physical plans.

Supports multi-threaded search.

ORCA: A MODULAR QUERY OPTIMIZER ARCHITECTURE FOR BIG DATA
SIGMOD 2014

GREENPLUM ORCA – ENGINEERING

Issue #1: Remote Debugging

→ Automatically dump the state of the optimizer (with inputs) whenever an error occurs.

→ The dump is enough to put the optimizer back in the exact same state later for further debugging.

Issue #2: Optimizer Accuracy

→ Automatically check whether the ordering of the estimate cost of two plans matches their actual execution cost.

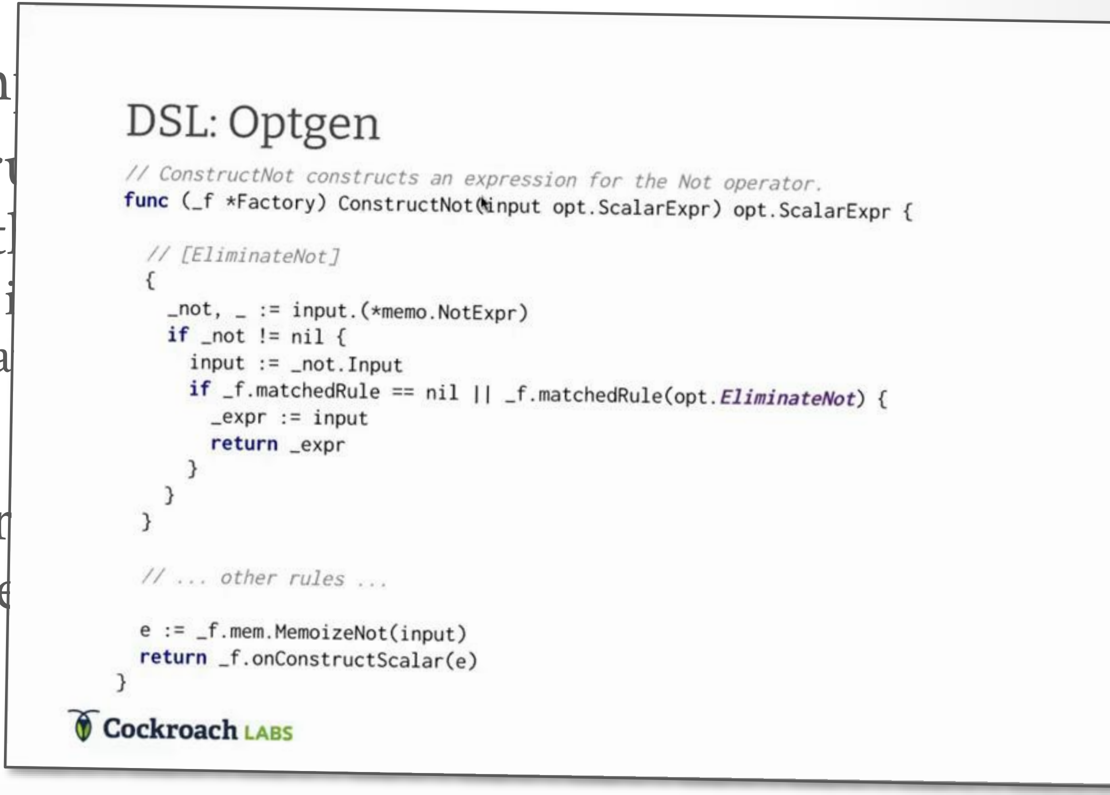
COCKROACHDB

Custom Cascades im

All transformation r DSL (OptGen) and t

→ Can embed Go logic i analysis and modifica

Also considers scalar transformations toge



Source: Rebecca Taft

SINGLESTORE OPTIMIZER

Rewriter

→ Logical-to-logical transformations with access to the cost-model.

Enumerator

→ Logical-to-physical transformations.

→ Mostly join ordering.

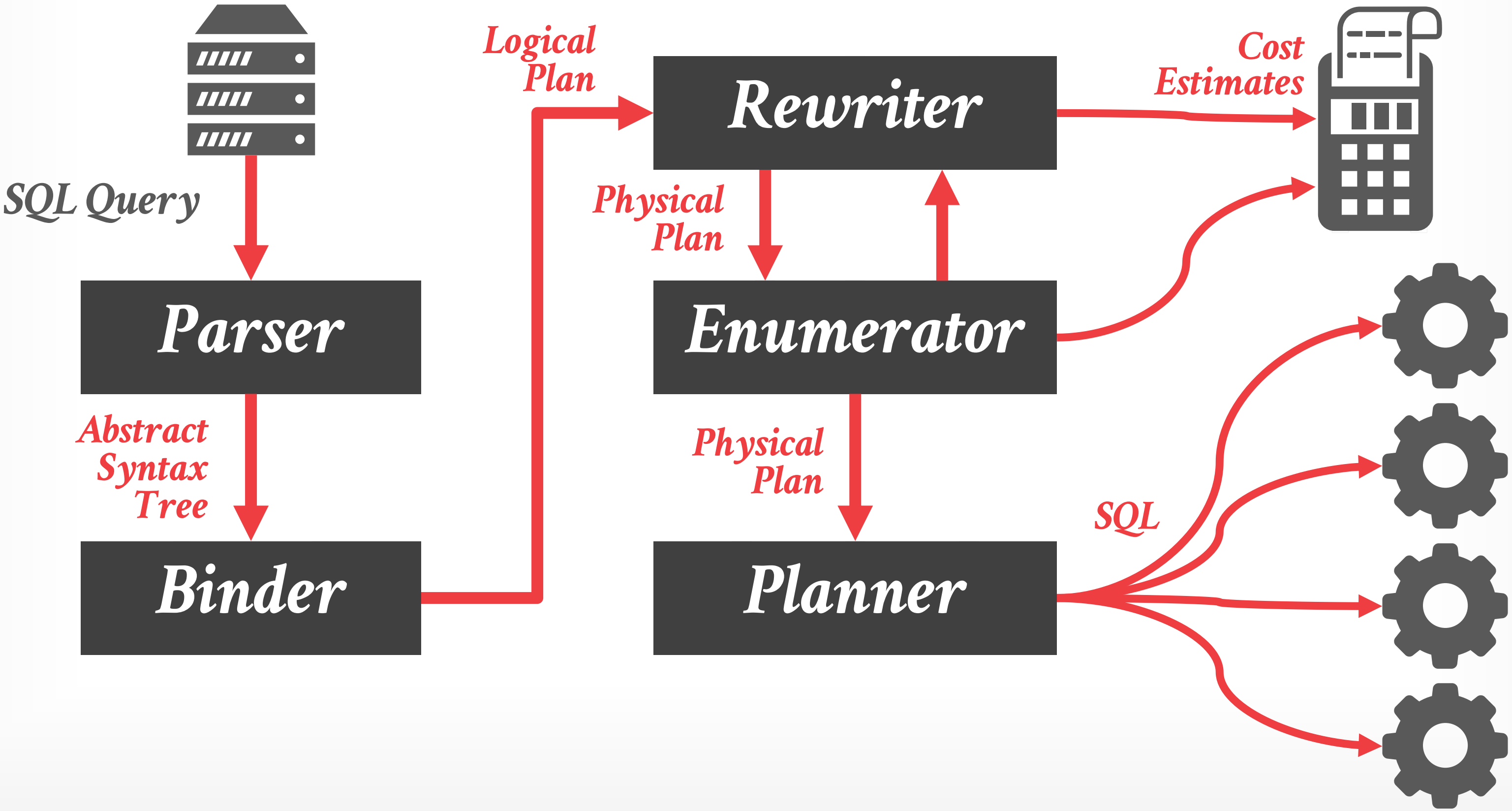
Planner

→ Convert physical plans back to SQL.

→ Contains SingleStore-specific commands for moving data.

THE MEMSQL QUERY OPTIMIZER
VLDB 2017

SINGLESTORE OPTIMIZER



PARTING THOUGHTS

All of this relies on a good **cost model**.
A good **cost model** needs good statistics.

NEXT CLASS

Cost Models