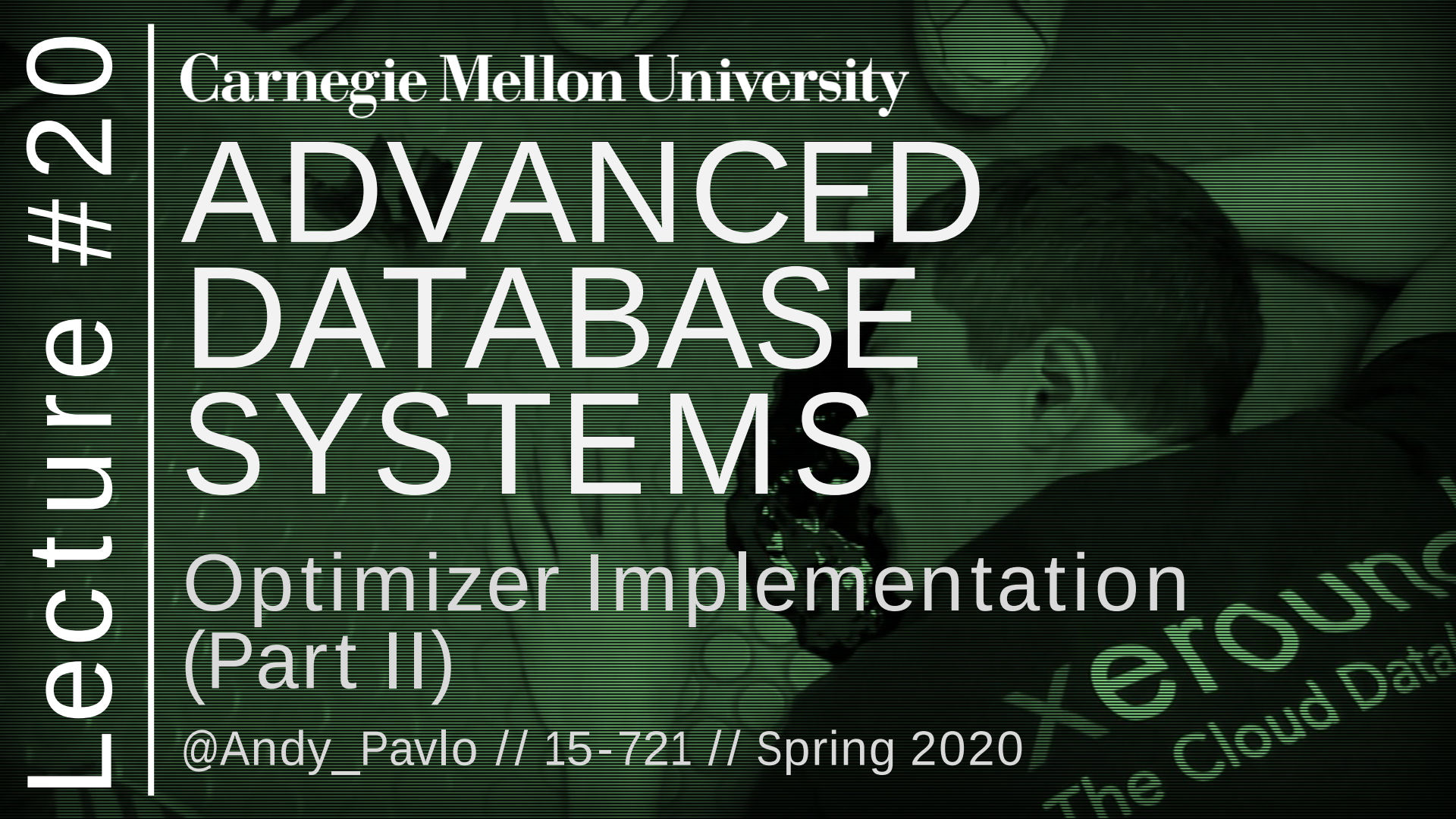
Lecture #20

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Optimizer Implementation (Part II)

@Andy_Pavlo // 15-721 // Spring 2020



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 a lot more transformations so it makes heavy use of memoization to reduce redundant work.

TOP-DOWN VS. BOTTOM-UP

Top-down Optimization

→ Start with the final outcome that you want, and then work down the tree to find the optimal plan that gets you to that goal.

→ Example: Volcano, Cascades

Bottom-up Optimization

→ Start with nothing and then build up the plan to get to the final outcome that you want.

→ Examples: System R, Starburst

TODAY'S AGENDA

Logical Query Optimization
Cascades / Columbia
Dynamic Programming
Other Implementations
Project #3 Code Reviews

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

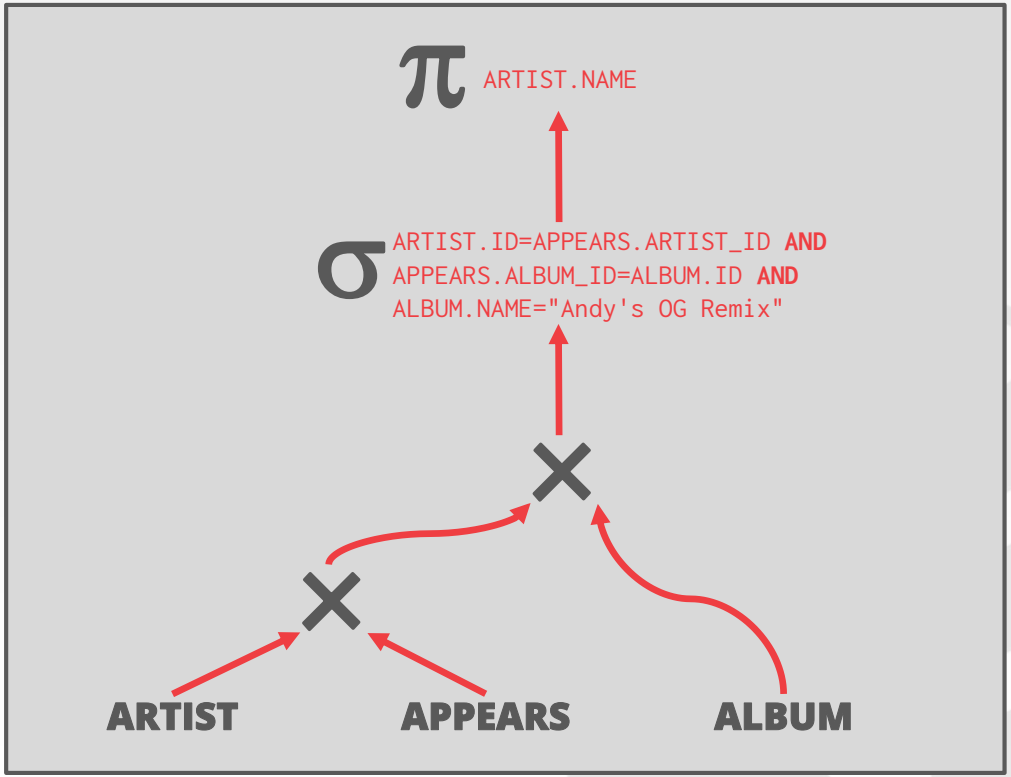
Projection Pushdown

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"
```

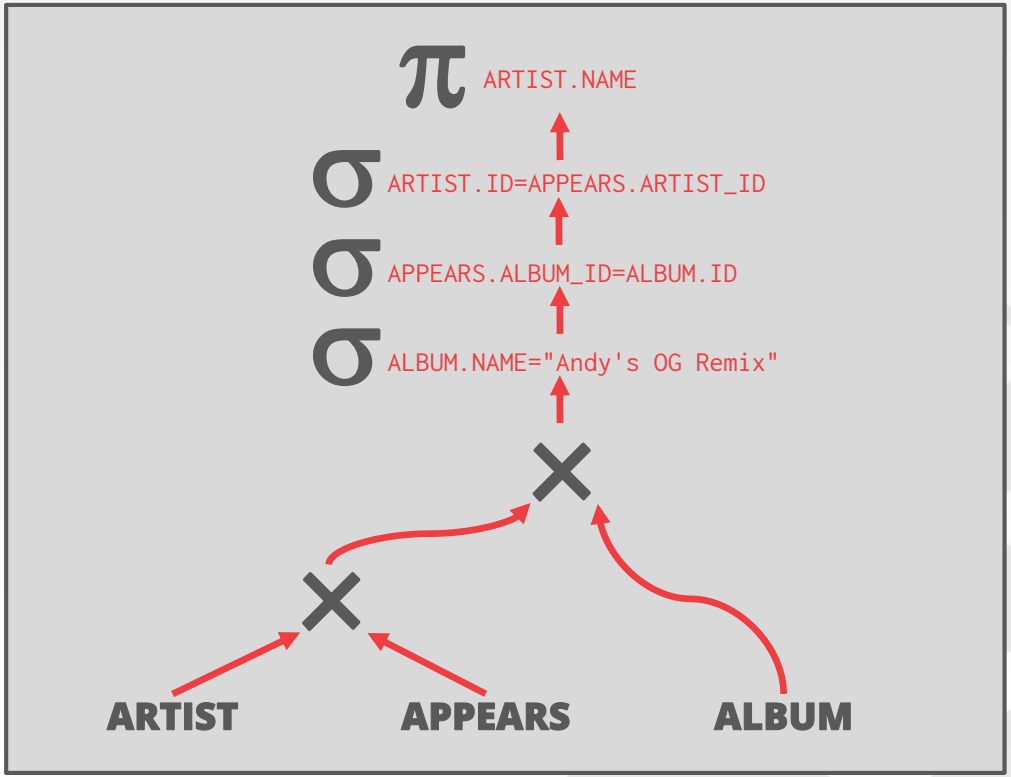
Decompose predicates into their simplest forms to make it easier for the optimizer to move them around.



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"
```

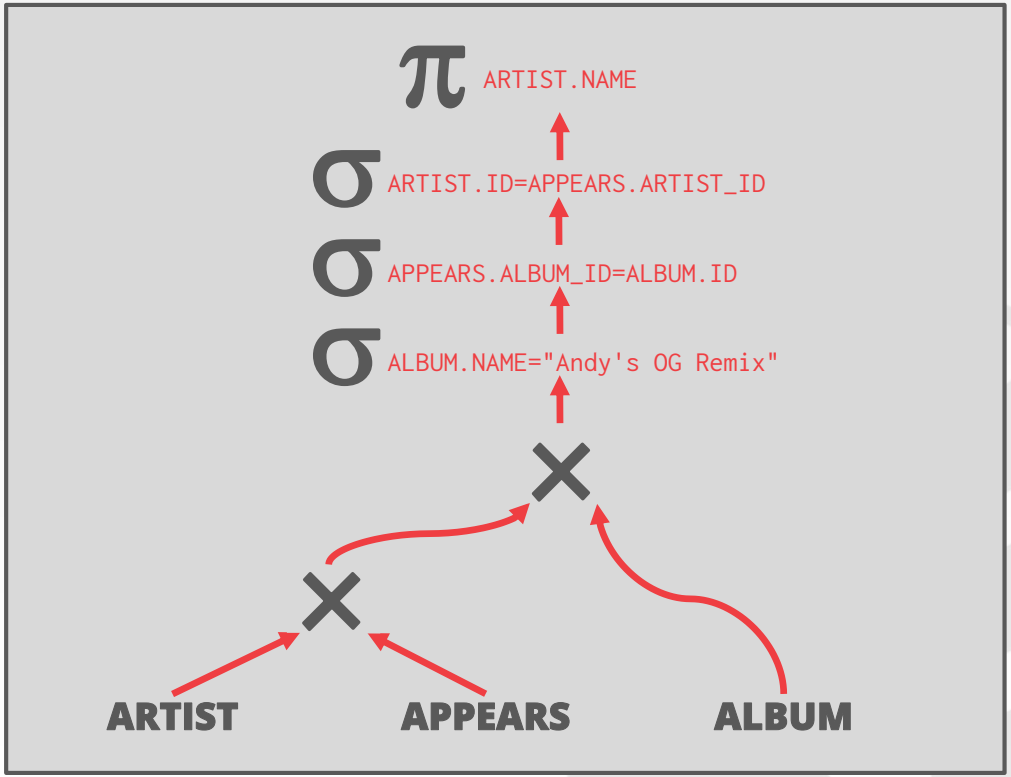
Decompose predicates into their simplest forms to make it easier for the optimizer to move them around.



PREDICATE PUSHDOWN

```
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"
```

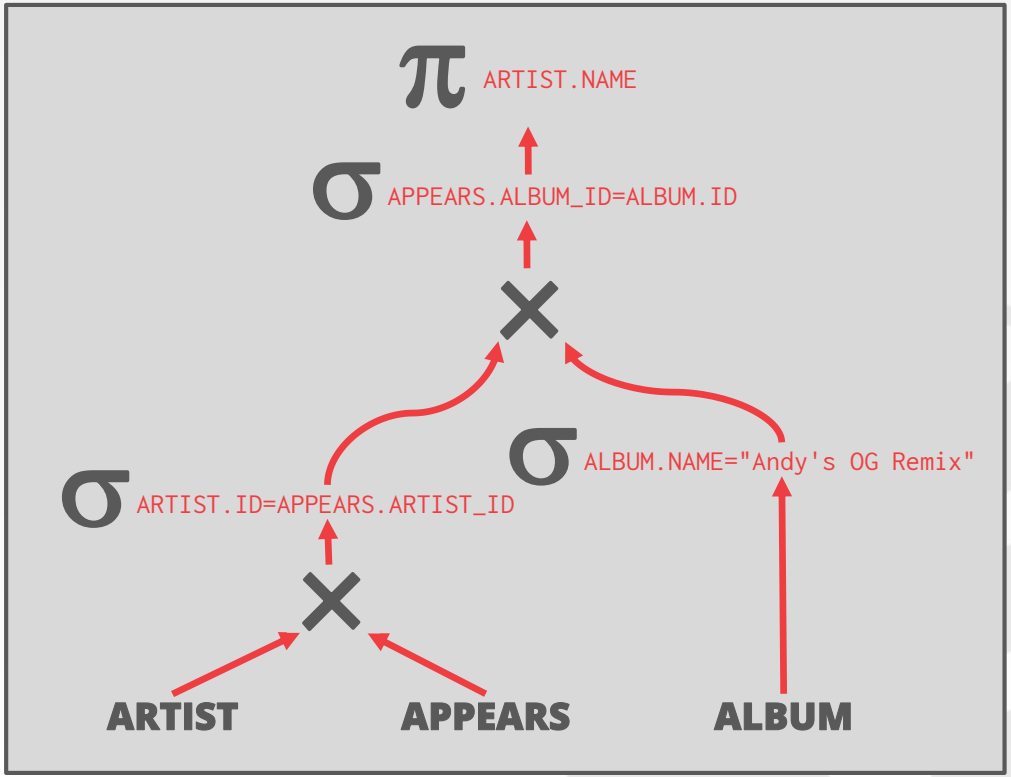
Move the predicate to the lowest point in the plan after Cartesian products.



PREDICATE 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"
```

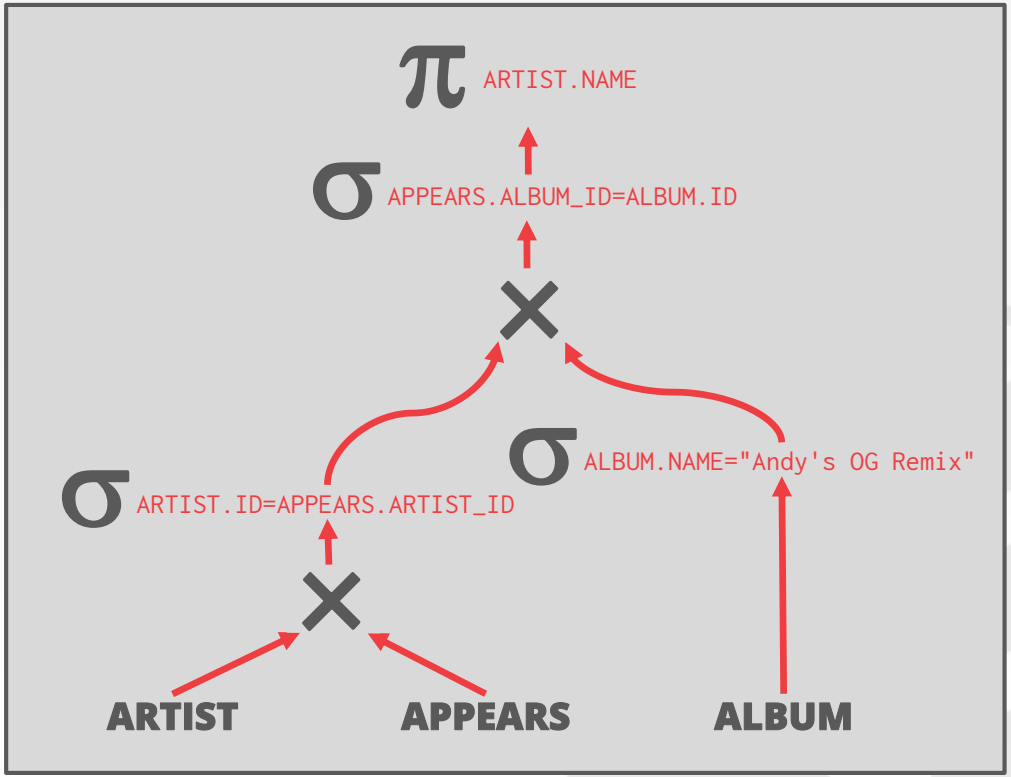
Move the predicate to the lowest point in the plan after Cartesian products.



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"
```

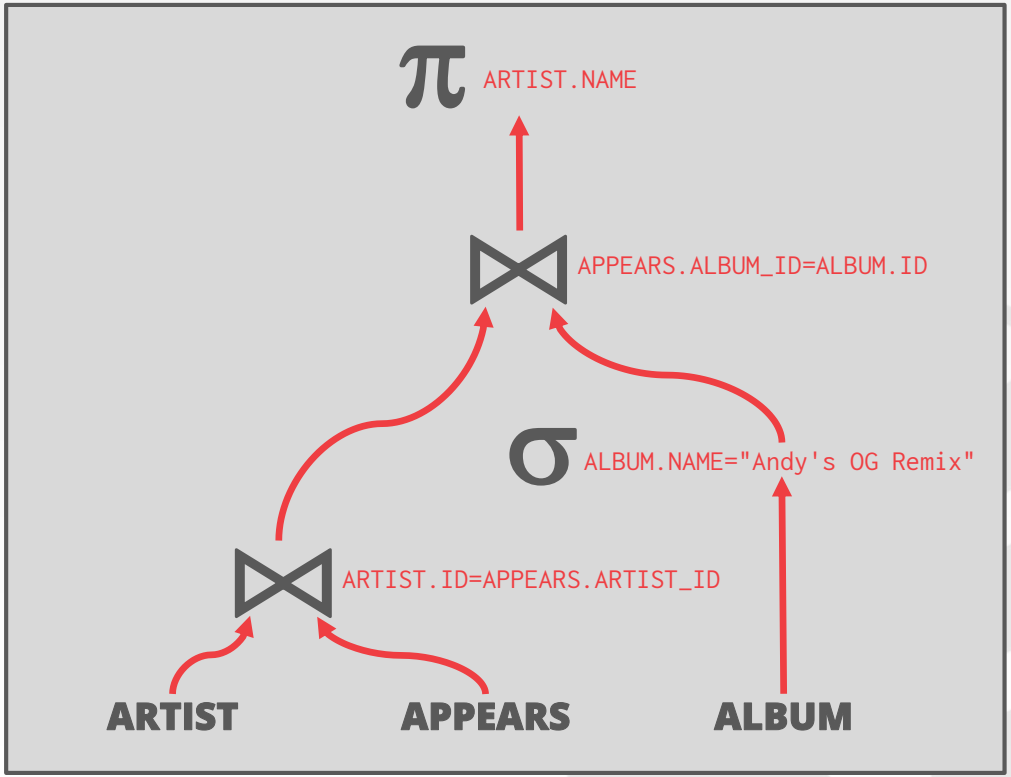
Replace all Cartesian Products with inner joins using the join predicates.



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"
```

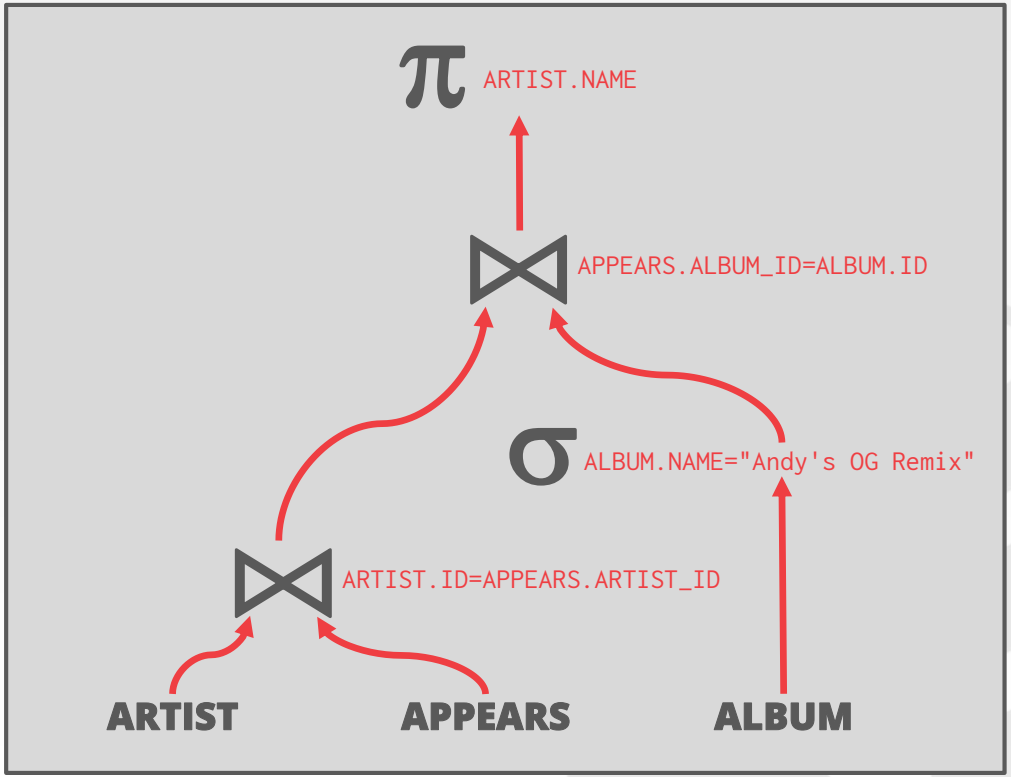
Replace all Cartesian Products with inner joins using the join predicates.



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"
```

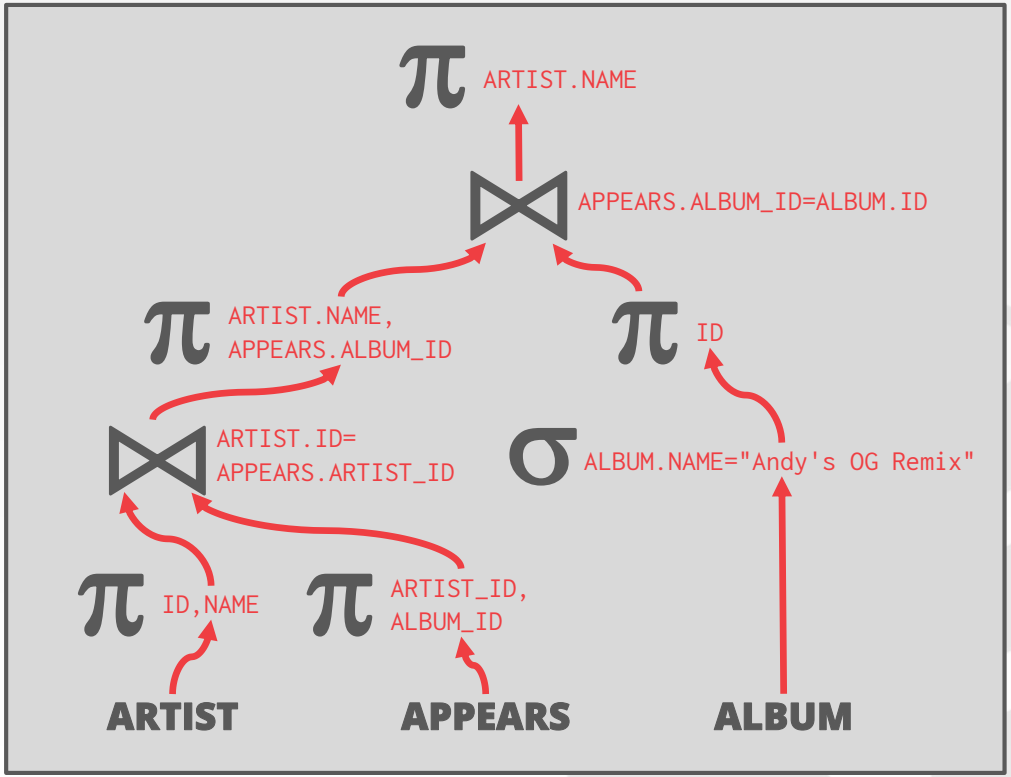
Eliminate redundant attributes before pipeline breakers to reduce materialization cost.



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:

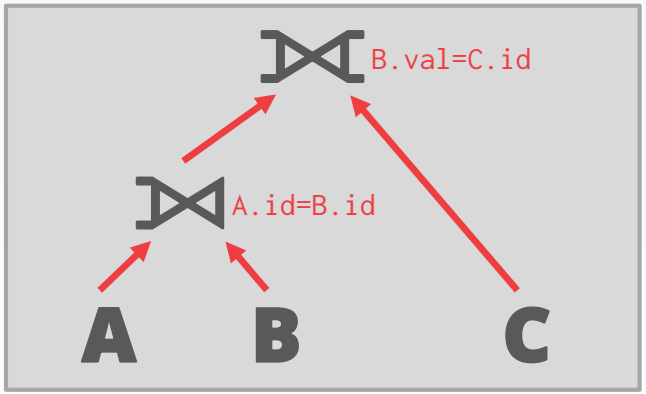
→ Equi/Inner Joins
→ Simple join predicates that reference only two tables.
→ No cross products

Real-world queries are much more complex:

→ Outer Joins
→ Semi-joins
→ Anti-joins

REORDERING LIMITATIONS

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

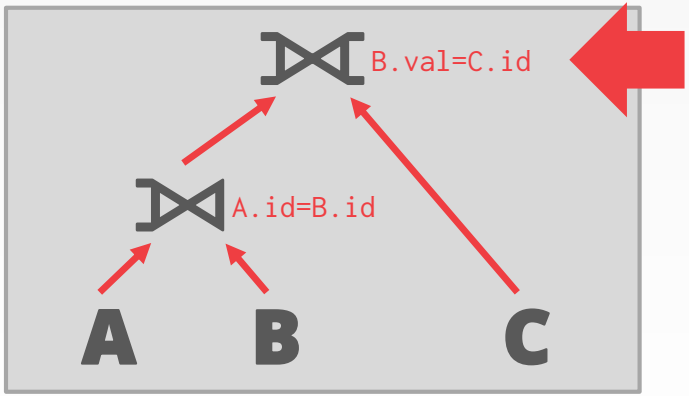


No valid reordering is possible.

Source: Pit Fender

REORDERING LIMITATIONS

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



Source: Pit Fender

No valid reordering is possible.

The $A \bowtie B$ operator is not commutative with $B \bowtie C$.

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

PLAN ENUMERATION

Approach #1: Transformation

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

Approach #2: Generative

→ Assemble building blocks to generate a query plan.

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.

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

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** is an operator with zero or more input expressions.

```
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.

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. $A_{Seq} \bowtie_{NL} (C_{Seq} \bowtie_{NL} B_{Seq})$
	⋮	⋮

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.

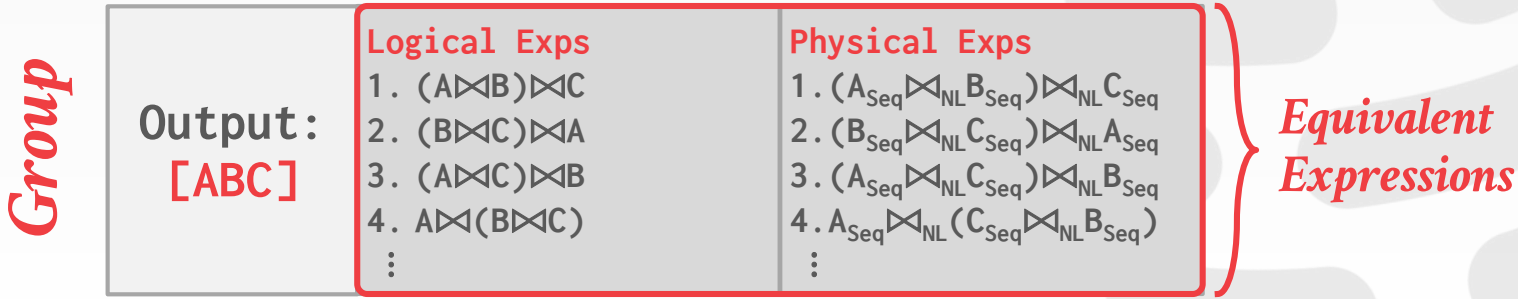
→ All physical expressions that can be derived from selecting the allowable physical operators for the corresponding logical forms.

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.



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

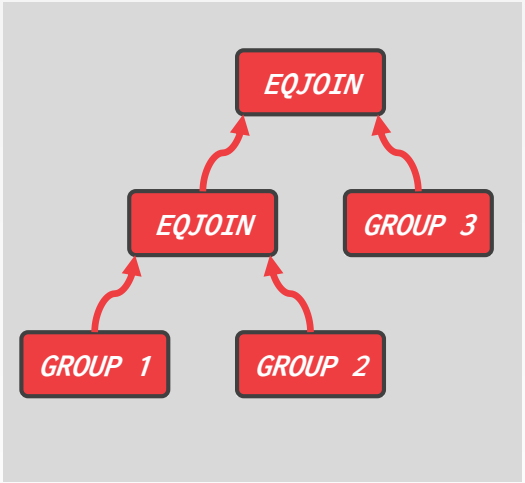
Each rule is represented as a pair of attributes:

→ **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

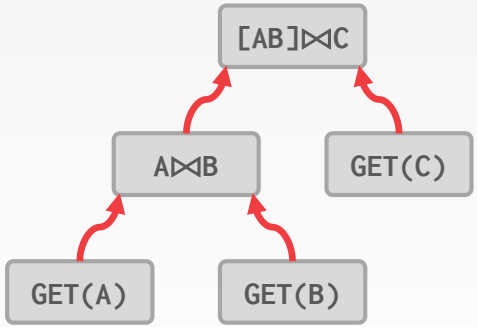
Pattern



Group

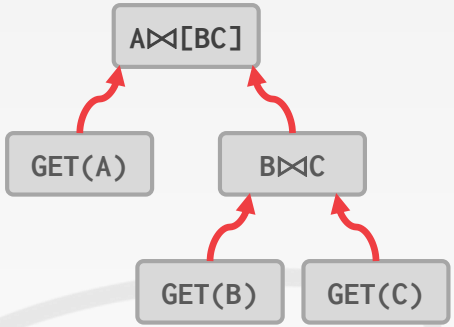
Logical Expr

Physical Expr

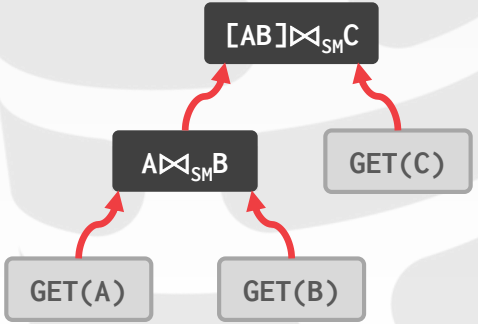


Matching Plan

Transformation Rule
Rotate Left-to-Right



Implementation Rule
EQJOIN→SORTMERGE



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

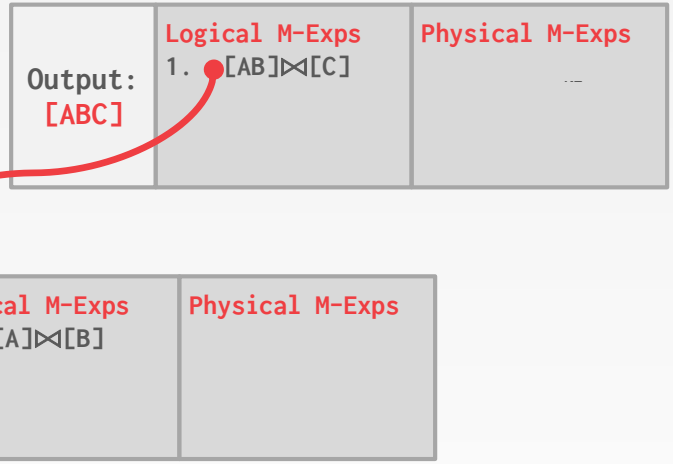
CASCADES – MEMO TABLE

	Best Expr	*Cost*
[ABC]		
[AB]		
[A]		
[B]		
[C]		

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

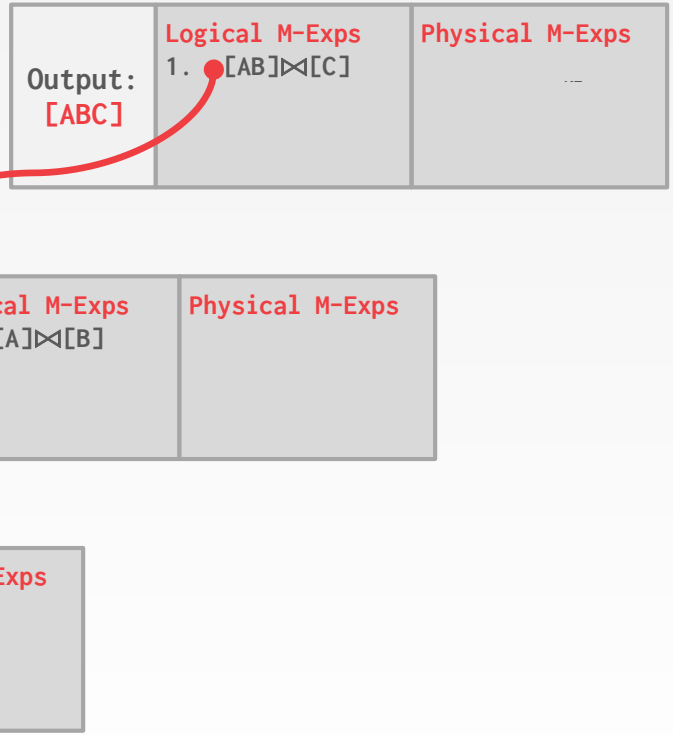
CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]		
[A]		
[B]		
[C]		



CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]		
[A]		
[B]		
[C]		



CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]		
[A]		
[B]		
[C]		

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A) 2. IdxScan(A)

CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]		
[A]	SeqScan(A)	10
[B]		
[C]		

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

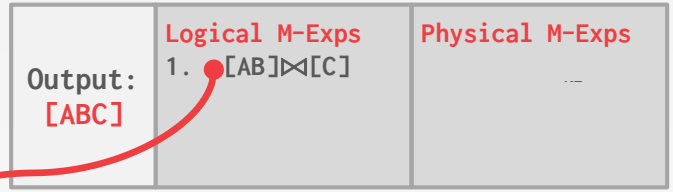
Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	

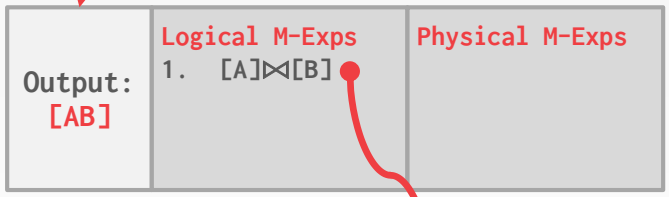
Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A) 2. IdxScan(A)

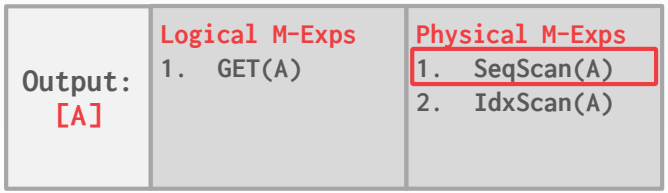
CASCADES – MEMO TABLE

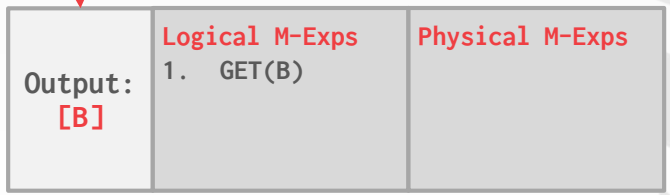
	Best Expr	Cost
[ABC]		
[AB]		
[A]	SeqScan(A)	10
[B]		
[C]		





Cost: 10





CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]		
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]		

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A) 2. IdxScan(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. SeqScan(B) 2. IdxScan(B)

CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]		
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]		

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	

Cost: 10

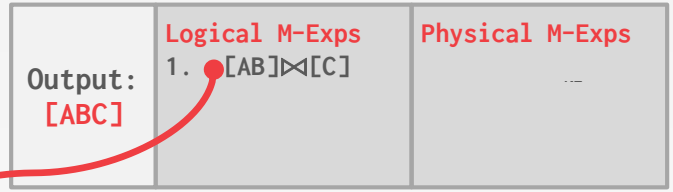
Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A) 2. IdxScan(A)

Cost: 20

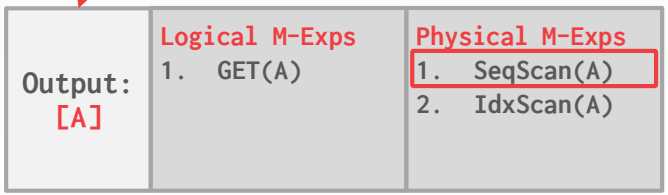
Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. SeqScan(B) 2. IdxScan(B)

CASCADES – MEMO TABLE

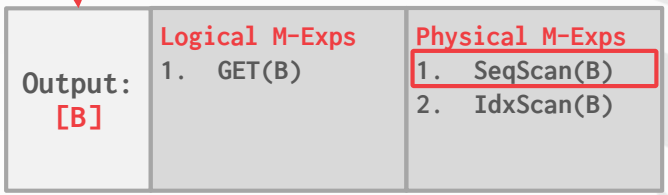
	Best Expr	Cost
[ABC]		
[AB]		
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]		



Cost: 10



Cost: 20



CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]		
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]		

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B] 2. ~~[B]⋈[A]~~	

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A) 2. IdxScan(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. SeqScan(B) 2. IdxScan(B)

CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]		
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]		

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B] 2. [B]⋈[A]	1. [A]$\bowtie_{NL}$[B] 2. [A]$\bowtie_{SM}$[B] 3. [B]$\bowtie_{NL}$[A]

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A) 2. IdxScan(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. SeqScan(B) 2. IdxScan(B)

CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]	[A]⋈$_{SM}$[B]	80
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]		

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Cost: 50+(10+20)

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B] 2. [B]⋈[A]	1. [A]⋈$_{NL}$[B] 2. [A]⋈$_{SM}$[B] 3. [B]⋈$_{NL}$[A]

Cost: 10

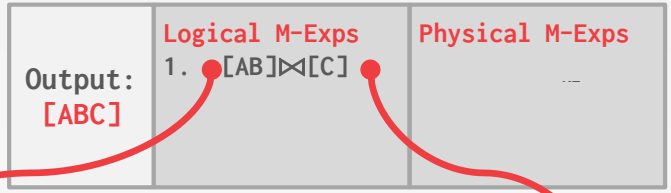
Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A) 2. IdxScan(A)

Cost: 20

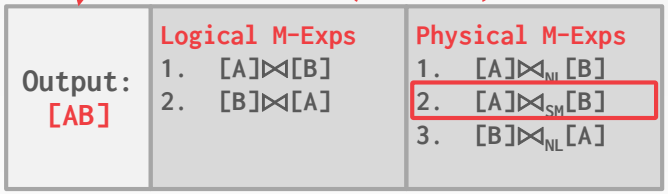
Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. SeqScan(B) 2. IdxScan(B)

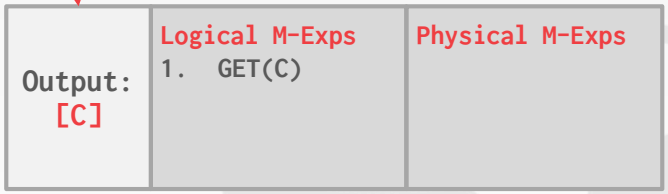
CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]	[A]⋈$_{SM}$[B]	80
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]		

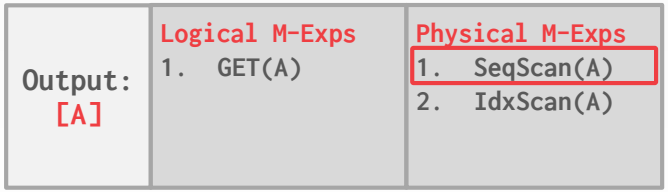


Cost: 50+(10+20)

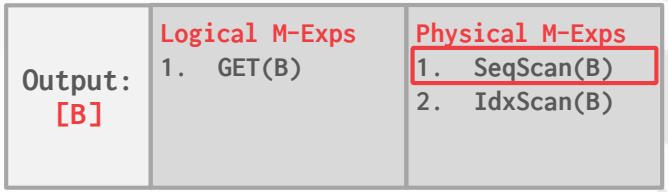




Cost: 10



Cost: 20



CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]	[A]⋈$_{SM}$[B]	80
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]	IdxScan(C)	5

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Cost: 50+(10+20)

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B] 2. [B]⋈[A]	1. [A]⋈$_{NL}$[B] 2. [A]⋈$_{SM}$[B] 3. [B]⋈$_{NL}$[A]

Cost: 5

Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	1. SeqScan(C) 2. IdxScan(C)

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A) 2. IdxScan(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. SeqScan(B) 2. IdxScan(B)

CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]	[A]⋈$_{SM}$[B]	80
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]	IdxScan(C)	5

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	

Cost: 50+(10+20)

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B] 2. [B]⋈[A]	1. [A]⋈$_{NL}$[B] 2. [A]⋈$_{SM}$[B] 3. [B]⋈$_{NL}$[A]

Cost: 5

Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	1. SeqScan(C) 2. IdxScan(C)

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A) 2. IdxScan(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. SeqScan(B) 2. IdxScan(B)

CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]	[A]⋈$_{SM}$[B]	80
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]	IdxScan(C)	5

Output: [ABC]	Logical M-Exps	Physical M-Exps
	1. [AB]⋈[C]	1. [AB]⋈$_{NL}$C
	2. [BC]⋈[A]	2. [BC]⋈$_{NL}$A
	3. [AC]⋈[B]	3. [AC]⋈$_{NL}$B
	4. [B]⋈[AC]	⋮

Cost: 50+(10+20)

Output: [AB]	Logical M-Exps	Physical M-Exps
	1. [A]⋈[B]	1. [A]⋈$_{NL}$[B]
	2. [B]⋈[A]	2. [A]⋈$_{SM}$[B]
		3. [B]⋈$_{NL}$[A]

Cost: 5

Output: [C]	Logical M-Exps	Physical M-Exps
	1. GET(C)	1. SeqScan(C)
		2. IdxScan(C)

Cost: 10

Output: [A]	Logical M-Exps	Physical M-Exps
	1. GET(A)	1. SeqScan(A)
		2. IdxScan(A)

Cost: 20

Output: [B]	Logical M-Exps	Physical M-Exps
	1. GET(B)	1. SeqScan(B)
		2. IdxScan(B)

CASCADES – MEMO TABLE

	Best Expr	Cost
[ABC]		
[AB]	[A]⋈$_{SM}$[B]	80
[A]	SeqScan(A)	10
[B]	SeqScan(B)	20
[C]	IdxScan(C)	5

Output: [ABC]

Logical M-Exps	Physical M-Exps
1. [AB]⋈[C]	1. [AB]⋈$_{NL}$C
2. [BC]⋈[A]	2. [BC]⋈$_{NL}$A
3. [AC]⋈[B]	3. [AC]⋈$_{NL}$B
4. [B]⋈[AC]	⋮

Cost: 50+(10+20)

Output: [AB]

Logical M-Exps	Physical M-Exps
1. [A]⋈[B]	1. [A]⋈$_{NL}$[B]

Output: [BC]

Logical M-Exps	Physical M-Exps
1. [B]⋈[C]	

Output: [AC]

Logical M-Exps	Physical M-Exps
1. [A]⋈[C]	
2. [C]⋈[A]	

Cost: 5

Output: [C]

Logical M-Exps	Physical M-Exps
1. GET(C)	1. SeqScan(C)
	2. IdxScan(C)

Cost: 10

Output: [A]

Logical M-Exps	Physical M-
1. GET(A)	1. SeqScan
	2. IdxScan(A)

[B]

l M-Exps
Scan(B)
IdxScan(B)

SEARCH TERMINATION

Approach #1: Wall-clock Time

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

Approach #2: Cost Threshold

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

Approach #3: Transformation Exhaustion

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

CASCADES IMPLEMENTATIONS

Standalone:

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

Integrated:

→ Microsoft SQL Server (1990s)
→ Tandem NonStop SQL (1990s)
→ Clustrix (2000s)
→ CMU Peloton (2010s – RIP)

PIVOTAL ORCA

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

A DBMS can use 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

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.

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

MEMSQL OPTIMIZER

Rewriter

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

Enumerator

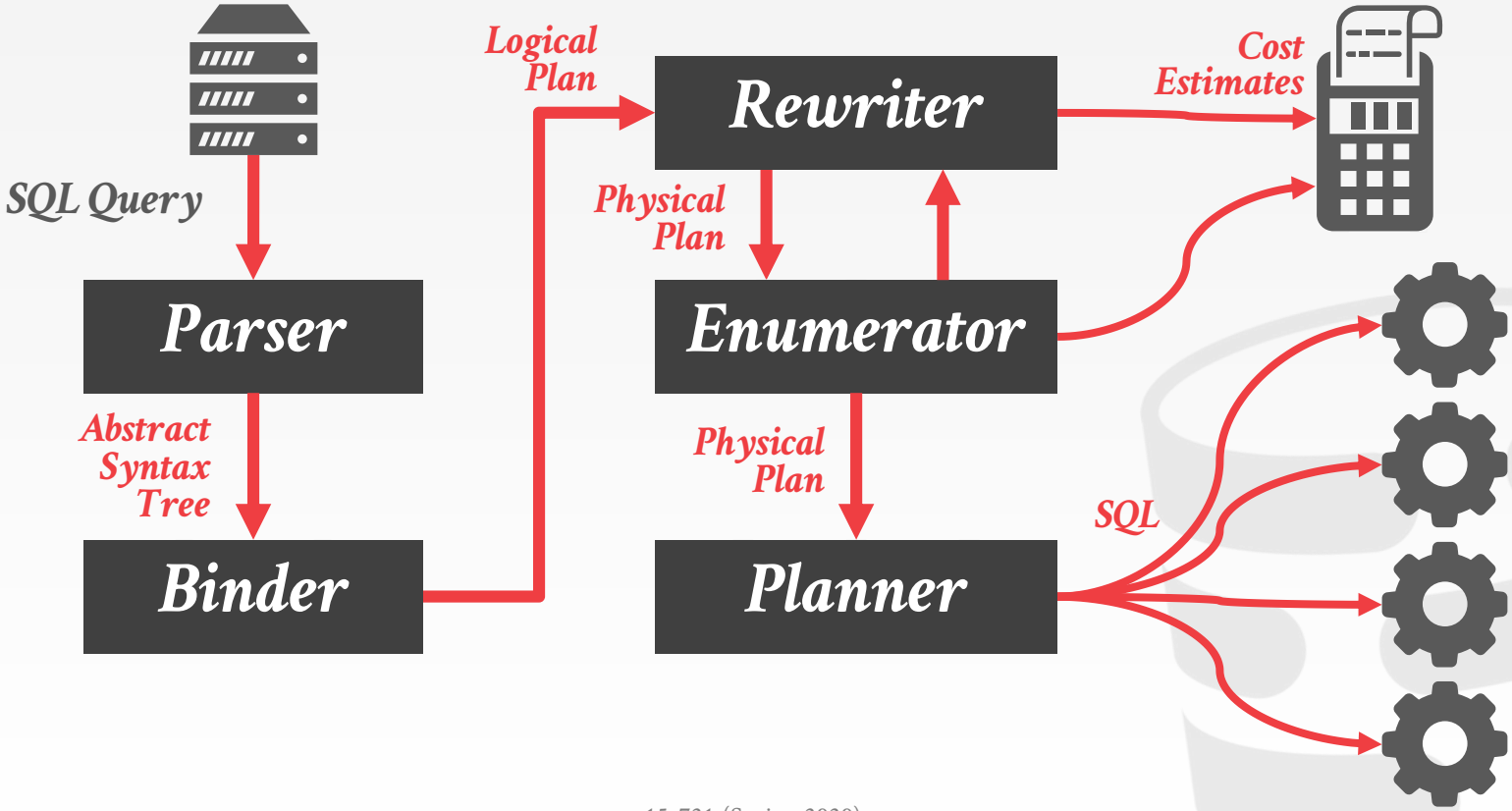
→ Logical-to-physical transformations.
→ Mostly join ordering.

Planner

→ Convert physical plans back to SQL.
→ Contains MemSQL-specific commands for moving data.

THE MEMSQL QUERY OPTIMIZER
VLDB 2017

MEMSQL OPTIMIZER OVERVIEW



PARTING THOUGHTS

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

PARTING THOUGHTS

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

Project #3

CODE REVIEW GUIDE

CODE REVIEWS

Each group will send a pull request to the CMU-DB master branch.

→ This will automatically run tests + coverage calculation.
→ PR must be able to merge cleanly into master branch.
→ Reviewing group will write comments on that request.
→ Add the URL to the Google spreadsheet and notify the reviewing team that it is ready.

Please be helpful and courteous.

GENERAL TIPS

The dev team should provide you with a summary of what files/functions the reviewing team should look at.

Review fewer than 400 lines of code at a time and only for at most 60 minutes.

Use a **checklist** to outline what kind of problems you are looking for.

Source: SmartBear

CHECKLIST – GENERAL

Does the code work?

Is all the code easily understood?

Is there any redundant or duplicate code?

Is the code as modular as possible?

Can any global variables be replaced?

Is there any commented out code?

Is it using proper debug log functions?

Source: Gareth Wilson

CHECKLIST – DOCUMENTATION

Do comments describe the intent of the code?

Are all functions commented?

Is any unusual behavior described?

Is the use of 3rd-party libraries documented?

Is there any incomplete code?

Source: Gareth Wilson

CHECKLIST – TESTING

Do tests exist and are they comprehensive?

Are the tests really testing the feature?

Are they relying on hardcoded answers?

What is the code coverage?

Source: Gareth Wilson

NEXT CLASS

Non-Traditional Query Optimization Methods