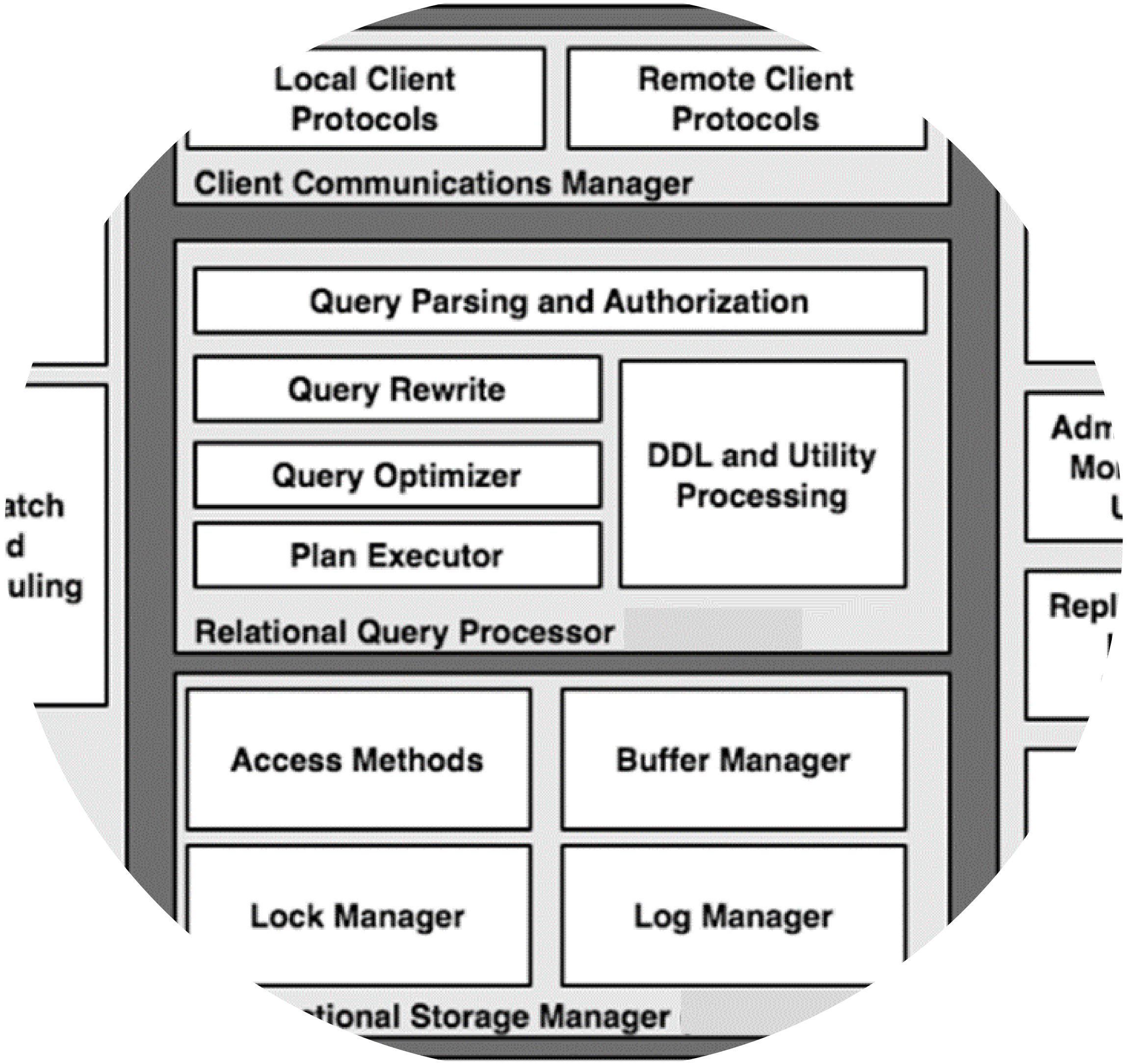
Advanced Databases

Query Optimization

Foundations and Trends in Databases
J. M. Hellerstein, M. Stonebraker
and J. Hamilton
Vol. 1, No. 2 (2007) 141–259, 2007
DOI: 10.1561/1900000002

Query Optimizer

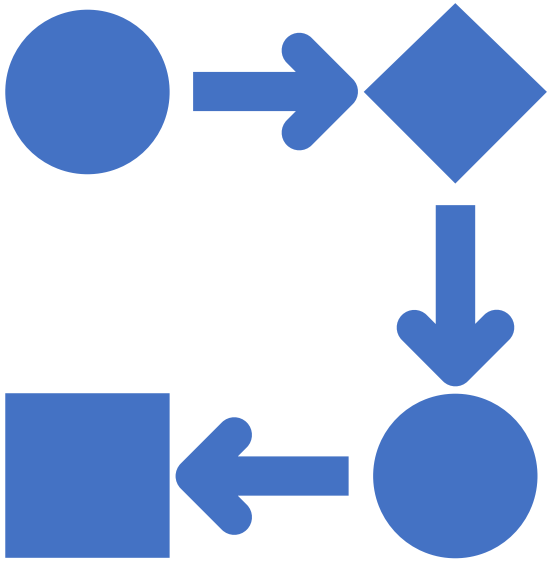


- Three components:
  - 1. Search space
  - 2. Plan enumeration algorithms
  - 3. Cardinality and cost estimation

- Note: First query optimizer was for System R, from IBM, in 1979, you will often see this references when reading about optimization in relational databases

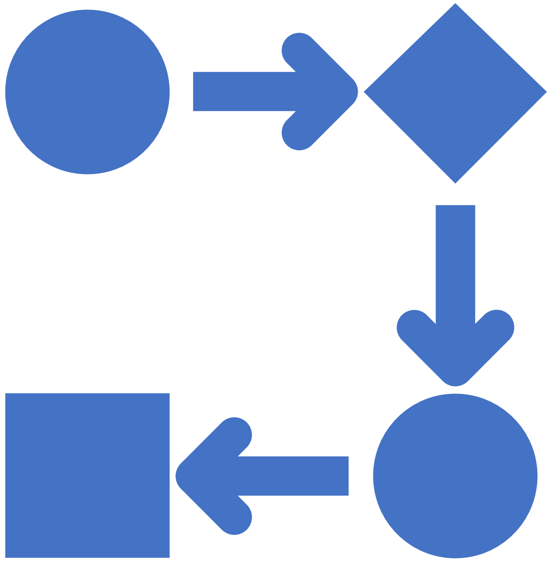
Approaches

- Heuristics
- Heuristics + Cost-based Join Order Search
- Randomized Algorithms
- Stratified Search
- Unified Search



Heuristic Optimization

- Define static rules that transform logical operators to a physical plan.
    - Perform most restrictive selection early
    - Perform all selections before joins
    - Predicate/Limit/Projection pushdowns
        - Predicate part of SQL that filters data
        - Pushdown
            - If you issue a query in one place to run against a lot of data that's in another place, you could spawn a lot of network traffic, which could be slow and costly.
            - If you can "push down" parts of the query to where the data is stored, and thus filter out most of the data, then you can greatly reduce network traffic.
    - Join ordering based on cardinality



Example

```sql
CREATE TABLE ARTIST (
  ID INT PRIMARY KEY,
  NAME VARCHAR(32)
);
```

```sql
CREATE TABLE ALBUM (
  ID INT PRIMARY KEY,
  NAME VARCHAR(32) UNIQUE
);
```

```sql
CREATE TABLE APPEARS (
  ARTIST_ID INT
   ↪REFERENCES ARTIST(ID),
  ALBUM_ID INT
   ↪REFERENCES ALBUM(ID),
  PRIMARY KEY
   ↪(ARTIST_ID, ALBUM_ID)
);
```

Example

Retrieve the names of people that appear on Joy's mixtape

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

Example

Retrieve the names of people that appear on Joy's mixtape

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

Q1

```
SELECT ALBUM.ID AS ALBUM_ID INTO TEMP1
  FROM ALBUM
 WHERE ALBUM.NAME="Joy's Covid Remix"
```

Q2

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, TEMP1
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=TEMP1.ALBUM_ID
```

Step #1: Decompose into single-variable queries

Example

Retrieve the names of people that appear on Joy's mixtape

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

Step #1: Decompose into single-variable queries

Q1

```
SELECT ALBUM.ID AS ALBUM_ID INTO TEMP1
  FROM ALBUM
 WHERE ALBUM.NAME="Joy's Covid Remix"
```

Q3

```
SELECT APPEARS.ARTIST_ID INTO TEMP2
  FROM APPEARS, TEMP1
 WHERE APPEARS.ALBUM_ID=TEMP1.ALBUM_ID
```

Q4

```
SELECT ARTIST.NAME
  FROM ARTIST, TEMP2
 WHERE ARTIST.ARTIST_ID=TEMP2.ARTIST_ID
```

Example

Retrieve the names of people that appear on Joy's mixtape

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

Step #1: Decompose into single-variable queries

Step #2: Substitute the values from Q1→Q3→Q4

Q1

```
SELECT ALBUM.ID AS ALBUM_ID INTO TEMP1
  FROM ALBUM
 WHERE ALBUM.NAME="Joy's Covid Remix"
```

Q3

```
SELECT APPEARS.ARTIST_ID INTO TEMP2
  FROM APPEARS, TEMP1
 WHERE APPEARS.ALBUM_ID=TEMP1.ALBUM_ID
```

Q4

```
SELECT ARTIST.NAME
  FROM ARTIST, TEMP2
 WHERE ARTIST.ARTIST_ID=TEMP2.ARTIST_ID
```

[redacted]

Example

Retrieve the names of people that appear on Joy's mixtape

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

Step #1: Decompose into single-variable queries

Step #2: Substitute the values from Q1→Q3→Q4

Q1

```
SELECT ALBUM.ID AS ALBUM_ID INTO TEMP1
  FROM ALBUM
 WHERE ALBUM.NAME="Joy's Covid Remix"
```

Q3

```
SELECT APPEARS.ARTIST_ID INTO TEMP2
  FROM APPEARS, TEMP1
 WHERE APPEARS.ALBUM_ID=TEMP1.ALBUM_ID
```

Q4

```
SELECT ARTIST.NAME
  FROM ARTIST, TEMP2
 WHERE ARTIST.ARTIST_ID=TEMP2.ARTIST_ID
```

Example

Retrieve the names of people that appear on Joy's mixtape

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

ALBUM_ID
9999

```
SELECT APPEARS.ARTIST_ID
  FROM APPEARS
 WHERE APPEARS.ALBUM_ID=9999
```

Q4

```
SELECT ARTIST.NAME
  FROM ARTIST, TEMP2
 WHERE ARTIST.ARTIST_ID=TEMP2.ARTIST_ID
```

Step #1: Decompose into single-variable queries

Step #2: Substitute the values from Q1→Q3→Q4

Example

Retrieve the names of people that appear on Joy's mixtape

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

ALBUM_ID
9999

ARTIST_ID
123
456

Q4

```
SELECT ARTIST.NAME
  FROM ARTIST, TEMP2
 WHERE ARTIST.ARTIST_ID=TEMP2.ARTIST_ID
```

Step #1: Decompose into single-variable queries

Step #2: Substitute the values from Q1→Q3→Q4

Example

Retrieve the names of people that appear on Joy's mixtape

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

ALBUM_ID
9999

ARTIST_ID
123
456

Step #1: Decompose into single-variable queries

Step #2: Substitute the values from Q1→Q3→Q4

```
SELECT ARTIST.NAME
  FROM ARTIST
 WHERE ARTIST.ARTIST_ID=123
```

```
SELECT ARTIST.NAME
  FROM ARTIST
 WHERE ARTIST.ARTIST_ID=456
```

Example

Retrieve the names of people that appear on Joy's mixtape

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

Step #1: Decompose into single-variable queries

Step #2: Substitute the values from Q1→Q3→Q4

ALBUM_ID
9999

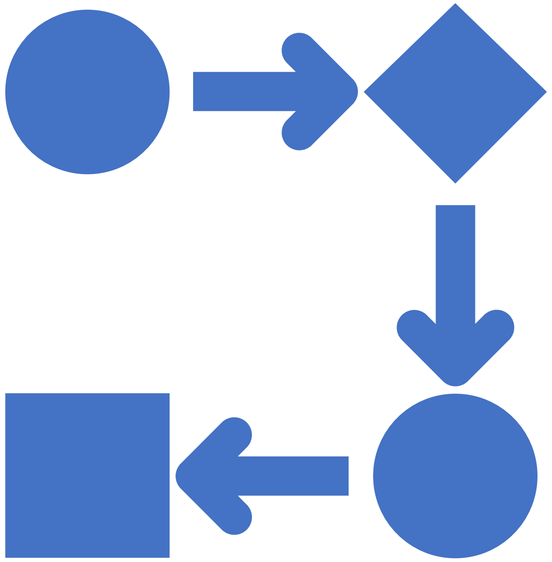
ARTIST_ID
123
456

NAME
O.D.B.

NAME
DJ Premier

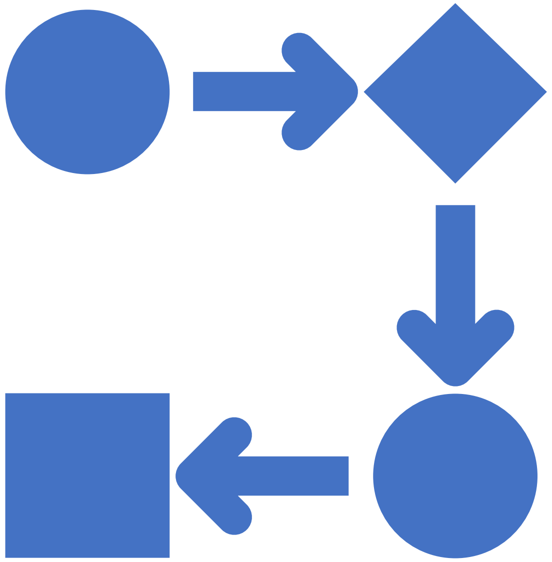
Heuristic Optimization

- Advantages:
  - Easy to implement and debug.
  - Works reasonably well and is fast for simple queries.
- Disadvantages:
  - Relies on magic constants that predict the efficacy of a planning decision.
    - Magic constants e.g. block size, parallelism granularity, thread weight
  - Nearly impossible to generate good plans when operators have complex inter-dependencies.



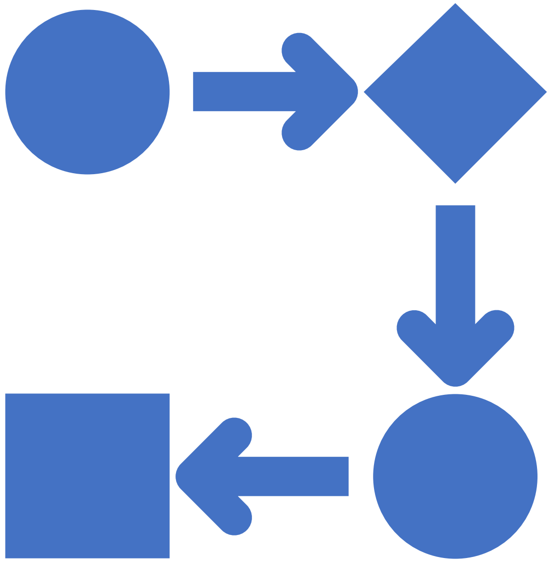
Heuristic Optimization + Cost Estimation

- Use static rules to perform initial optimization.
- Then use dynamic programming to determine the best join order for tables.
  - First cost-based query optimizer
  - Bottom-up planning (forward chaining) using a divide-and-conquer search method



System R Optimizer

- Break query up into blocks and generate the logical operators for each block.
- For each logical operator, generate a set of physical operators that implement it.
  - All combinations of join algorithms and access paths
- Then iteratively construct a “left-deep” tree that minimizes the estimated amount of work to execute the plan.



Example

Retrieve the names of people that appear on Joy's mixtape ordered by their artist id.

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Joy's Covid Remix"
 ORDER BY ARTIST.ID
```

Example

Retrieve the names of people that appear on Joy's mixtape ordered by their artist id.

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Joy's Covid Remix"
 ORDER BY ARTIST.ID
```

Step #1: Choose the best access paths to each table

Step #2: Enumerate all possible join orderings for tables

Step #3: Determine the join ordering with the lowest cost

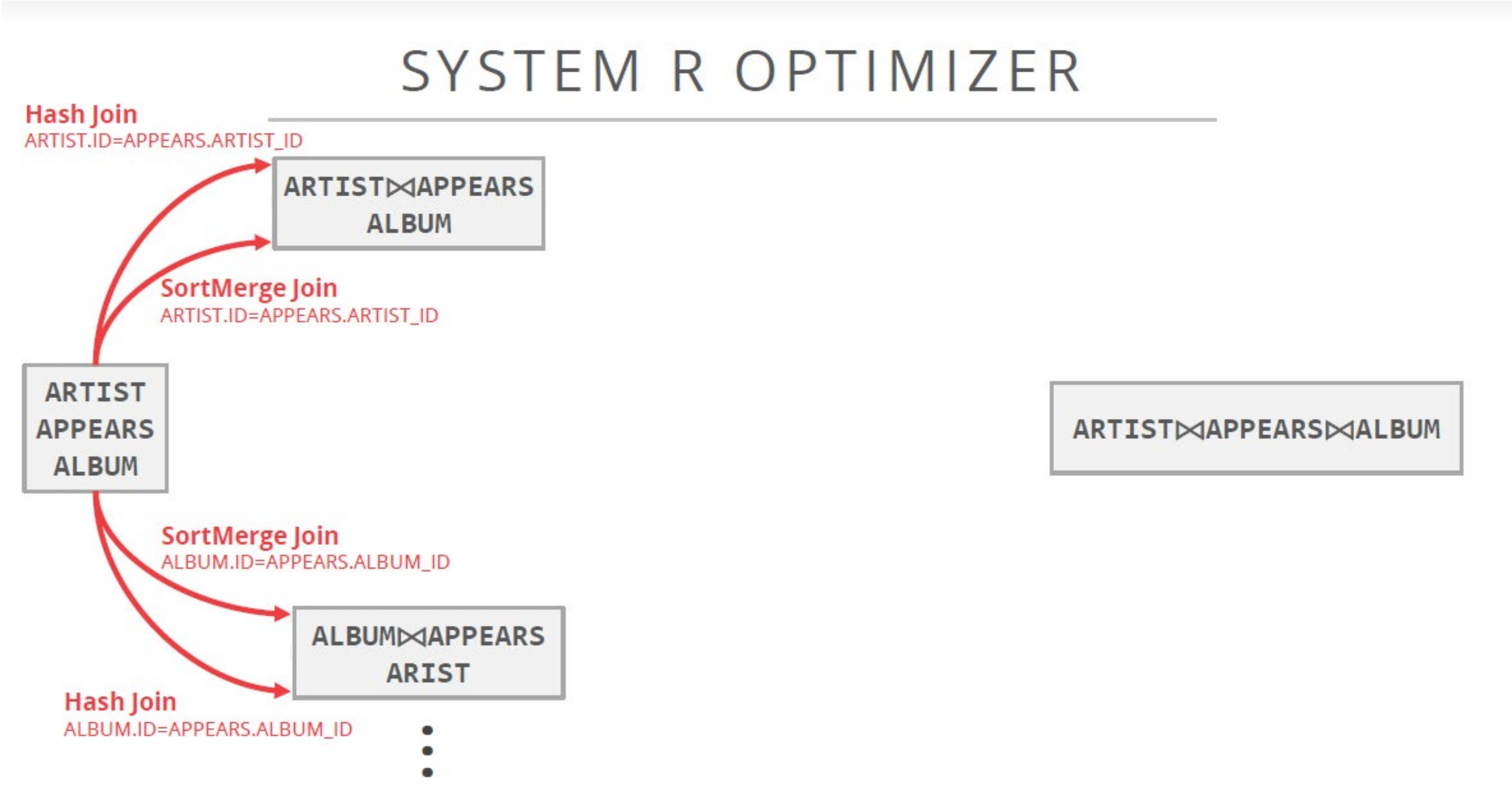
ARTIST: Sequential Scan

APPEARS: Sequential Scan

ALBUM: Index Look-up on NAME

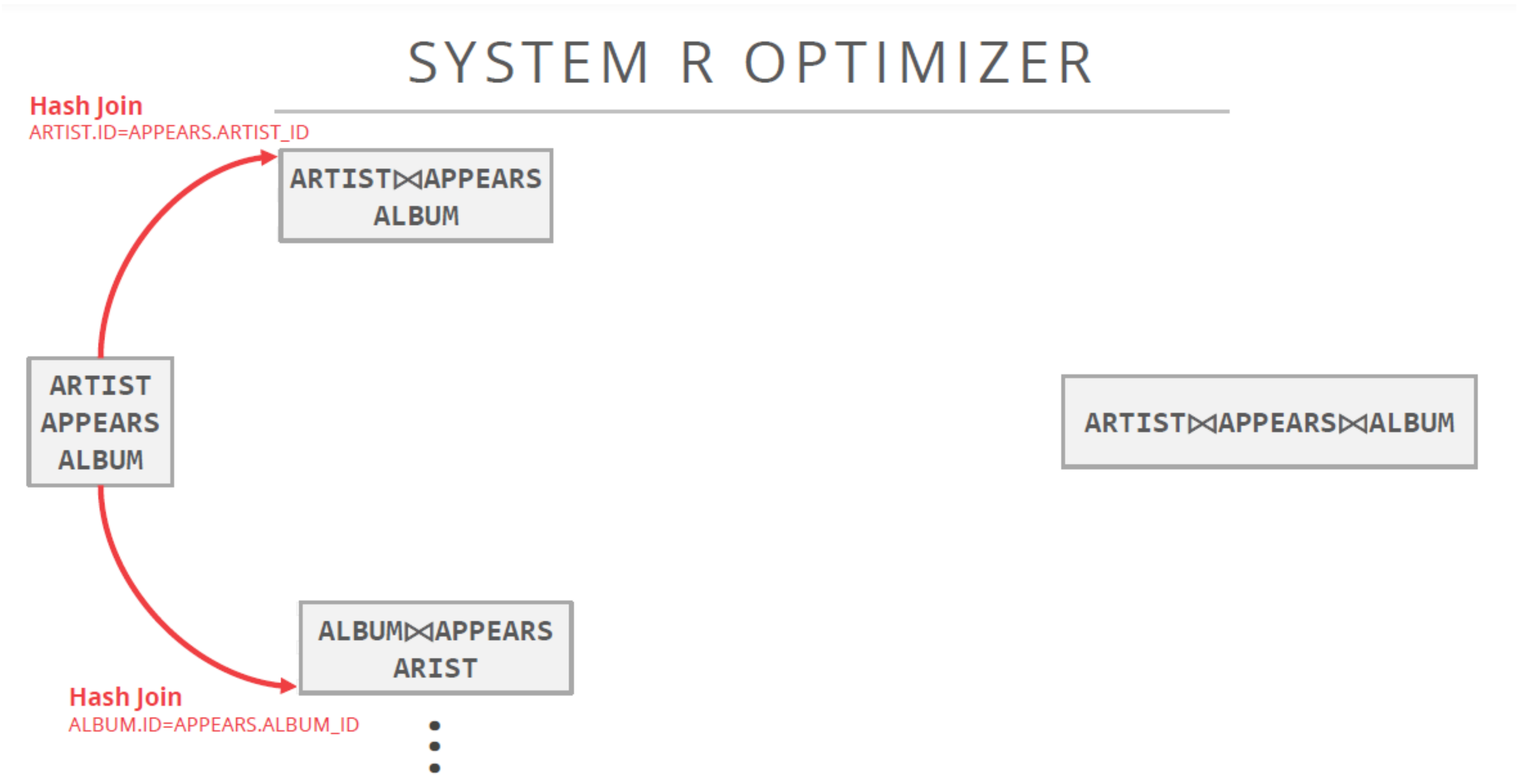
ARTIST	⋈	APPEARS	⋈	ALBUM
APPEARS	⋈	ALBUM	⋈	ARTIST
ALBUM	⋈	APPEARS	⋈	ARTIST
APPEARS	⋈	ARTIST	⋈	ALBUM
ARTIST		ALBUM	⋈	APPEARS
ALBUM		ARTIST	⋈	APPEARS
⋮		⋮		⋮

Example



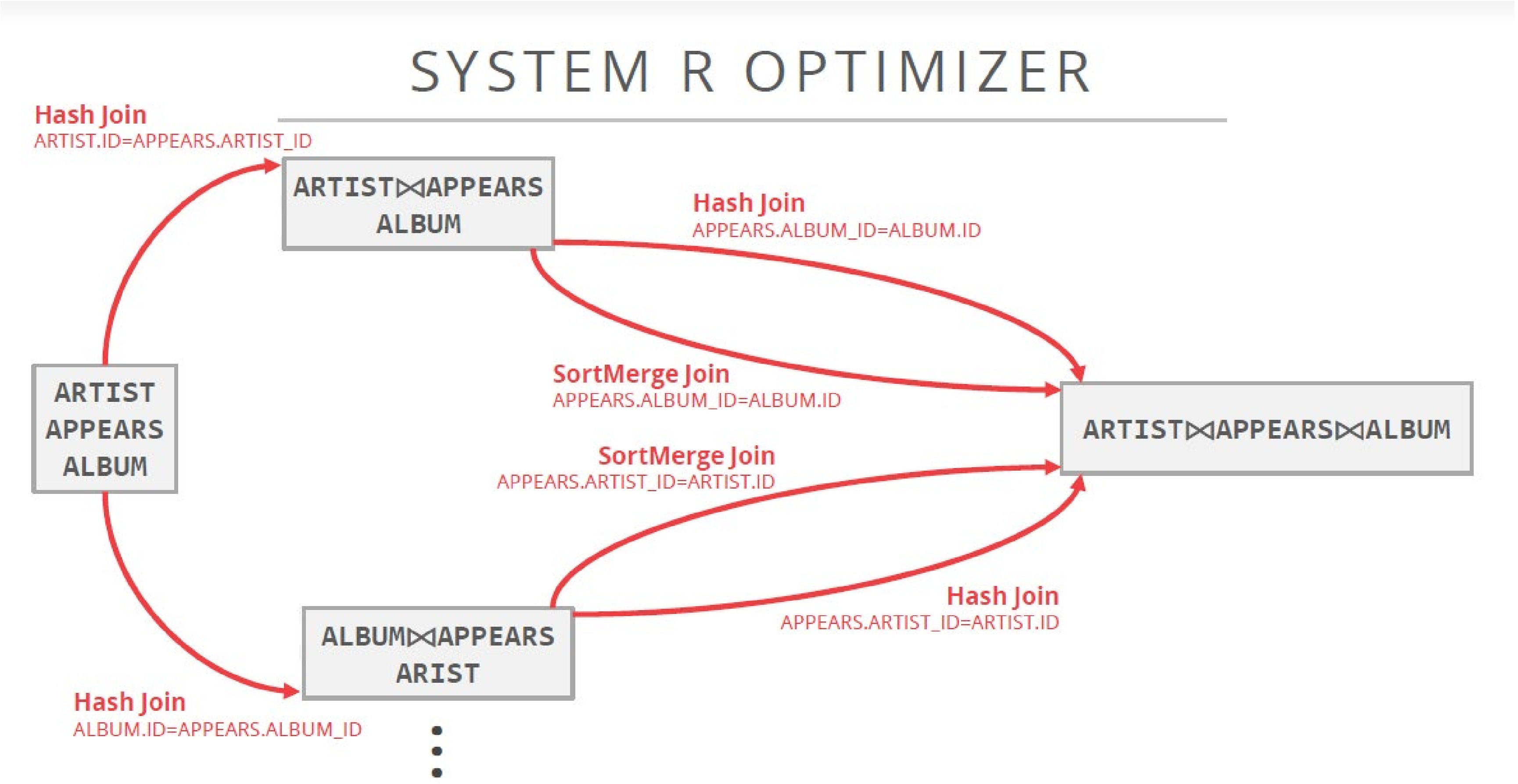
SYSTEM R OPTIMIZER
Hash Join
ARTIST.ID=APPEARS.ARTIST_ID
ARTIST⋈APPEARS
ALBUM
SortMerge Join
ARTIST.ID=APPEARS.ARTIST_ID
ARTIST
APPEARS
ALBUM
ARTIST⋈APPEARS⋈ALBUM
SortMerge Join
ALBUM.ID=APPEARS.ALBUM_ID
ALBUM⋈APPEARS
ARIST
Hash Join
ALBUM.ID=APPEARS.ALBUM_ID

Example

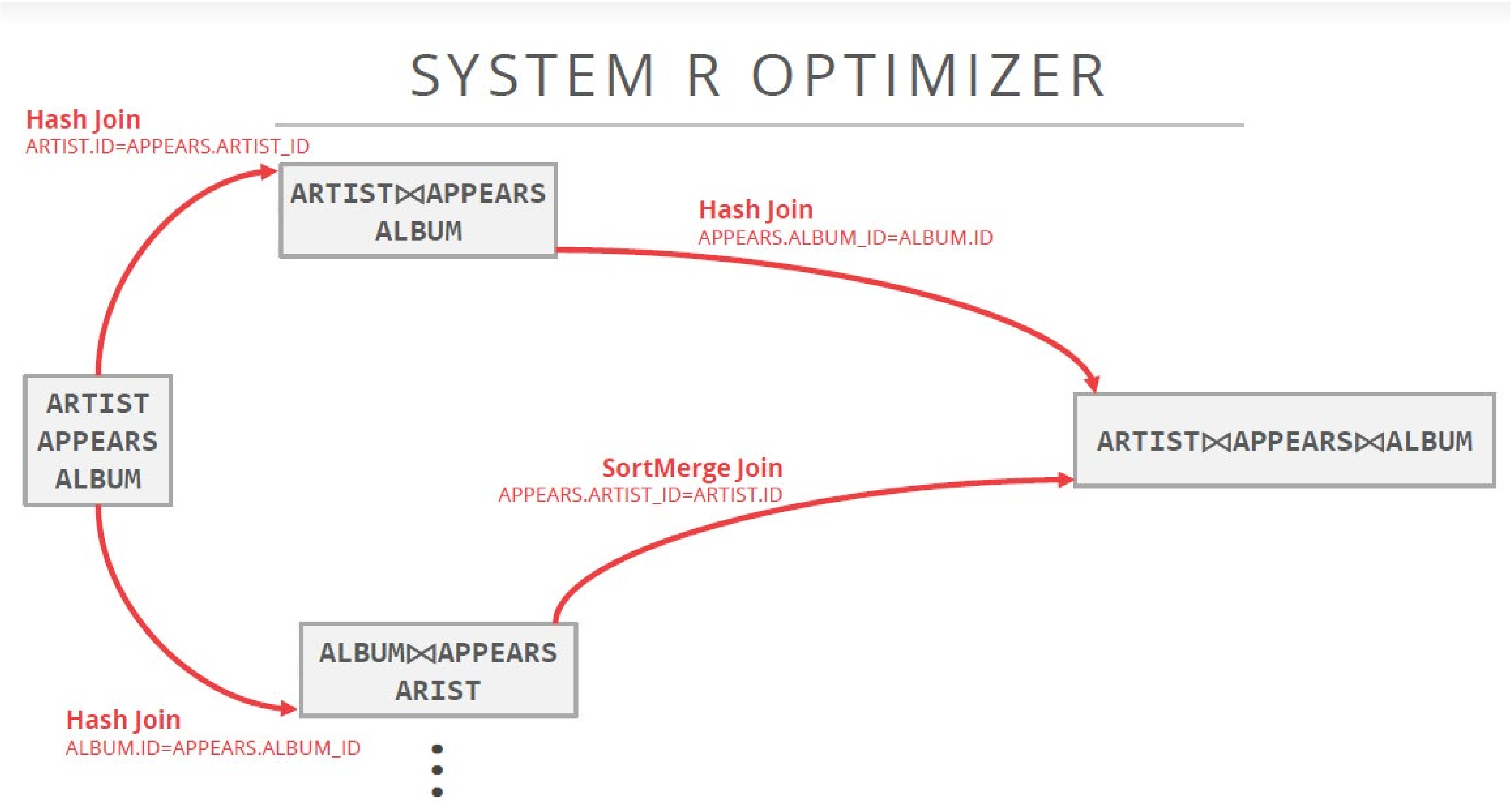


SYSTEM R OPTIMIZER
Hash Join
ARTIST.ID=APPEARS.ARTIST_ID
ARTIST⋈APPEARS
ALBUM
ARTIST
APPEARS
ALBUM
ARTIST⋈APPEARS⋈ALBUM
ALBUM⋈APPEARS
ARIST
Hash Join
ALBUM.ID=APPEARS.ALBUM_ID

Example


SYSTEM R OPTIMIZER
Hash Join
ARTIST.ID=APPEARS.ARTIST_ID
ARTIST⋈APPEARS
ALBUM
Hash Join
APPEARS.ALBUM_ID=ALBUM.ID
ARTIST
APPEARS
ALBUM
SortMerge Join
APPEARS.ALBUM_ID=ALBUM.ID
ARTIST⋈APPEARS⋈ALBUM
SortMerge Join
APPEARS.ARTIST_ID=ARTIST.ID
Hash Join
APPEARS.ARTIST_ID=ARTIST.ID
ALBUM⋈APPEARS
ARIST
Hash Join
ALBUM.ID=APPEARS.ALBUM_ID

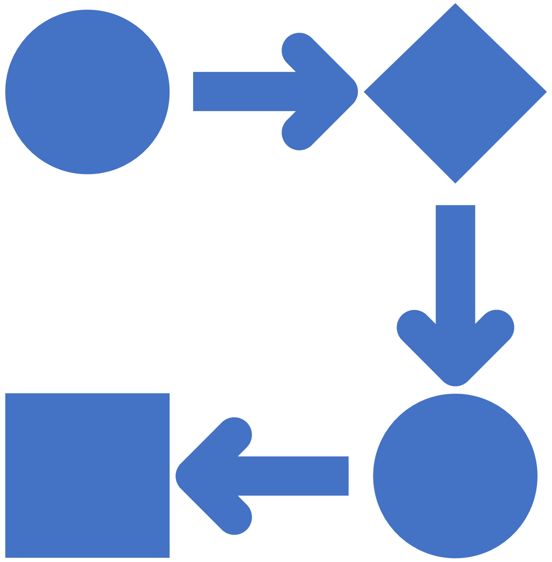
Example



SYSTEM R OPTIMIZER
Hash Join
ARTIST.ID=APPEARS.ARTIST_ID
ARTIST⋈APPEARS
ALBUM
Hash Join
APPEARS.ALBUM_ID=ALBUM.ID
ARTIST
APPEARS
ALBUM
ARTIST⋈APPEARS⋈ALBUM
SortMerge Join
APPEARS.ARTIST_ID=ARTIST.ID
ALBUM⋈APPEARS
ARIST
Hash Join
ALBUM.ID=APPEARS.ALBUM_ID

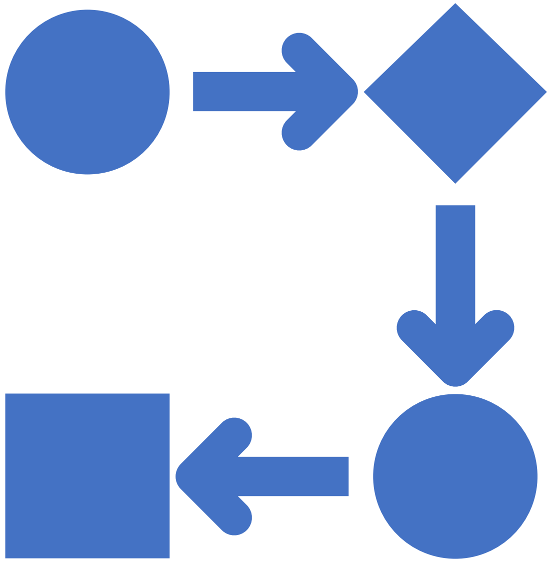
Heuristic Optimization + Cost Estimation

- Advantages:
  - Easy to implement and debug.
  - Works reasonably well and is fast for simple queries.
- Disadvantages:
  - Relies on magic constants that predict the efficacy of a planning decision.
  - Nearly impossible to generate good plans when operators have complex inter-dependencies



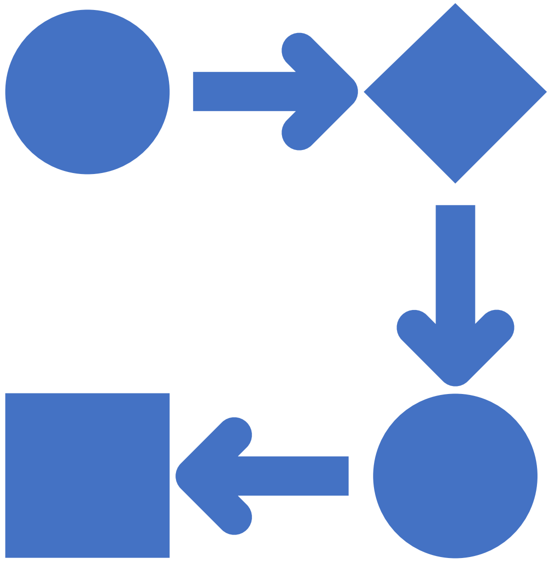
Randomized Algorithms

- Perform a random walk over a solution space of all possible (valid) plans for a query
- Continue searching until a cost threshold is reached or the optimizer runs for a particular length of time
- Example: Postgres' genetic algorithm



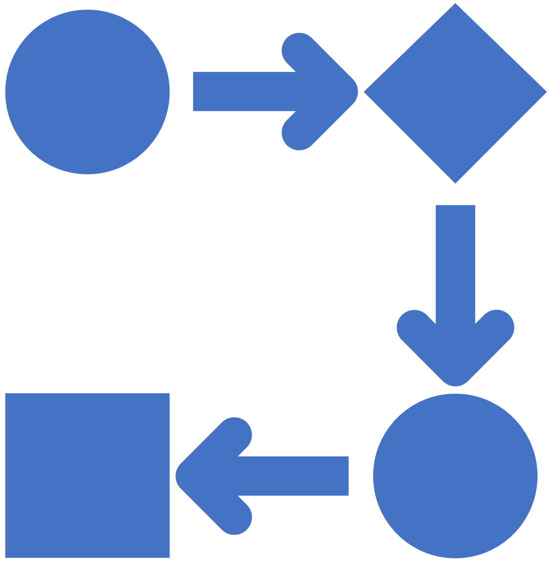
Randomized Algorithms
Simulated Annealing

- Start with a query plan that is generated using the heuristic-only approach
- Compute random permutations of operators (e.g., swap the join order of two tables)
  - Always accept a change that reduces cost
  - Only accept a change that increases cost with some probability
  - Reject any change that violates correctness (e.g., sort ordering)



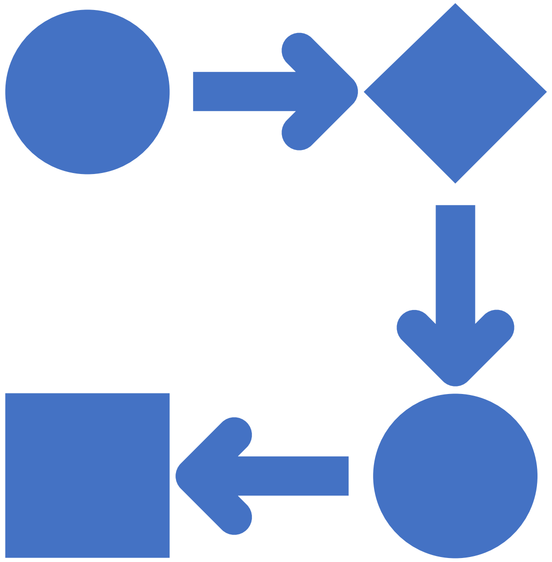
Randomized Algorithm

- Advantages:
  - Jumping around the search space randomly allows the optimizer to get out of local minimums.
  - Low memory overhead (if no history is kept).
- Disadvantages:
  - Difficult to determine why the DBMS may have chose a particular plan.
  - Have to do extra work to ensure that query plans are deterministic.
  - Still have to implement correctness rules.



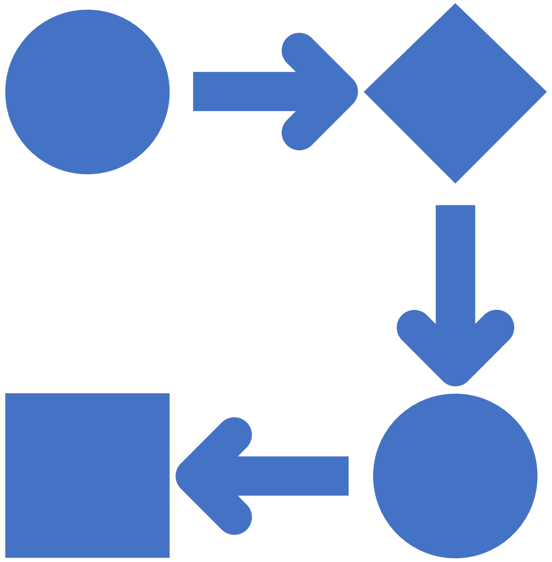
Optimizer Generators

- Use a rule engine that allows transformations to modify the query plan operators.
- The physical properties of data is embedded with the operators themselves.
- Choice #1: Stratified Search
  - Planning is done in multiple stages
- Choice #2: Unified Search
  - Perform query planning all at once.



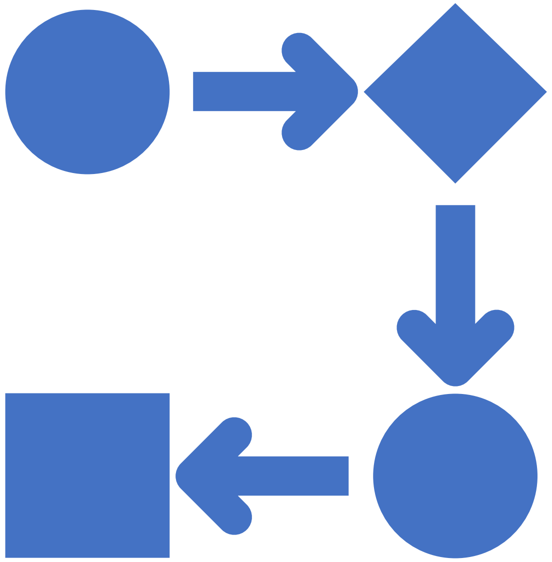
Stratified Search

- 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



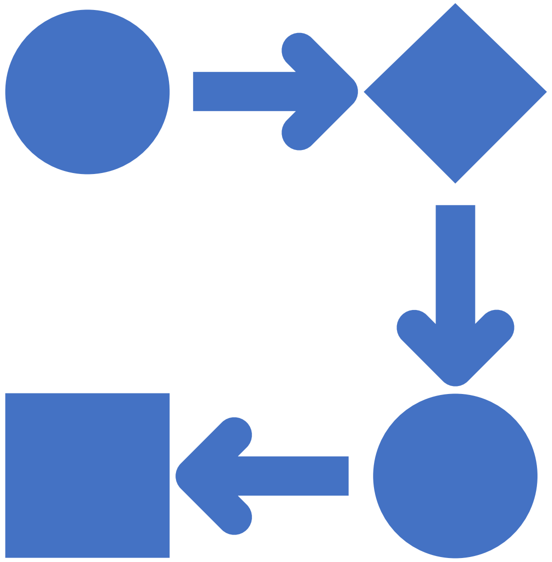
Stratified Search

- Example: Starburst Optimizer
  - Better implementation of the System R optimizer that uses declarative rules.
  - Stage #1: Query Rewrite
    - Compute a SQL-block-level, relational calculus-like representation of queries.
  - Stage #2: Plan Optimization
    - Execute a System R-style dynamic programming phase once query rewrite has completed.
- Used by latest version of IBM DB2



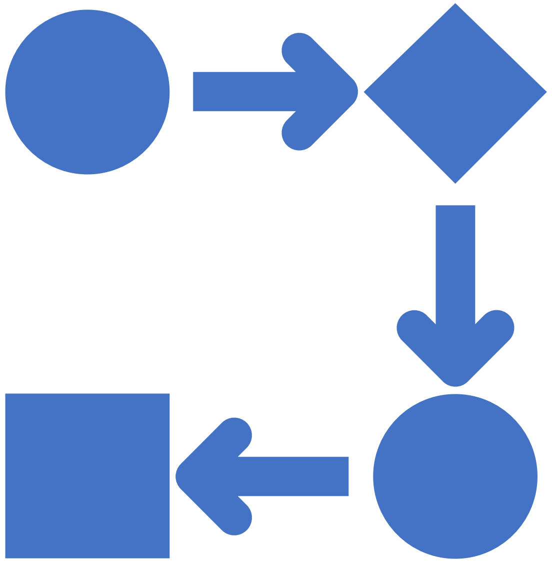
Stratified Search

- Advantages:
  - Works well in practice with fast performance.
- Disadvantages:
  - Difficult to assign priorities to transformations
  - Some transformations are difficult to assess without computing multiple cost estimations.
  - Rules maintenance is a huge pain.



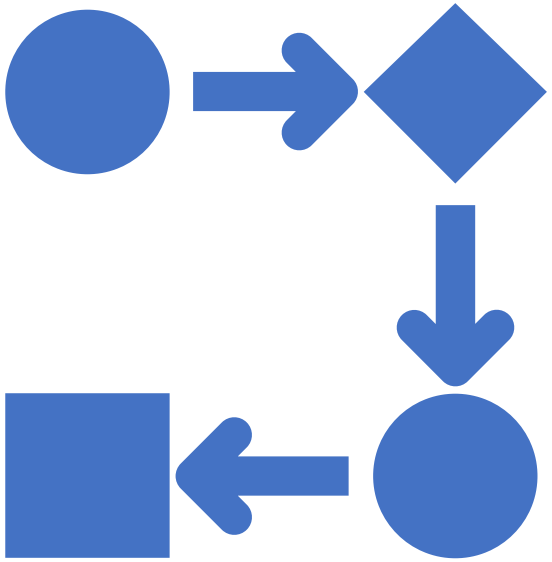
Unified Search

- 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 memorization to reduce redundant work



Unified Search

- Example: Volcano Optimizer
  - General purpose cost-based query optimizer, based on equivalence rules on algebras.
  - Easily add new operations and equivalence rules.
  - Treats physical properties of data as first-class entities during planning.
  - Top-down approach (backward chaining) using branch-and-bound search.
- Used by NonStop SQL



Unified Search

- Example: Cascades Optimizer
    - Object-oriented implementation of the Volcano query optimizer.
    - Simplistic expression re-writing can be through a direct mapping function rather than an exhaustive search.
- Used by SQL Server, Greenplum’s Orca, and Apache Calcite.