Carnegie Mellon University

Query Optimization

Lecture #13

Database Systems
15-445/15-645
Fall 2018

AP
Andy Pavlo
Computer Science
Carnegie Mellon Univ.

ADMINISTRIVIA

Mid-term Exam is on Wednesday October 17th
→ See mid-term exam guide for more info.

Project #2 – Checkpoint #2 is due Friday October 19th @ 11:59pm.

QUERY OPTIMIZATION

Remember that SQL is declarative.

→ User tells the DBMS what answer they want, not how to get the answer.

There can be a big difference in performance based on plan is used:

→ See last week: 1.3 hours vs. 0.45 seconds

IBM SYSTEM R

First implementation of a query optimizer.
People argued that the DBMS could never choose a query plan better than what a human could write.

A lot of the concepts from **System R**'s optimizer are still used today.

QUERY OPTIMIZATION

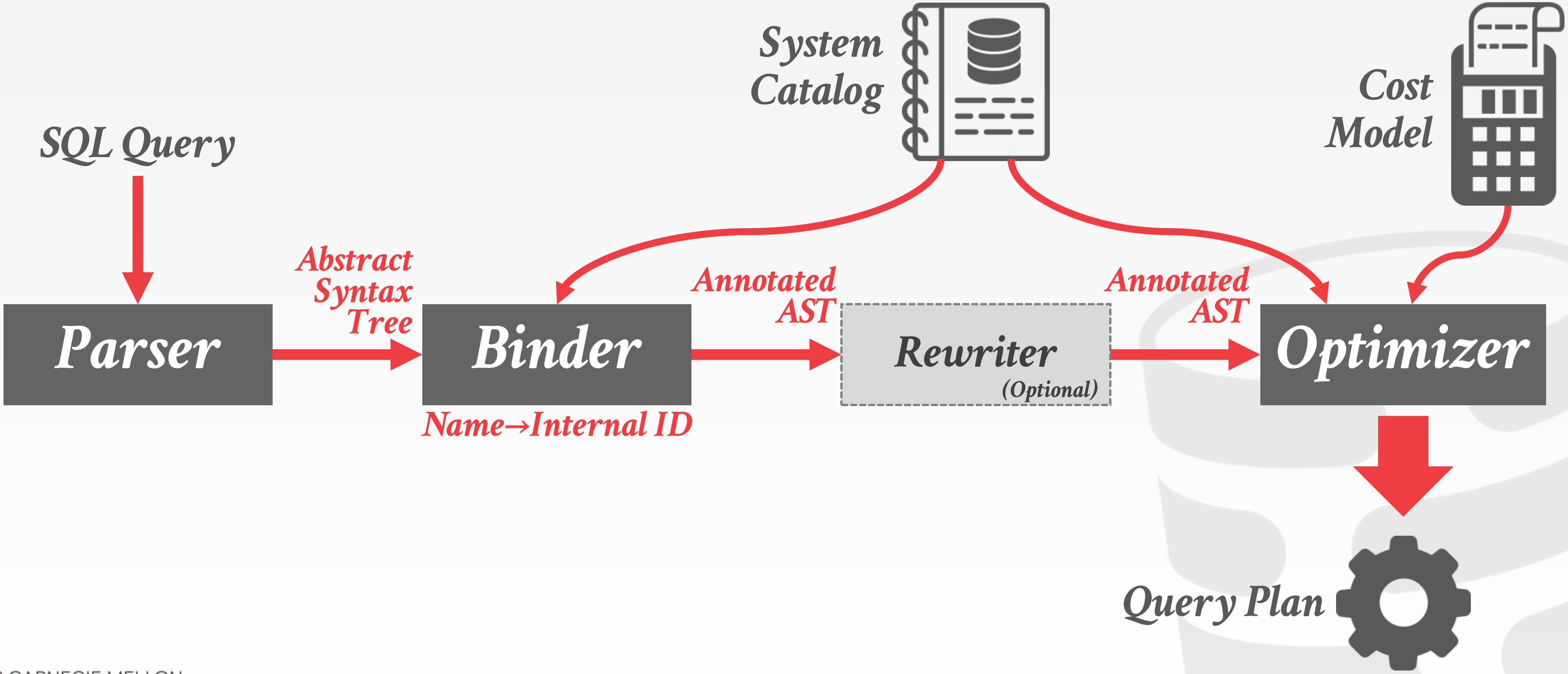
Heuristics / Rules

→ Rewrite the query to remove stupid / inefficient things.
→ Does not require a cost model.

Cost-based Search

→ Use a cost model to evaluate multiple equivalent plans and pick the one with the lowest cost.

QUERY PLANNING OVERVIEW



TODAY'S AGENDA

Relational Algebra Equivalences
Plan Cost Estimation
Plan Enumeration
Nested Sub-queries
Mid-Term Review

RELATIONAL ALGEBRA EQUIVALENCES

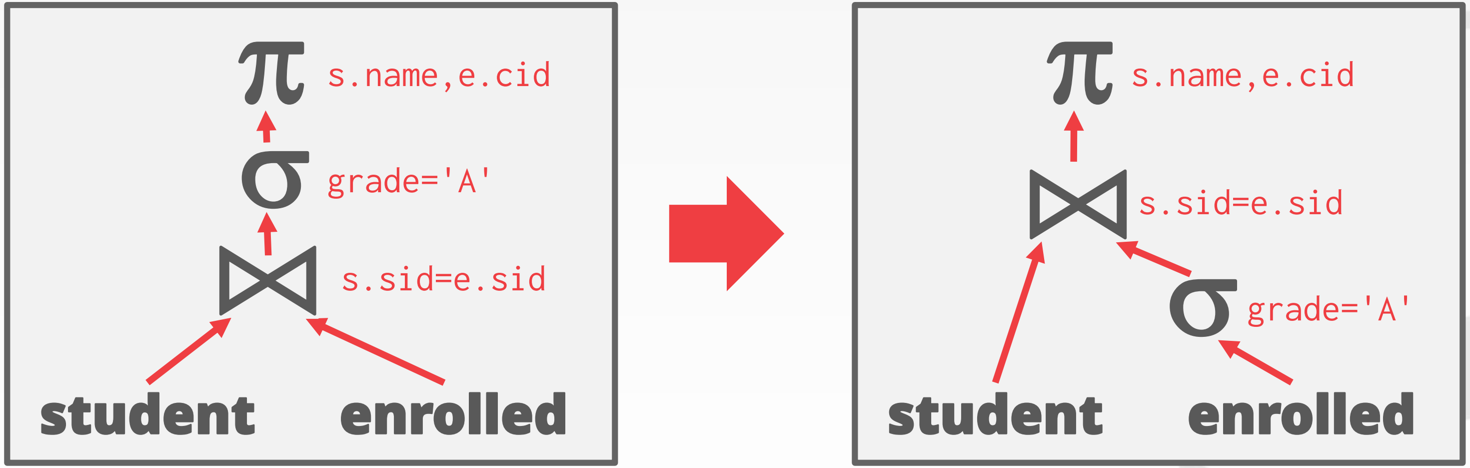
Two relational algebra expressions are equivalent if they generate the same set of tuples.

The DBMS can identify better query plans without a cost model.

This is often called **query rewriting**.

PREDICATE PUSHDOWN

```
SELECT s.name, e.cid
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
   AND e.grade = 'A'
```



RELATIONAL ALGEBRA EQUIVALENCES

```
SELECT s.name, e.cid
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
   AND e.grade = 'A'
```

$$\pi_{name,\ cid}(\sigma_{grade='A'}(student \bowtie enrolled))$$

$$=$$

$$\pi_{name,\ cid}(student \bowtie (\sigma_{grade='A'}(enrolled)))$$

RELATIONAL ALGEBRA EQUIVALENCES

Selections:

→ Perform filters as early as possible.

→ Reorder predicates so that the DBMS applies the most selective one first.

→ Break a complex predicate, and push down

$$\sigma_{p1 \wedge p2 \wedge \ldots pn}(\mathbf{R}) = \sigma_{p1}(\sigma_{p2}(\ldots\sigma_{pn}(\mathbf{R})))$$

Simplify a complex predicate

→ `(X=Y AND Y=3) → X=3 AND Y=3`

RELATIONAL ALGEBRA EQUIVALENCES

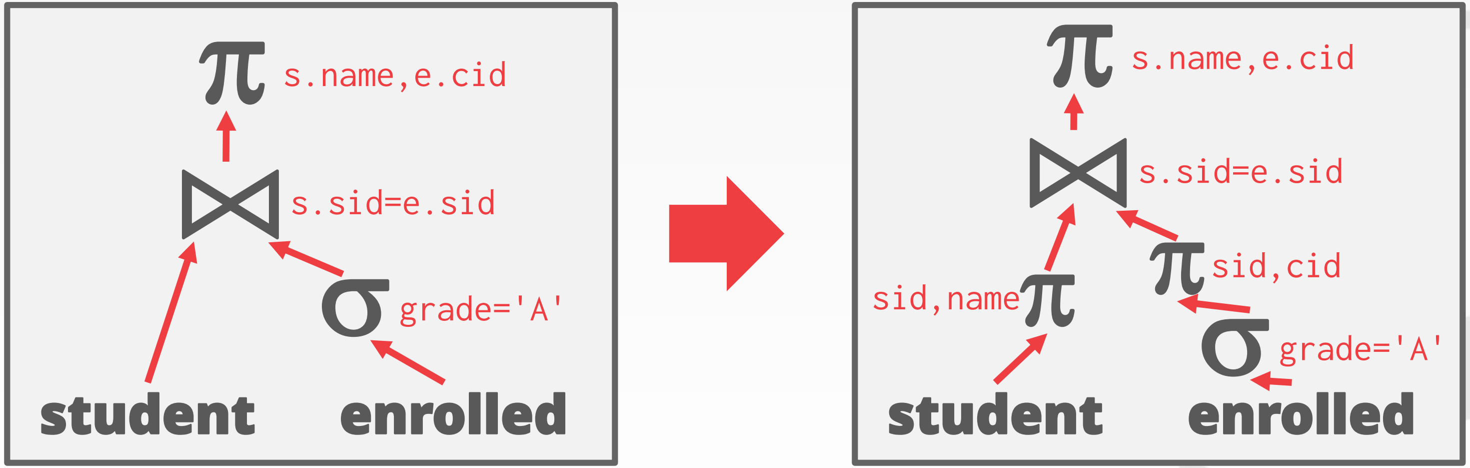
Projections:

→ Perform them early to create smaller tuples and reduce intermediate results (if duplicates are eliminated)
→ Project out all attributes except the ones requested or required (e.g., joining keys)

This is not important for a column store…

PROJECTION PUSHDOWN

```
SELECT s.name, e.cid
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
   AND e.grade = 'A'
```



```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```
SELECT * FROM A WHERE 1 = 0;
```
X

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```
SELECT * FROM A WHERE 1 = 0;
```

X

```
SELECT * FROM A WHERE 1 = 1;
```

Source: Lukas Eder

```sql
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```sql
SELECT * FROM A WHERE 1 = 0;
```

X

```sql
SELECT * FROM A;
```

Source: Lukas Eder

```sql
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```sql
SELECT * FROM A WHERE 1 = 0;
```

```sql
SELECT * FROM A;
```

Join Elimination

```sql
SELECT A1.*
  FROM A AS A1 JOIN A AS A2
    ON A1.id = A2.id;
```

Source: Lukas Eder

```sql
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```sql
SELECT * FROM A WHERE 1 = 0;
```

X

```sql
SELECT * FROM A;
```

Join Elimination

```sql
SELECT * FROM A;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Ignoring Projections

```
SELECT * FROM A AS A1
 WHERE EXISTS(SELECT * FROM A AS A2
               WHERE A1.id = A2.id);
```

Source: Lukas Eder

```sql
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Ignoring Projections

```sql
SELECT * FROM A;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Ignoring Projections

```
SELECT * FROM A;
```

Merging Predicates

```
SELECT * FROM A
 WHERE val BETWEEN 1 AND 100
    OR val BETWEEN 50 AND 150;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Ignoring Projections

```
SELECT * FROM A;
```

Merging Predicates

```
SELECT * FROM A
 WHERE val BETWEEN 1 AND 150;
```

Source: Lukas Eder

RELATIONAL ALGEBRA EQUIVALENCES

Joins:

→ Commutative, associative

$R \bowtie S = S \bowtie R$

$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

How many different orderings are there for an *n*-way join?

RELATIONAL ALGEBRA EQUIVALENCES

How many different orderings are there for an *n*-way join?

Catalan number $\approx 4^n$

→ Exhaustive enumeration will be too slow.

We'll see in a second how an optimizer limits the search space...

COST ESTIMATION

How long will a query take?

→ CPU: Small cost; tough to estimate
→ Disk: # of block transfers
→ Memory: Amount of DRAM used
→ Network: # of messages

How many tuples will be read/written?

What statistics do we need to keep?

STATISTICS

The DBMS stores internal statistics about tables, attributes, and indexes in its internal catalog.

Different systems update them at different times.

Manual invocations:

→ Postgres/SQLite: **ANALYZE**

→ Oracle/MySQL: **ANALYZE TABLE**

→ SQL Server: **UPDATE STATISTICS**

→ DB2: **RUNSTATS**

STATISTICS

For each relation **R**, the DBMS maintains the following information:

- → N_R: Number of tuples in **R**.
- → `V(A,R)`: Number of distinct values for attribute **A**.

DERIVABLE STATISTICS

The **selection cardinality** SC(A,R) is the average number of records with a value for an attribute A given N_R / V(A,R)

Note that this assumes ***data uniformity.***

→ 10,000 students, 10 colleges – how many students in SCS?

SELECTION STATISTICS

Equality predicates on unique keys are easy to estimate.

```
SELECT * FROM people
 WHERE id = 123
```

What about more complex predicates? What is their selectivity?

```
SELECT * FROM people
 WHERE val > 1000
```

```
SELECT * FROM people
 WHERE age = 30
   AND status = 'Lit'
```

COMPLEX PREDICATES

The **selectivity** (**sel**) of a predicate **P** is the fraction of tuples that qualify.

Formula depends on type of predicate:

→ Equality
→ Range
→ Negation
→ Conjunction
→ Disjunction

COMPLEX PREDICATES

The **selectivity** (`sel`) of a predicate `P` is the fraction of tuples that qualify.

Formula depends on type of predicate:

→ Equality
→ Range
→ Negation
→ Conjunction
→ Disjunction

SELECTIONS – COMPLEX PREDICATES

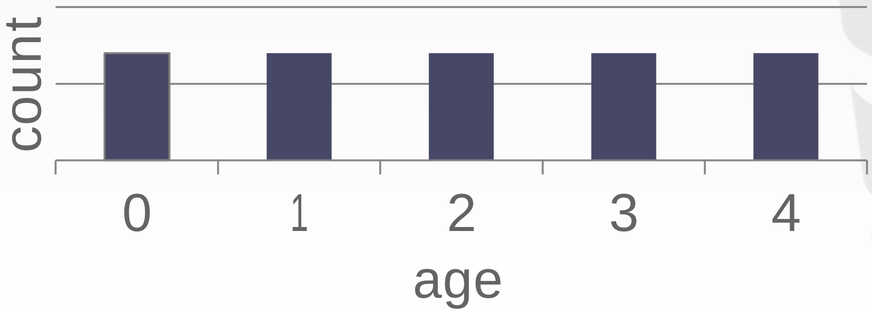
Assume that **V(age,people)** has five distinct values (0–4) and $N_R = 5$

Equality Predicate: **A=constant**

→ **sel(A=constant) = SC(P) / V(A,R)**

→ Example: **sel(age=2) =**

```
SELECT * FROM people
 WHERE age = 2
```



SELECTIONS – COMPLEX PREDICATES

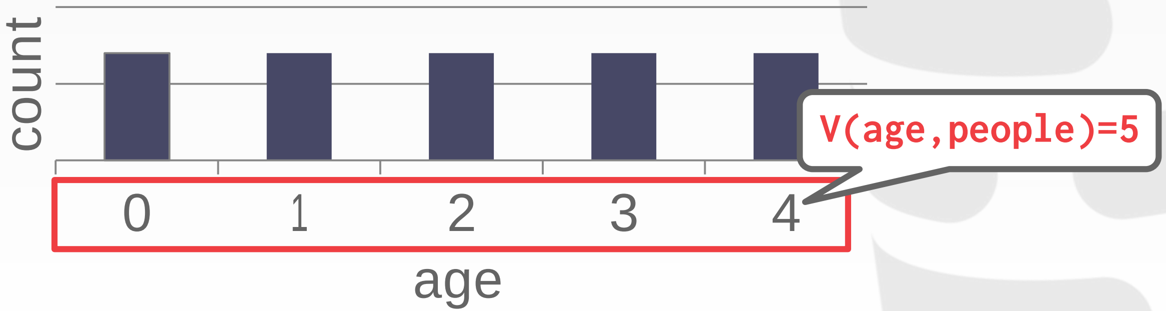
Assume that **V(age,people)** has five distinct values (0–4) and N_R = 5

Equality Predicate: **A=constant**

→ `sel(A=constant) = SC(P) / V(A,R)`

→ Example: `sel(age=2) =`

```
SELECT * FROM people
 WHERE age = 2
```



SELECTIONS – COMPLEX PREDICATES

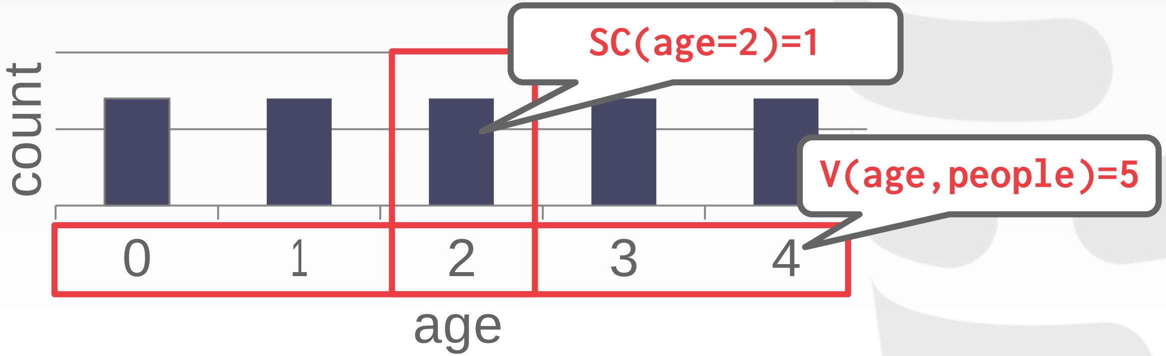
Assume that **V(age,people)** has five distinct values (0–4) and N_R = 5

```
SELECT * FROM people
 WHERE age = 2
```

Equality Predicate: **A=constant**

→ **sel(A=constant) = SC(P) / V(A,R)**

→ Example: **sel(age=2) = 1/5**



SELECTIONS – COMPLEX PREDICATES

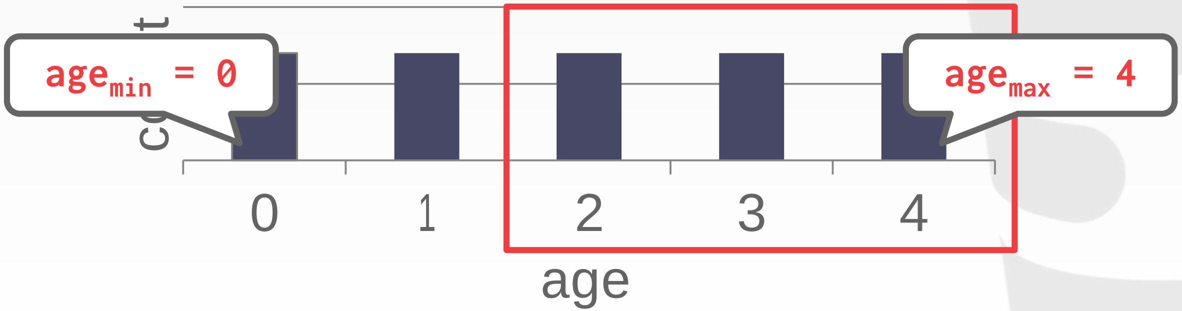
Range Query:

→ sel(A>=a) = $(A_{max} - a) / (A_{max} - A_{min})$

→ Example: sel(age>=2) = (4 – 2) / (4 – 0)

= 1/2

```
SELECT * FROM people
 WHERE age >= 2
```

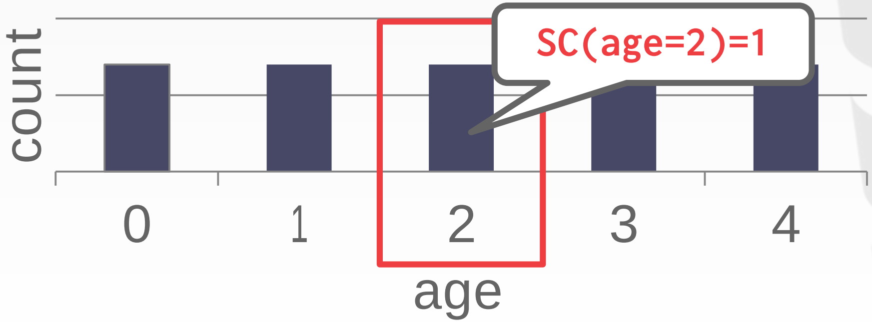


SELECTIONS – COMPLEX PREDICATES

Negation Query:

→ `sel(not P) = 1 - sel(P)`
→ Example: `sel(age != 2)`

```
SELECT * FROM people
 WHERE age != 2
```



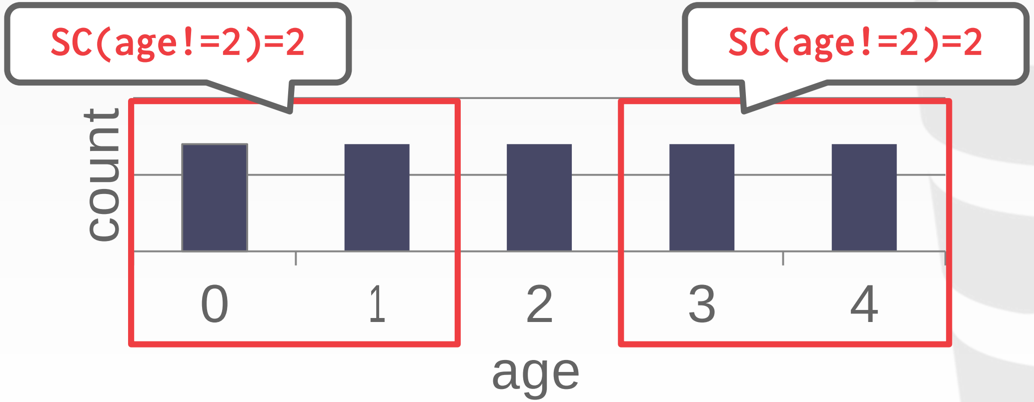
SELECTIONS – COMPLEX PREDICATES

Negation Query:

→ `sel(not P) = 1 - sel(P)`
→ Example: `sel(age != 2) = 1 - (1/5) = 4/5`

Observation: Selectivity ≈ Probability

```
SELECT * FROM people
 WHERE age != 2
```



SELECTIONS – COMPLEX PREDICATES

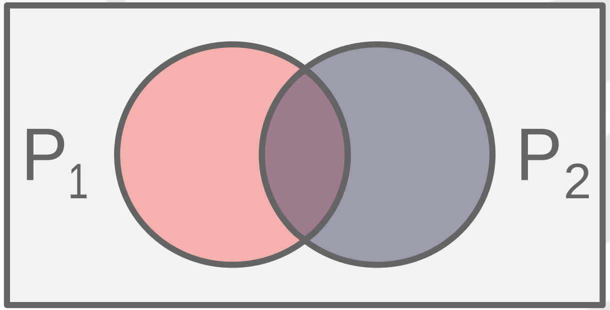
Conjunction:

→ sel(P1 ⋀ P2) = sel(P1) • sel(P2)

→ sel(age=2 ⋀ name LIKE 'A%')

This assumes that the predicates are independent.

```
SELECT * FROM people
 WHERE age = 2
   AND name LIKE 'A%'
```



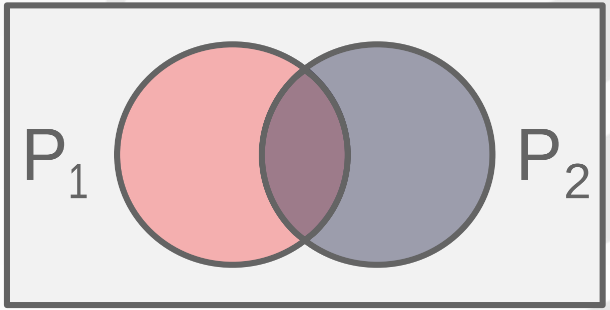
SELECTIONS – COMPLEX PREDICATES

Conjunction:

→ sel(P1 ∧ P2) = sel(P1) • sel(P2)
→ sel(age=2 ∧ name LIKE 'A%')

This assumes that the predicates are independent.

```
SELECT * FROM people
 WHERE age = 2
   AND name LIKE 'A%'
```



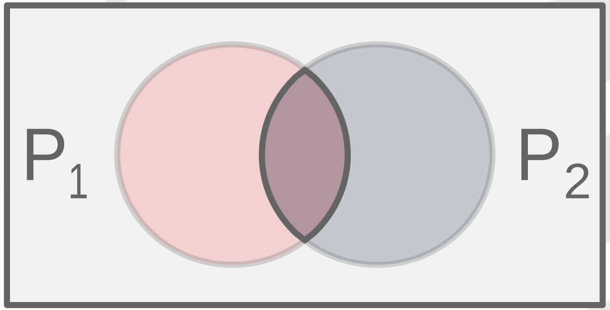
SELECTIONS – COMPLEX PREDICATES

Conjunction:

→ sel(P1 ∧ P2) = sel(P1) • sel(P2)
→ sel(age=2 ∧ name LIKE 'A%')

This assumes that the predicates are independent.

```
SELECT * FROM people
 WHERE age = 2
   AND name LIKE 'A%'
```



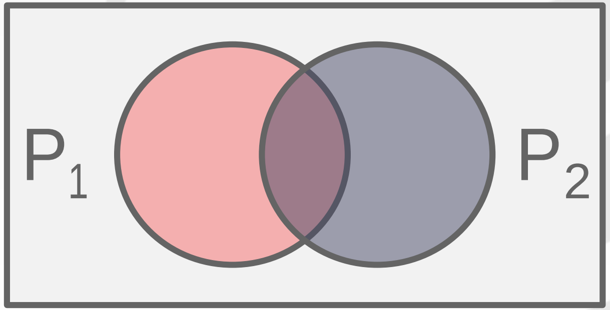
SELECTIONS – COMPLEX PREDICATES

Disjunction:

→ sel(P1 V P2)
  = sel(P1) + sel(P2) - sel(P1VP2)
  = sel(P1) + sel(P2) - sel(P1) • sel(P2)

→ sel(age=2 OR name LIKE 'A%')

```
SELECT * FROM people
 WHERE age = 2
    OR name LIKE 'A%'
```



This again assumes that the selectivities are independent.

SELECTIONS – COMPLEX PREDICATES

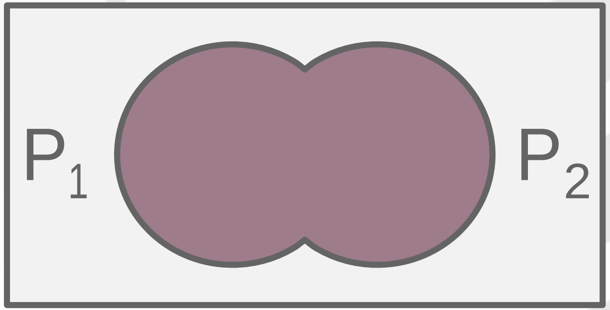
Disjunction:

→ sel(P1 ⋁ P2)
 = sel(P1) + sel(P2) - sel(P1⋁P2)
 = sel(P1) + sel(P2) - sel(P1) • sel(P2)

→ sel(age=2 OR name LIKE 'A%')

```
SELECT * FROM people
 WHERE age = 2
    OR name LIKE 'A%'
```

This again assumes that the selectivities are independent.



RESULT SIZE ESTIMATION FOR JOINS

Given a join of **R** and **S**, what is the range of possible result sizes in # of tuples?

In other words, for a given tuple of **R**, how many tuples of **S** will it match?

RESULT SIZE ESTIMATION FOR JOINS

General case: $R_{cols} \cap S_{cols} = \{A\}$ where **A** is not a key for either table.

→ Match each **R**-tuple with **S**-tuples:
$estSize \approx N_R \cdot N_S / V(A,S)$

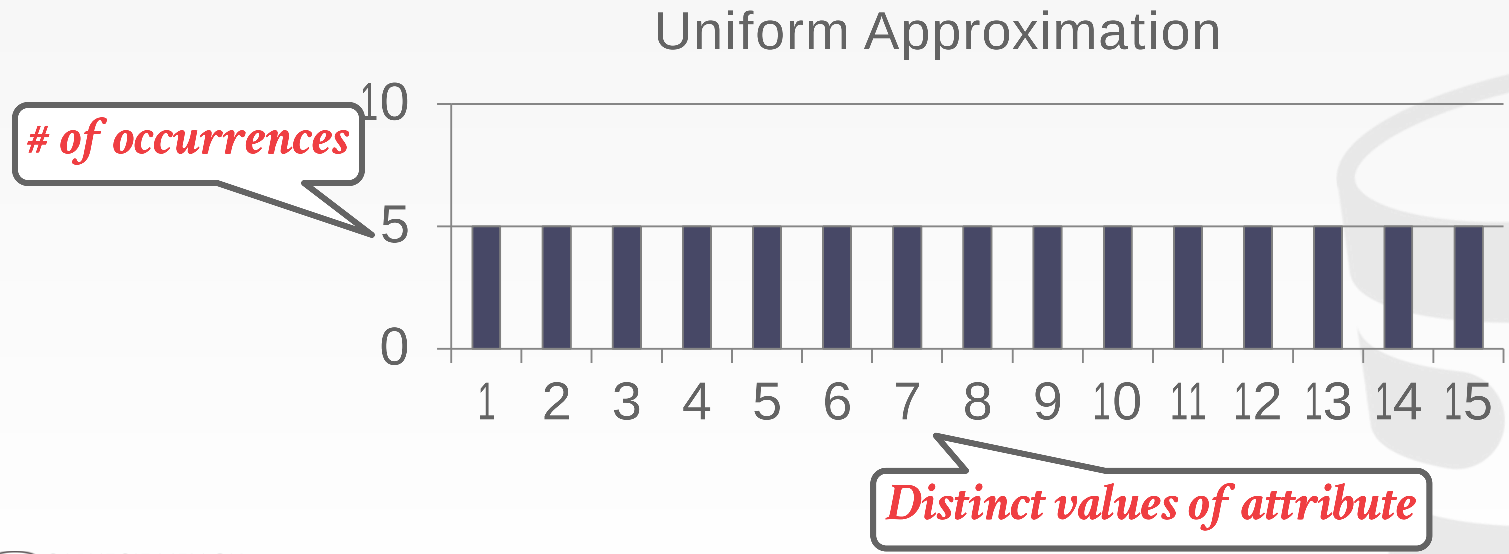
→ Symmetrically, for **S**:
$estSize \approx N_R \cdot N_S / V(A,R)$

Overall:

→ $estSize \approx N_R \cdot N_S / max(\{V(A,S), V(A,R)\})$

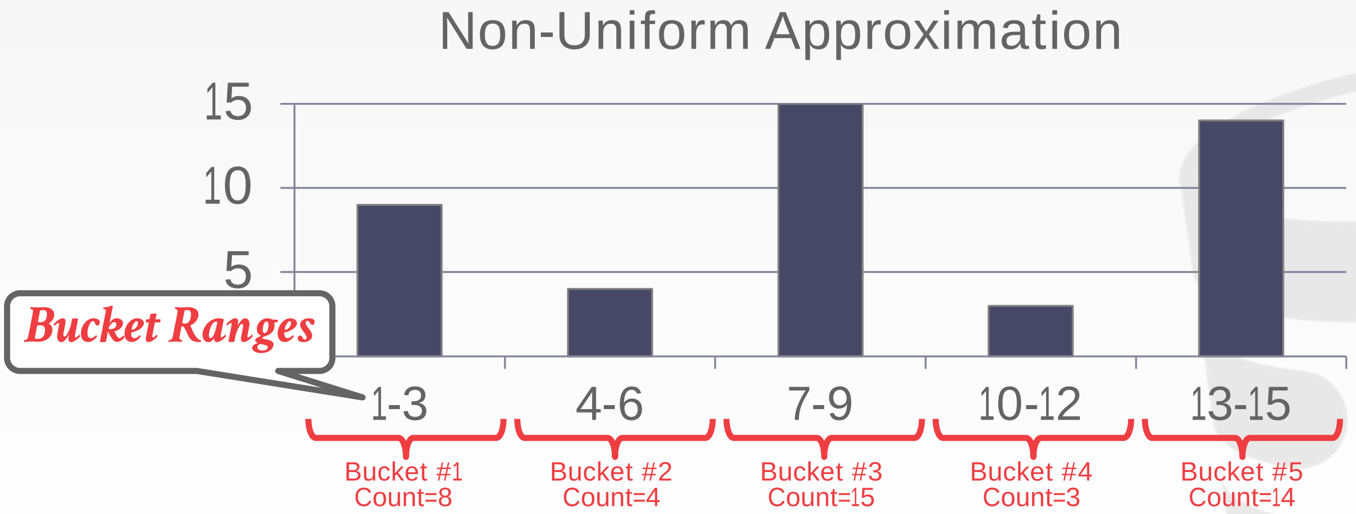
COST ESTIMATIONS

Our formulas are nice but we assume that data values are uniformly distributed.



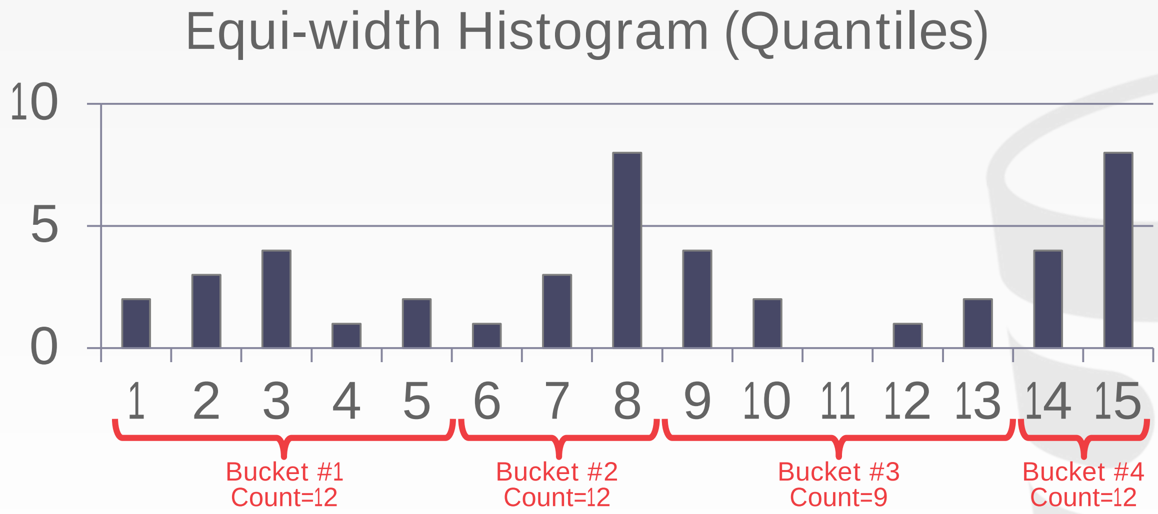
COST ESTIMATIONS

Our formulas are nice but we assume that data values are uniformly distributed.



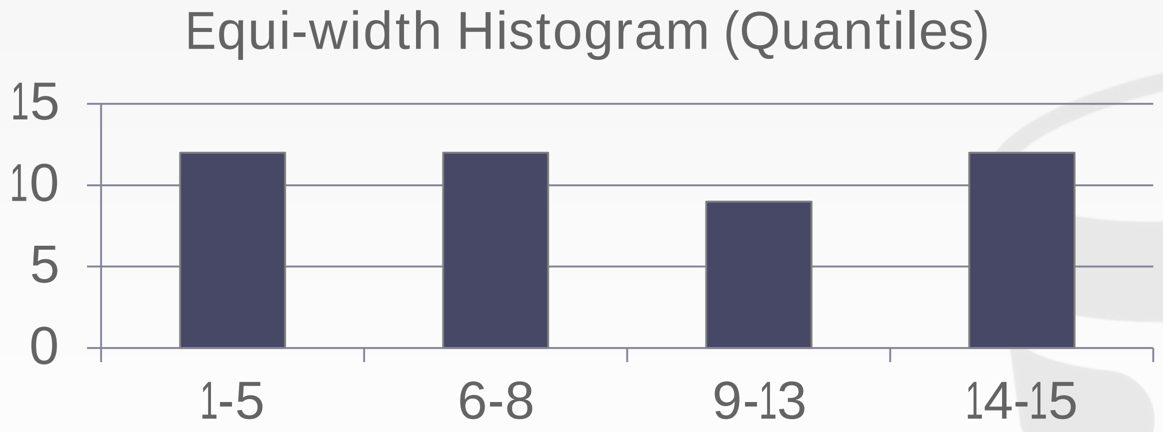
HISTOGRAMS WITH QUANTILES

A histogram type wherein the "spread" of each bucket is same.



HISTOGRAMS WITH QUANTILES

A histogram type wherein the "spread" of each bucket is same.



SAMPLING

Modern DBMSs also collect samples from tables to estimate selectivities.

Update samples when the underlying tables changes significantly.

```
SELECT AVG(age)
  FROM people
 WHERE age > 50
```

id	name	age	status
1001	Obama	56	Rested
1002	Kanye	40	Weird
1003	Tupac	25	Dead
1004	Bieber	23	Crunk
1005	Andy	37	Lit

⋮

1 billion tuples

SAMPLING

Modern DBMSs also collect samples from tables to estimate selectivities.

Update samples when the underlying tables changes significantly.

```
SELECT AVG(age)
  FROM people
 WHERE age > 50
```

id	name	age	status
1001	Obama	56	Rested
1002	Kanye	40	Weird
1003	Tupac	25	Dead
1004	Bieber	23	Crunk
1005	Andy	37	Lit

⋮

1 billion tuples

Table Sample

1001	Obama	56	Rested
1003	Tupac	25	Dead
1005	Andy	37	Lit

sel(age>50) =

SAMPLING

Modern DBMSs also collect samples from tables to estimate selectivities.

Update samples when the underlying tables changes significantly.

```
SELECT AVG(age)
  FROM people
 WHERE age > 50
```

id	name	age	status
1001	Obama	56	Rested
1002	Kanye	40	Weird
1003	Tupac	25	Dead
1004	Bieber	23	Crunk
1005	Andy	37	Lit

⋮

1 billion tuples

Table Sample

1001	Obama	56	Rested
1003	Tupac	25	Dead
1005	Andy	37	Lit

sel(age>50) = 1/3

OBSERVATION

Now that we can (roughly) estimate the selectivity of predicates, what can we actually do with them?

QUERY OPTIMIZATION

After performing rule-based rewriting, the DBMS will enumerate different plans for the query and estimate their costs.

→ Single relation.
→ Multiple relations.
→ Nested sub-queries.

It chooses the best plan it has seen for the query after exhausting all plans or some timeout.

SINGLE-RELATION QUERY PLANNING

Pick the best access method.

→ Sequential Scan
→ Binary Search (clustered indexes)
→ Index Scan

Simple heuristics are often good enough for this.

OLTP queries are especially easy.

OLTP QUERY PLANNING

Query planning for OLTP queries is easy because they are **sargable**.

→ **S**earch **Arg**ument **Able**

→ It is usually just picking the best index.

→ Joins are almost always on foreign key relationships with a small cardinality.

→ Can be implemented with simple heuristics.

MULTI-RELATION QUERY PLANNING

As number of joins increases, number of alternative plans grows rapidly

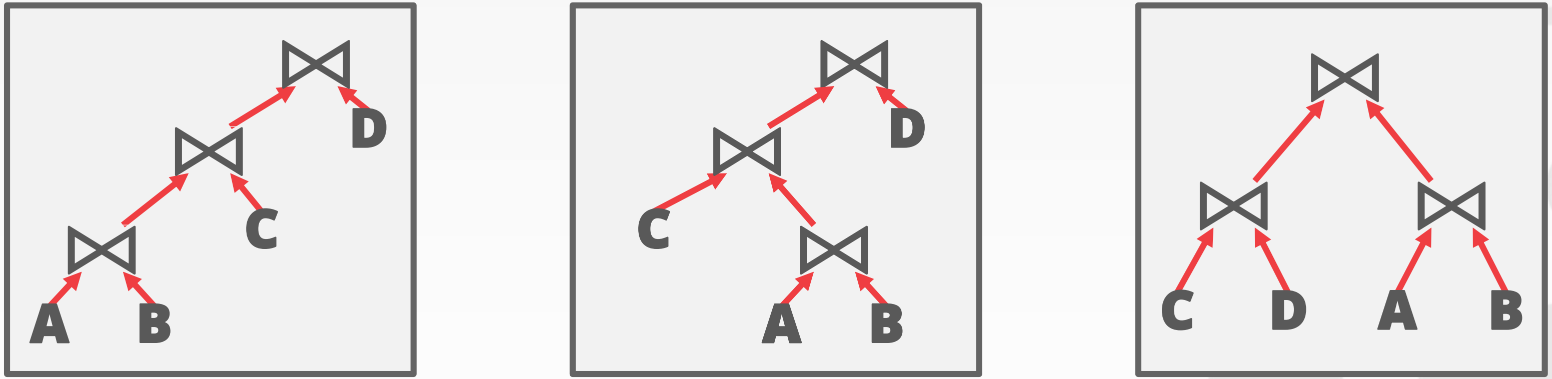
→ We need to restrict search space.

Fundamental decision in System R: only left-deep join trees are considered.

→ Modern DBMSs do not always make this assumption anymore.

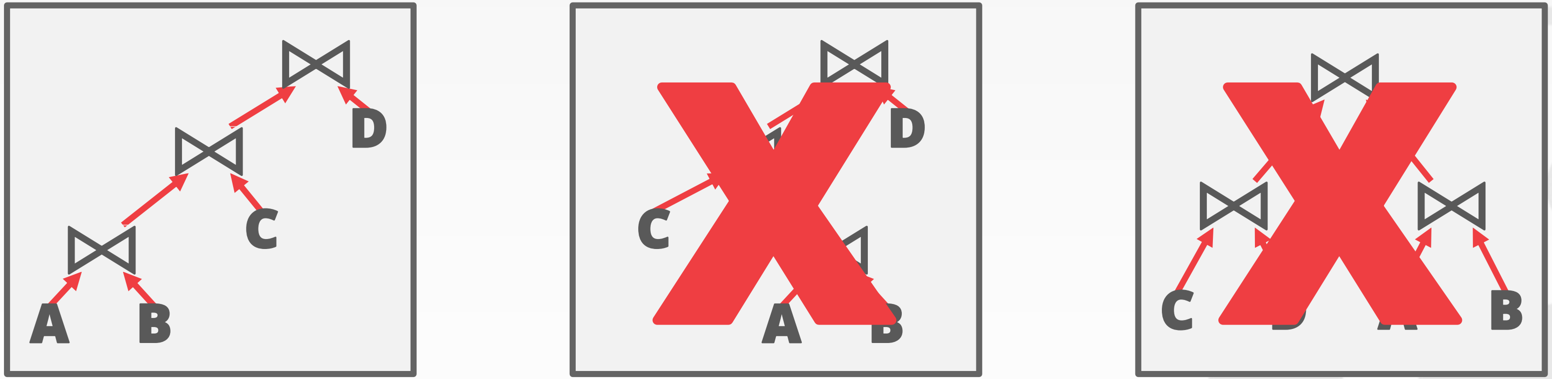
MULTI-RELATION QUERY PLANNING

Fundamental decision in **System R**: Only consider left-deep join trees.



MULTI-RELATION QUERY PLANNING

Fundamental decision in **System R**: Only consider left-deep join trees.



MULTI-RELATION QUERY PLANNING

Fundamental decision in **System R**: Only consider left-deep join trees.

Allows for fully pipelined plans where intermediate results are not written to temp files.
→ Not all left-deep trees are fully pipelined.

MULTI-RELATION QUERY PLANNING

Enumerate the orderings
→ Example: Left-deep tree #1, Left-deep tree #2…

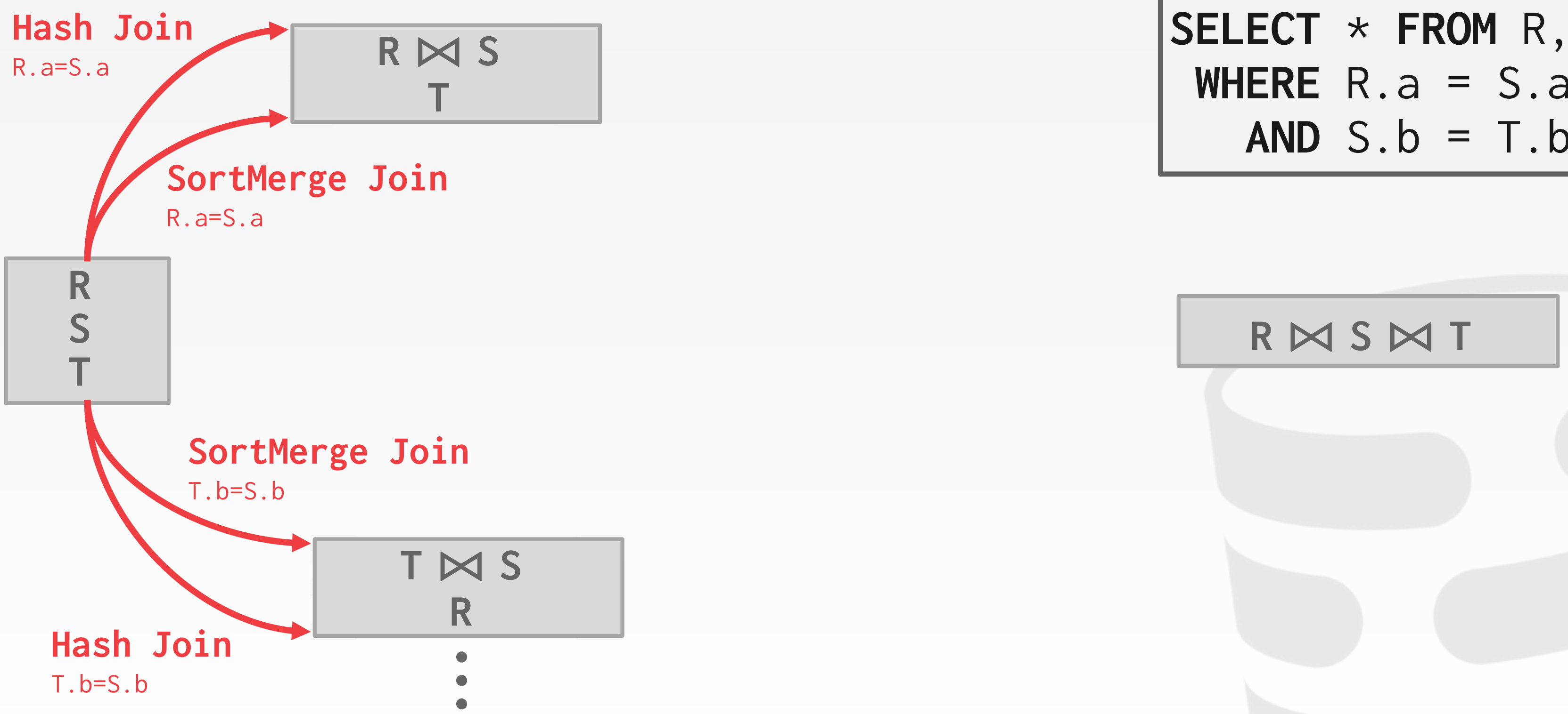
Enumerate the plans for each operator
→ Example: Hash, Sort-Merge, Nested Loop…

Enumerate the access paths for each table
→ Example: Index #1, Index #2, Seq Scan…

Use **dynamic programming** to reduce the number of cost estimations.

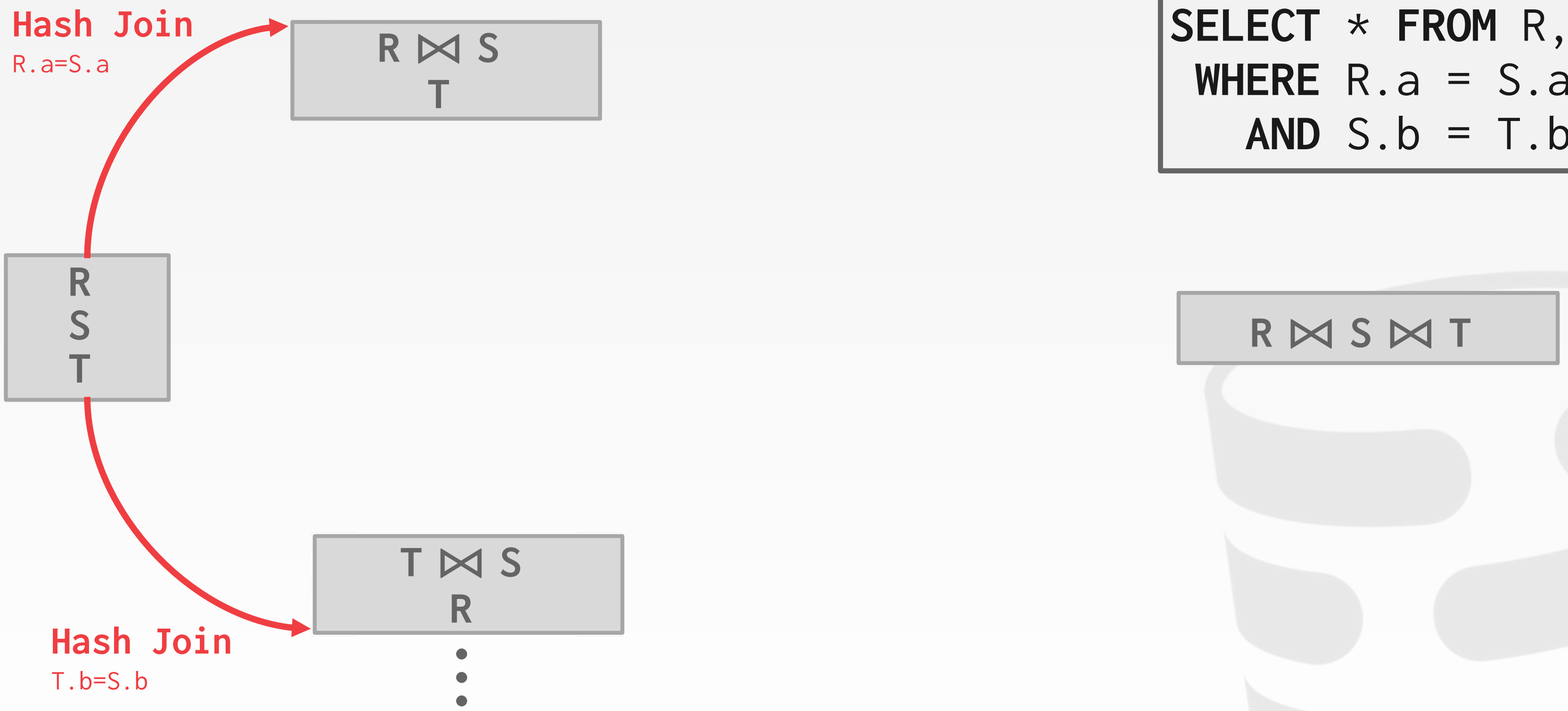
DYNAMIC PROGRAMMING

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



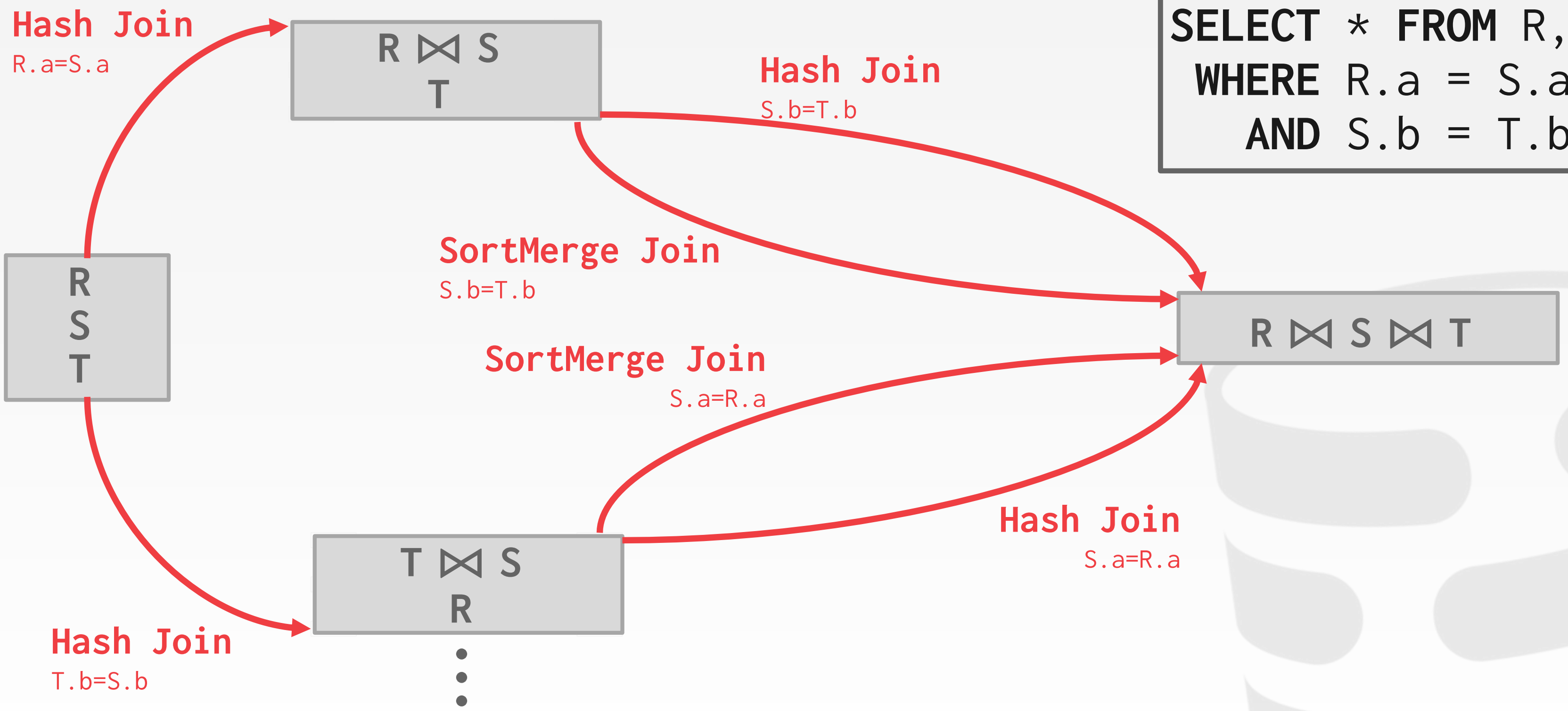
DYNAMIC PROGRAMMING

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



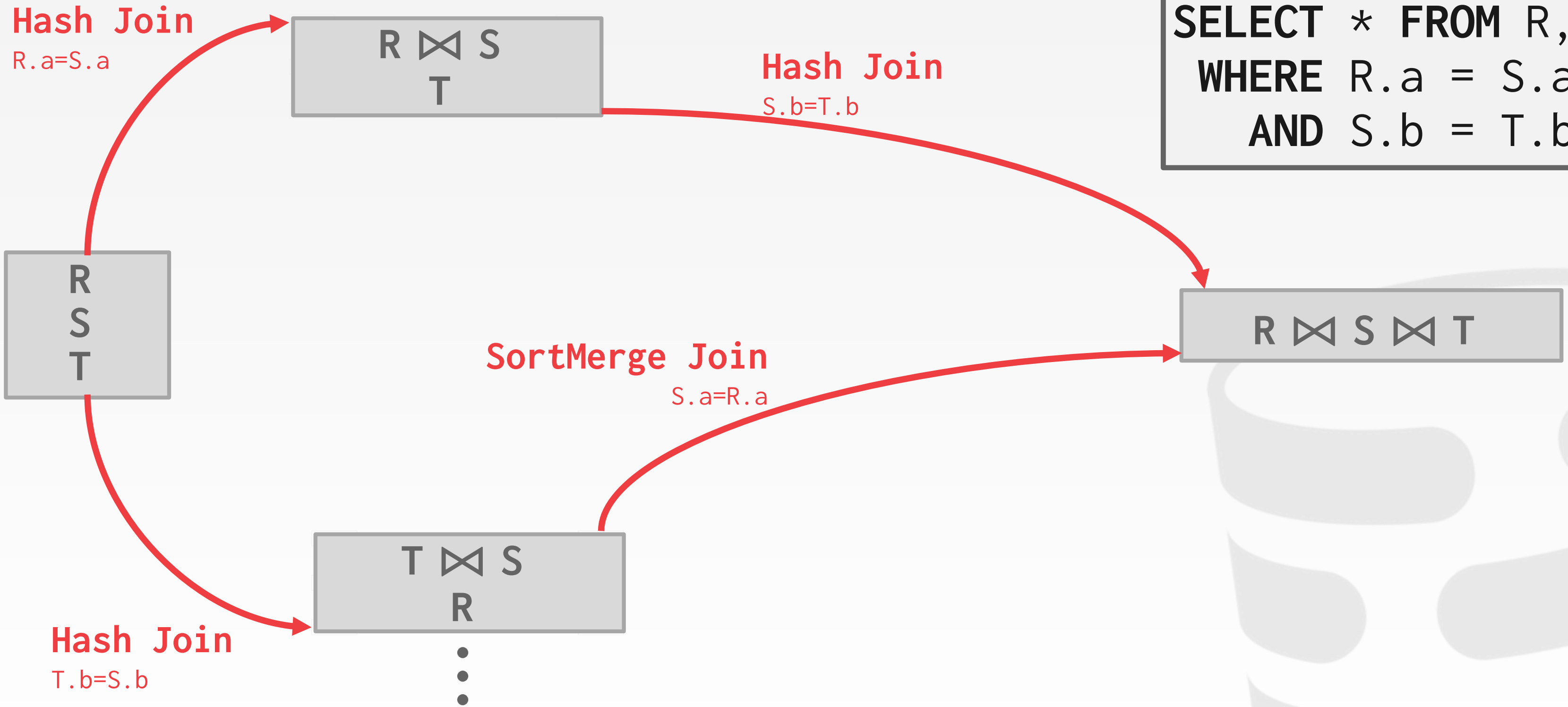
DYNAMIC PROGRAMMING

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



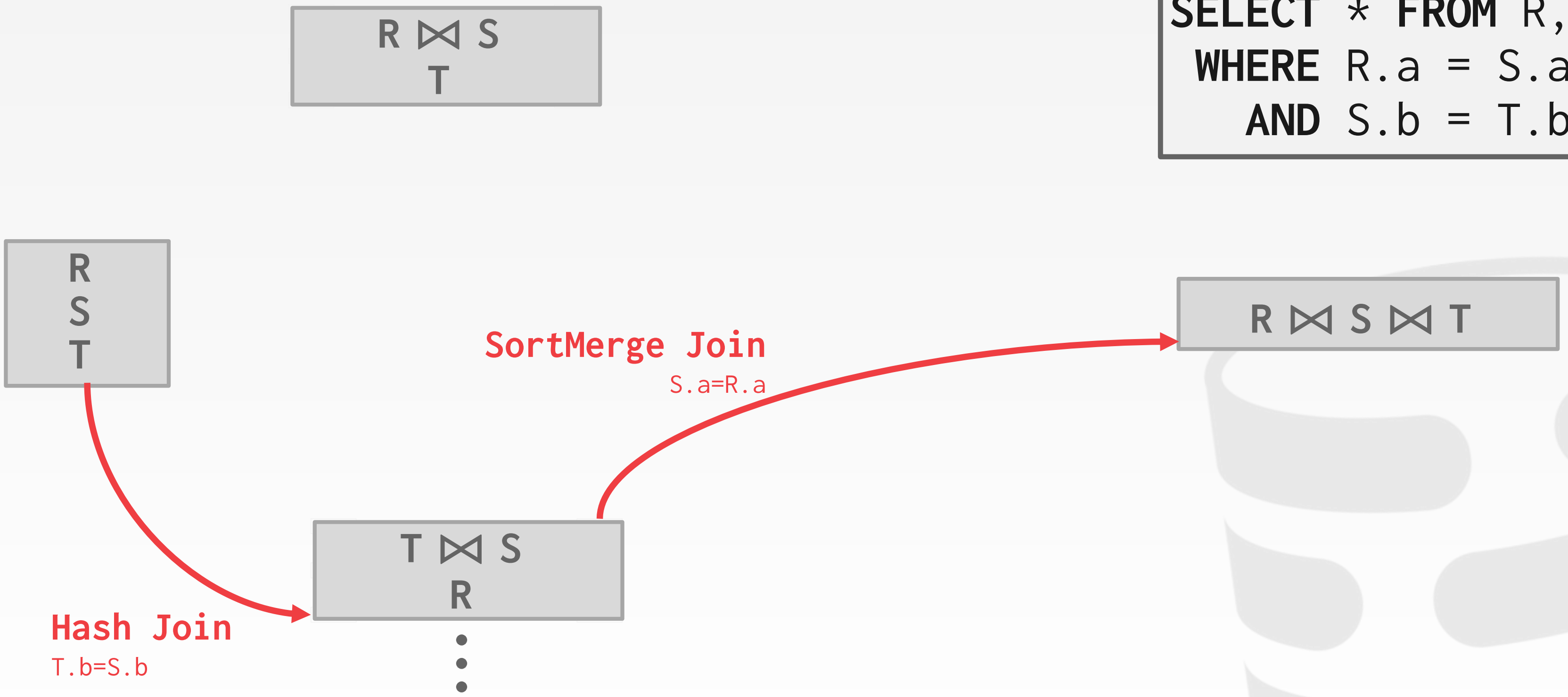
DYNAMIC PROGRAMMING

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



DYNAMIC PROGRAMMING

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



CANDIDATE PLAN EXAMPLE

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

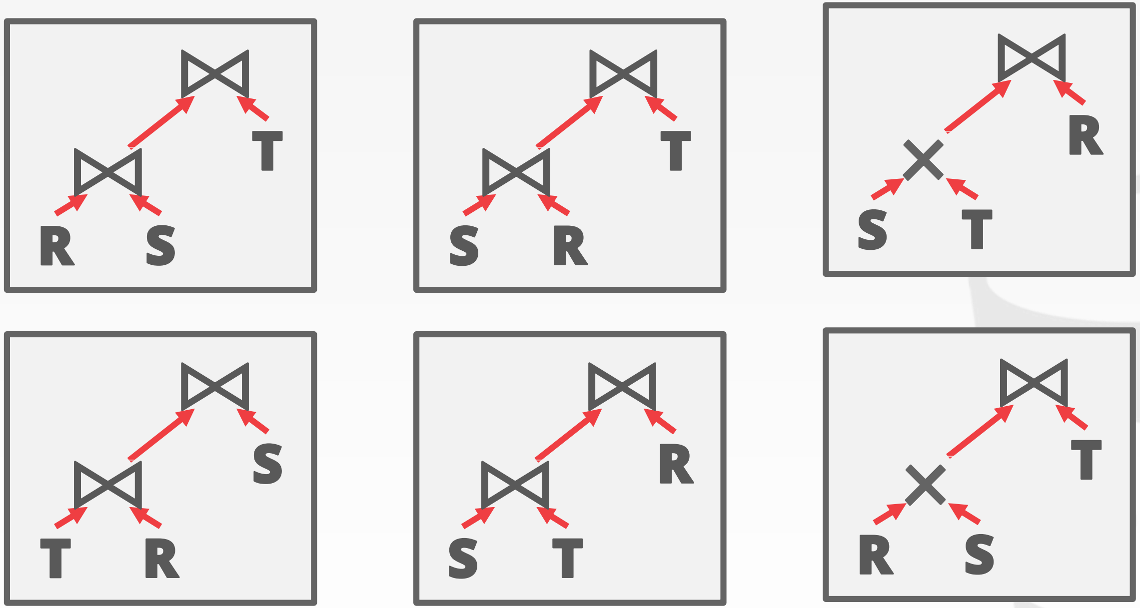
How to generate plans for search algorithm:

→ Enumerate relation orderings
→ Enumerate join algorithm choices
→ Enumerate access method choices

No real DBMSs does it this way.
It's actually more messy…

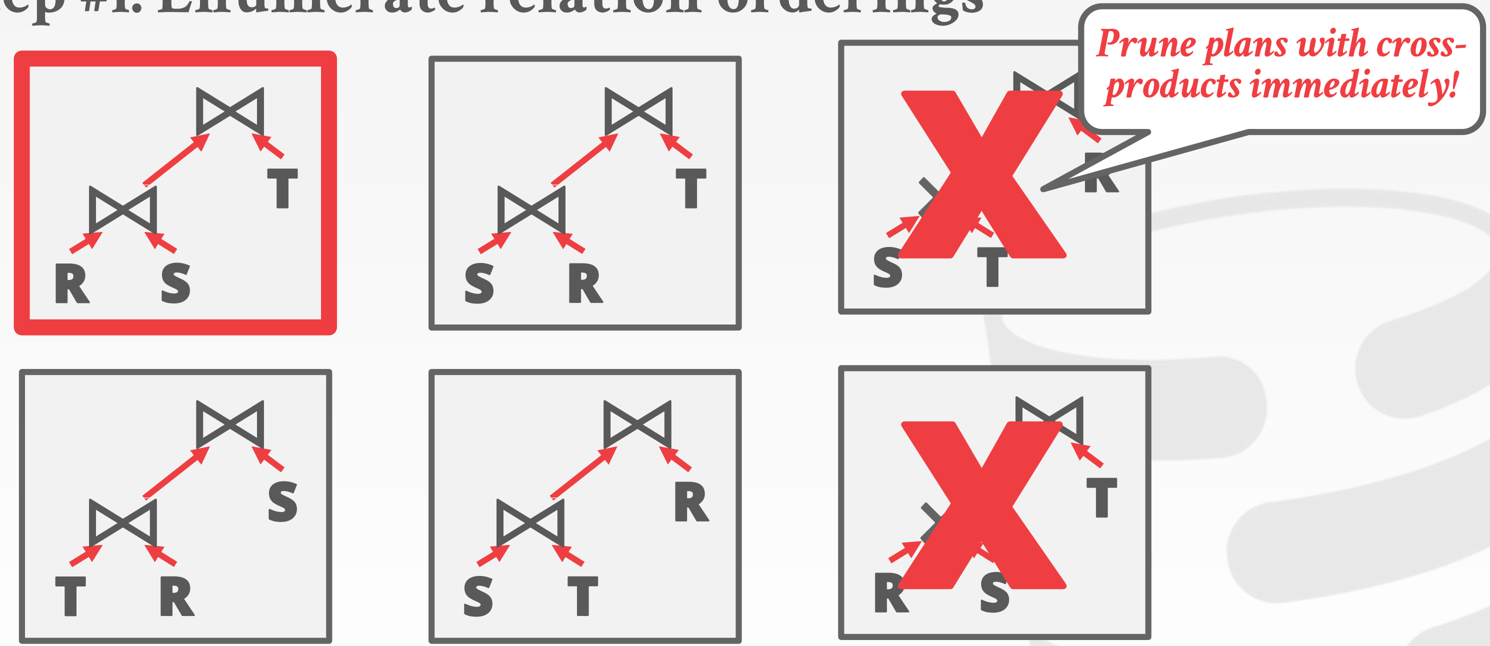
CANDIDATE PLANS

Step #1: Enumerate relation orderings



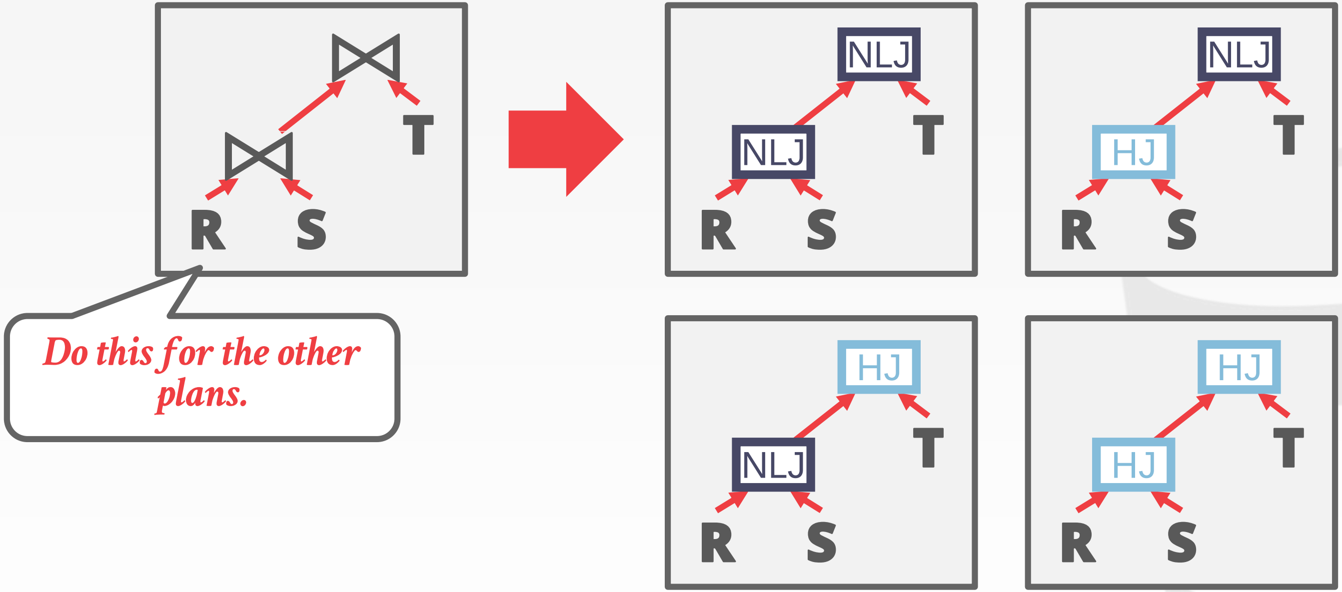
CANDIDATE PLANS

Step #1: Enumerate relation orderings



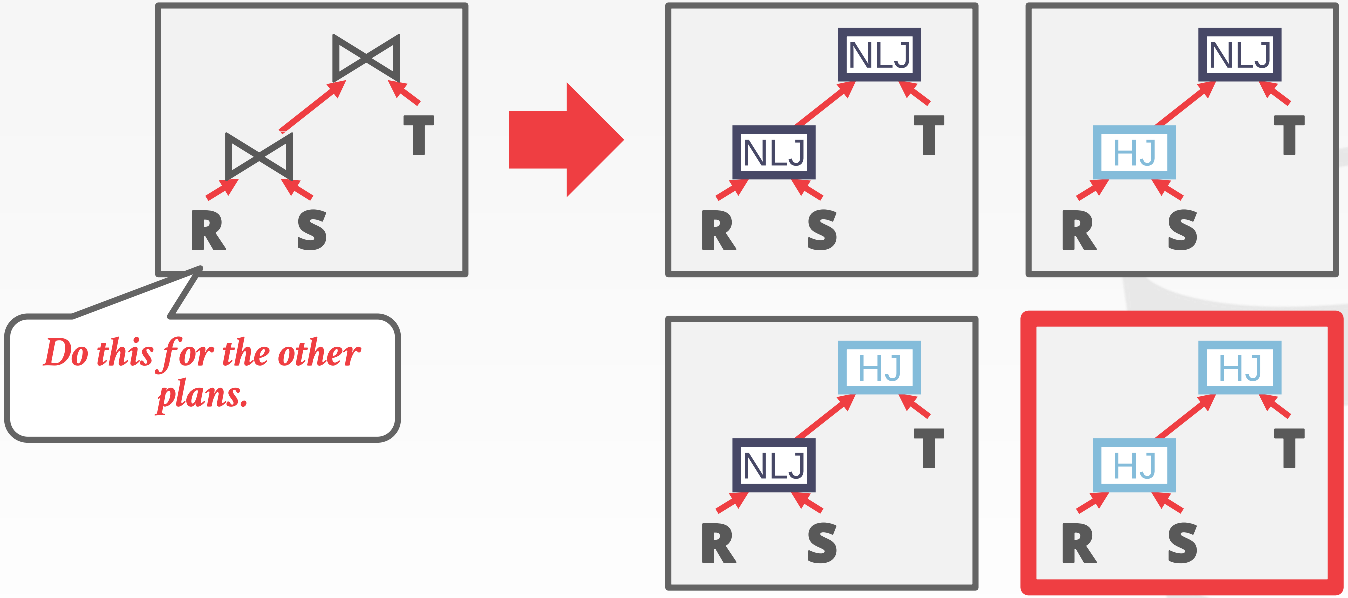
CANDIDATE PLANS

Step #2: Enumerate join algorithm choices



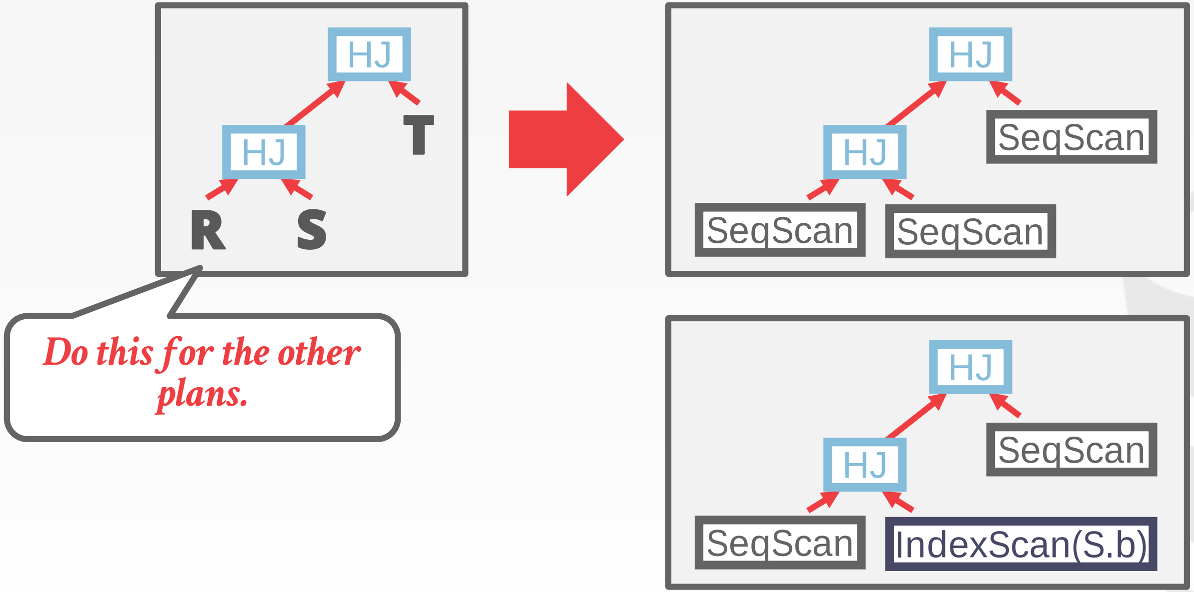
CANDIDATE PLANS

Step #2: Enumerate join algorithm choices



CANDIDATE PLANS

Step #3: Enumerate access method choices



NESTED SUB-QUERIES

The DBMS treats nested sub-queries in the where clause as functions that take parameters and return a single value or set of values.

Two Approaches:

→ Rewrite to de-correlate and/or flatten them
→ Decompose nested query and store result to temporary table

NESTED SUB-QUERIES: REWRITE

```
SELECT name FROM sailors AS S
 WHERE EXISTS (
    SELECT * FROM reserves AS R
     WHERE S.sid = R.sid
       AND R.day = '2018-10-15'
 )
```

```
SELECT name
  FROM sailors AS S, reserves AS R
 WHERE S.sid = R.sid
   AND R.day = '2018-10-15'
```

NESTED SUB-QUERIES: DECOMPOSE

```
SELECT S.sid, MIN(R.day)
  FROM sailors S, reserves R, boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = 'red'
   AND S.rating = (SELECT MAX(S2.rating)
                     FROM sailors S2)
 GROUP BY S.sid
HAVING COUNT(*) > 1
```

For each sailor with the highest rating (over all sailors) and at least two reservations for red boats, find the sailor id and the earliest date on which the sailor has a reservation for a red boat.

DECOMPOSING QUERIES

For harder queries, the optimizer breaks up queries into blocks and then concentrates on one block at a time.

Sub-queries are written to a temporary table that are discarded after the query finishes.

DECOMPOSING QUERIES

```
SELECT S.sid, MIN(R.day)
  FROM sailors S, reserves R, boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = 'red'
   AND S.rating = (SELECT MAX(S2.rating)
                     FROM sailors S2)
 GROUP BY S.sid
HAVING COUNT(*) > 1
```

Nested Block

DECOMPOSING QUERIES

```
SELECT MAX(rating) FROM sailors
```

```
SELECT S.sid, MIN(R.day)
  FROM sailors S, reserves R, boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = 'red'
   AND S.rating = ###

 GROUP BY S.sid
HAVING COUNT(*) > 1
```

Nested Block

DECOMPOSING QUERIES

```
SELECT MAX(rating) FROM sailors
```

```
SELECT S.sid, MIN(R.day)
  FROM sailors S, reserves R, boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = 'red'
   AND S.rating = ###

 GROUP BY S.sid
HAVING COUNT(*) > 1
```

DECOMPOSING QUERIES

```
SELECT MAX(rating) FROM sailors
```

```
SELECT S.sid, MIN(R.day)
  FROM sailors S, reserves R, boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = 'red'
   AND S.rating = ###

 GROUP BY S.sid
HAVING COUNT(*) > 1
```

Outer Block

CONCLUSION

Filter early as possible.

Selectivity estimations
- → Uniformity
- → Independence
- → Histograms
- → Join selectivity

Dynamic programming for join orderings

Rewrite nested queries

Query optimization is really hard…

Midterm Exam

Who: You

What: Midterm Exam

When: Wed Oct 17th 12:00pm - 1:20pm

Where: Posner Mellon Auditorium

Why: https://youtu.be/xgMiaIPxSlc

MIDTERM

What to bring:

→ CMU ID

→ Calculator

→ One 8.5x11" page of notes (double-sided)

What not to bring:

→ Live animals

→ Your wet laundry

MIDTERM

Covers up to Joins (inclusive).

→ Closed book, one sheet of notes (double-sided)

→ Please email Andy if you need special accommodations.

https://15445.courses.cs.cmu.edu/fall2018/midterm-guide.html

RELATIONAL MODEL

Integrity Constraints

Relation Algebra

SQL

Basic operations:
- → SELECT / INSERT / UPDATE / DELETE
- → WHERE predicates
- → Output control

More complex operations:
- → Joins
- → Aggregates
- → Common Table Expressions

STORAGE

Buffer Management Policies

→ LRU / MRU / CLOCK

On-Disk File Organization

→ Heaps
→ Linked Lists

Page Layout

→ Slotted Pages
→ Log-Structured

HASHING

Static Hashing
→ Linear Probing
→ Robin Hood
→ Cuckoo Hashing

Dynamic Hashing
→ Extendible Hashing
→ Linear Hashing

Comparison with B+Trees

TREE INDEXES

B+Tree
→ Insertions / Deletions
→ Splits / Merges
→ Difference with B-Tree
→ Latch Crabbing / Coupling

Radix Trees

Skip Lists

SORTING

Two-way External Merge Sort

General External Merge Sort

Cost to sort different data sets with different number of buffers.

QUERY PROCESSING

Processing Models
→ Advantages / Disadvantages

Join Algorithms
→ Nested Loop
→ Sort-Merge
→ Hash

NEXT CLASS

Parallel Query Execution