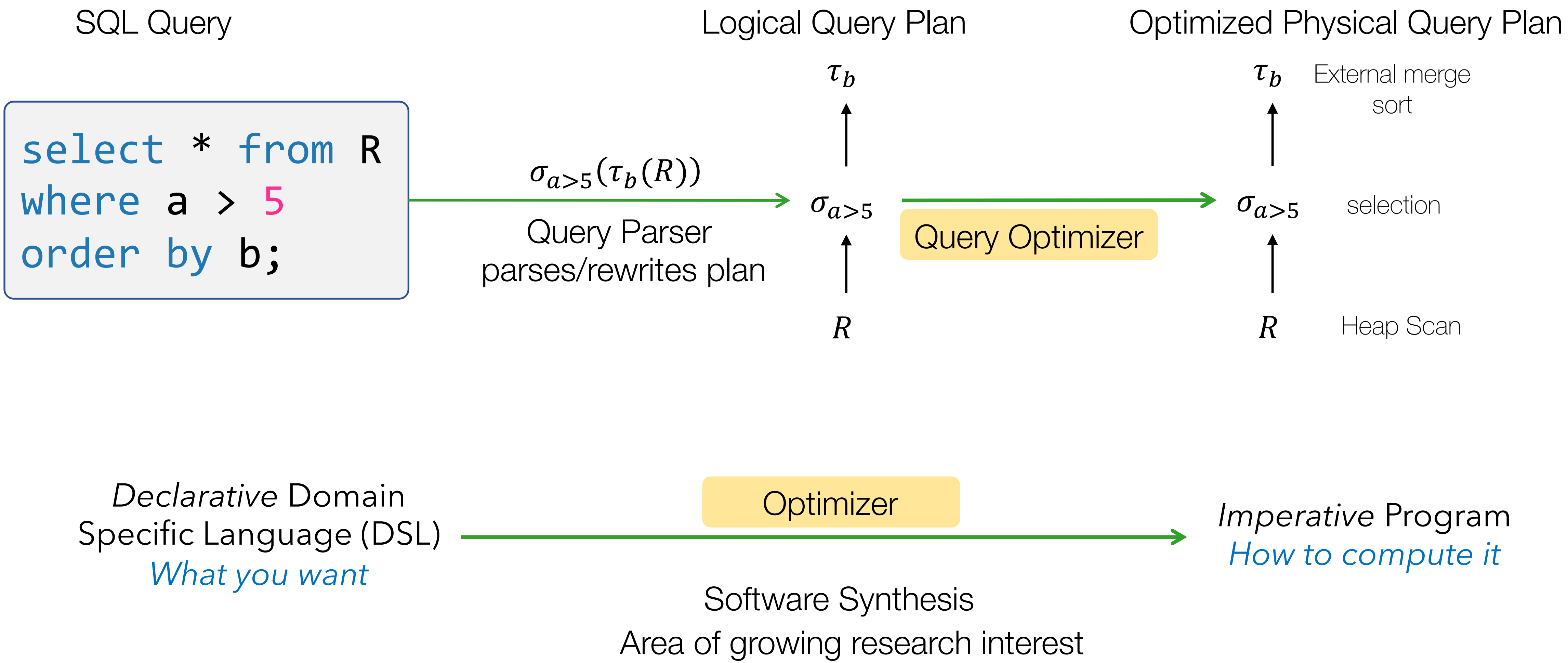
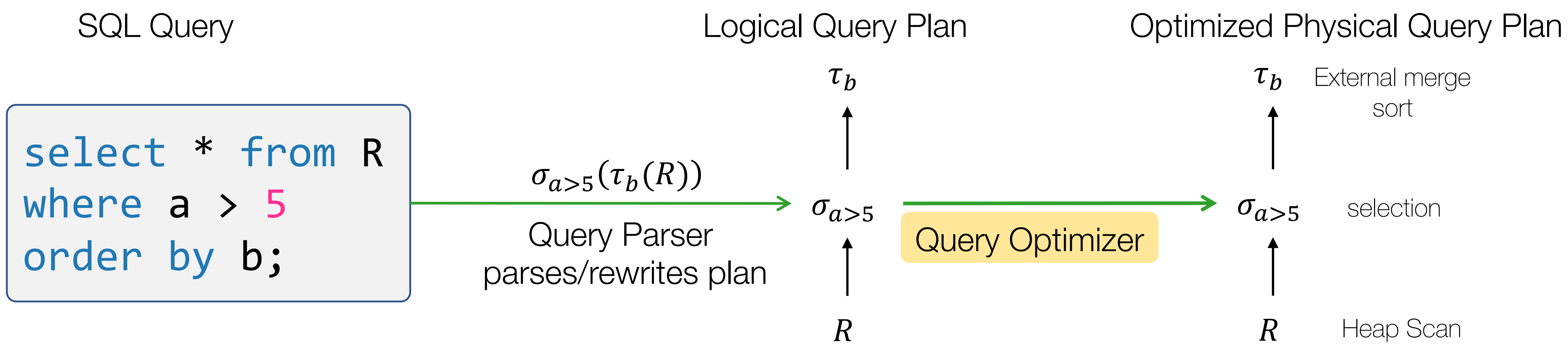
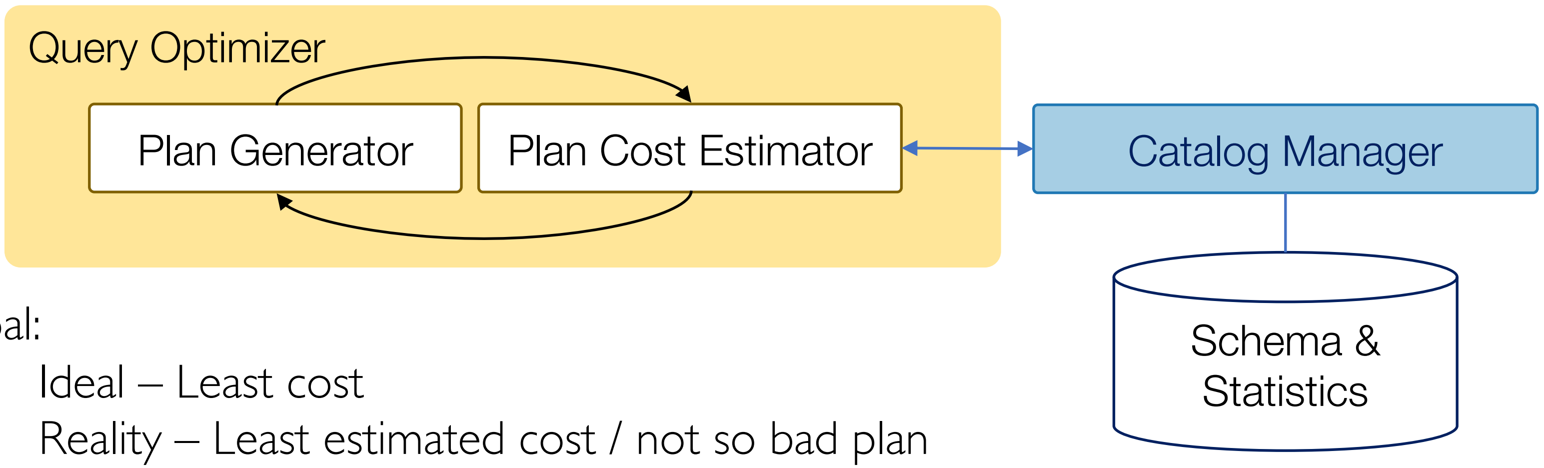
Query Optimizers



SQL Query
select * from R
where a > 5
order by b;
$\sigma_{a>5}(\tau_b(R))$
Query Parser
parses/rewrites plan
Logical Query Plan
τ_b
$\sigma_{a>5}$
R
Query Optimizer
Optimized Physical Query Plan
τ_b
External merge sort
$\sigma_{a>5}$
selection
R
Heap Scan
Declarative Domain Specific Language (DSL)
What you want
Optimizer
Software Synthesis
Area of growing research interest
Imperative Program
How to compute it





- Goal:
  - Ideal – Least cost
  - Reality – Least estimated cost / not so bad plan
- Focus on the 1979 **System R / "Selinger"** Optimizer
  - Cost: #IOs + CPU-factor $\times$#tuples;

The Dimensions of Query Optimization

	Abstraction	Concrete Formulation	Challenges	The Selinger Solution
Plan Space	What are all the possible plans to consider?	How to enumerate the space? • Relational Equivalences • Physical Equivalences	Massive Space! Catalan number $C_n \approx 4^n$ is the number of full binary trees with $n+1$ base tables; n joins $\approx 4^n$ plans!	Prune (Heuristics) • Avoid plans with cartesian products • Consider only left-deep join trees
Cost Estimation	How to compare two plans? Which plan is better?	What is the cost of a plan?	How to estimate the cost of a plan without executing it?	Update statistics periodically in a catalog Use crude but practical formulae • Selectivity to estimate intermediate results
Search Strategy	How do we find the lowest-cost plan?	How to search efficiently without full enumeration?	Structure search to avoid sub-optimal plan spaces.	Use *Dynamic Programming* + maintain "interesting orders" • Assume principle of optimality

Plan Space

Plan Space	What are all the possible plans to consider?	How to enumerate the space? • Relational Equivalences • Physical Equivalences	Massive Space! n joins $\approx 4^n$ plans!	Prune (Heuristics) • Avoid plans with cross products • Consider only left-deep join trees

Selections

Cascade $\sigma_{c_1 \wedge c_2 \wedge \cdots \wedge c_k}(R) \equiv \sigma_{c_1}\left(\sigma_{c_2}\left(\ldots\left(\sigma_{c_k}(R)\right)\ldots\right)\right)$

Reorder $\sigma_{c_1}\left(\sigma_{c_2}(R)\right) \equiv \sigma_{c_2}\left(\sigma_{c_1}(R)\right)$

Projections

Cascade $\pi_{\mathbb{A}_1}(R) \equiv \pi_{\mathbb{A}_1}\left(\pi_{\mathbb{A}_2}\left(\ldots\left(\pi_{\mathbb{A}_k}(R)\right)\ldots\right)\right)$, if $\mathbb{A}_1 \subseteq \cdots \subseteq \mathbb{A}_k$

Joins & Cross-products

Commutative

$R \times S \equiv S \times R$

$R \bowtie_{x=x} S \equiv S \bowtie_{x=x} R$

Associative

$R \times (S \times T) \equiv (R \times S) \times T$

$R \bowtie_{x=x} (S \bowtie_{x=x} T) \equiv (R \bowtie_{x=x} S) \bowtie_{x=x} T$

WARNING!

$R(a,b); S(a,c); T(b,d)$

$(R \bowtie_{a=a} S) \bowtie_{b=b} T$

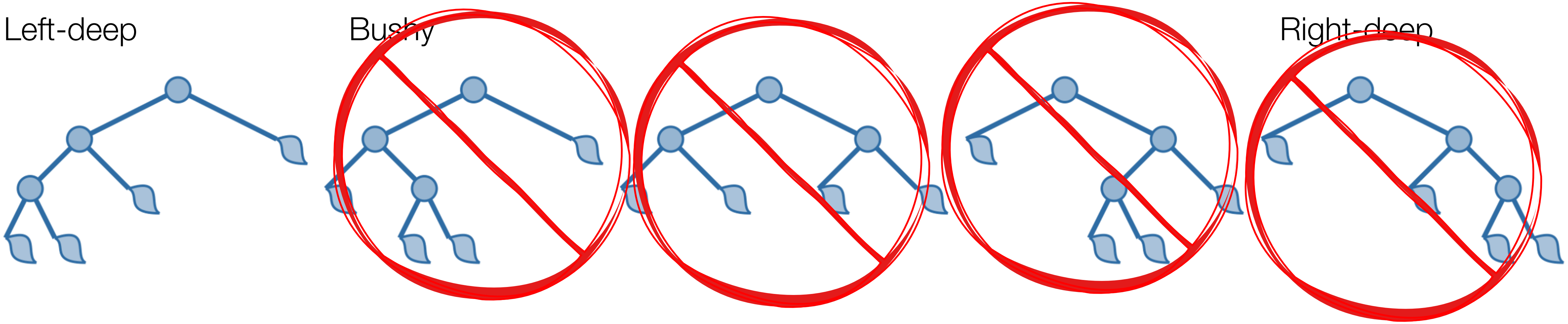
$\not\equiv R \bowtie_{a=a} (S \bowtie_{b=b} T)$ S doesn't have b!

$\not\equiv R \bowtie_{a=a} (S \times T)$ We lost b=b!

$\equiv R \bowtie_{a=a \wedge b=b} (S \times T)$ We replaced a join with a product!

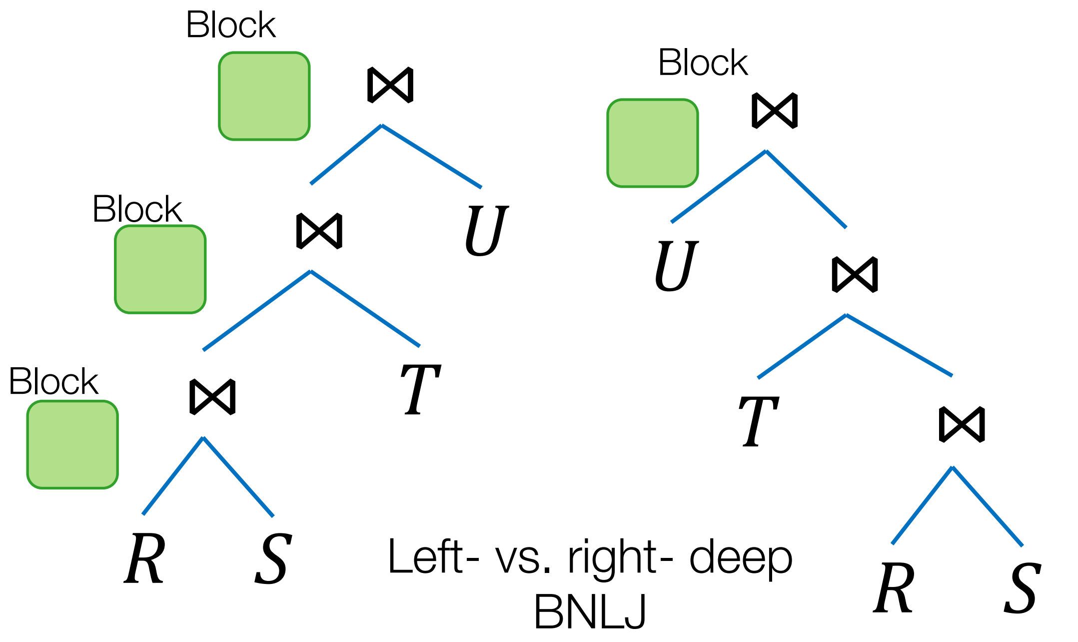
Relational Equivalences

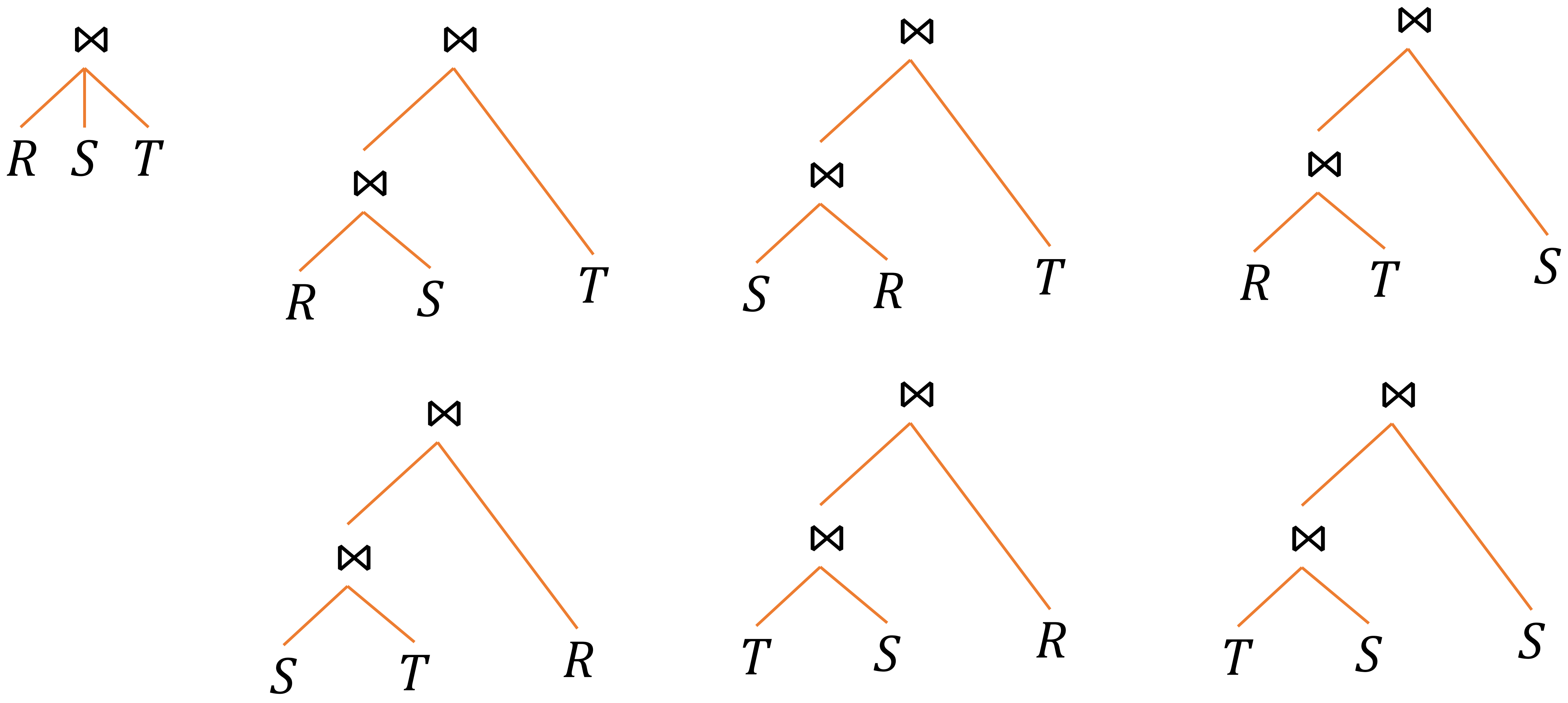
Relational Equivalences – Join Pruning Heuristics



Why left deep?

Left-deep trees allow us to generate pipelined plans where intermediate results are not written to temporary files (e.g. with BNLJ) and buffer pools are not exhausted (e.g. with Hash Join).





# of permutations: $n!$

How many left-deep trees?

Plan Space What are all the possible plans to consider?

How to enumerate the space?
- Relational Equivalences
- Physical Equivalences

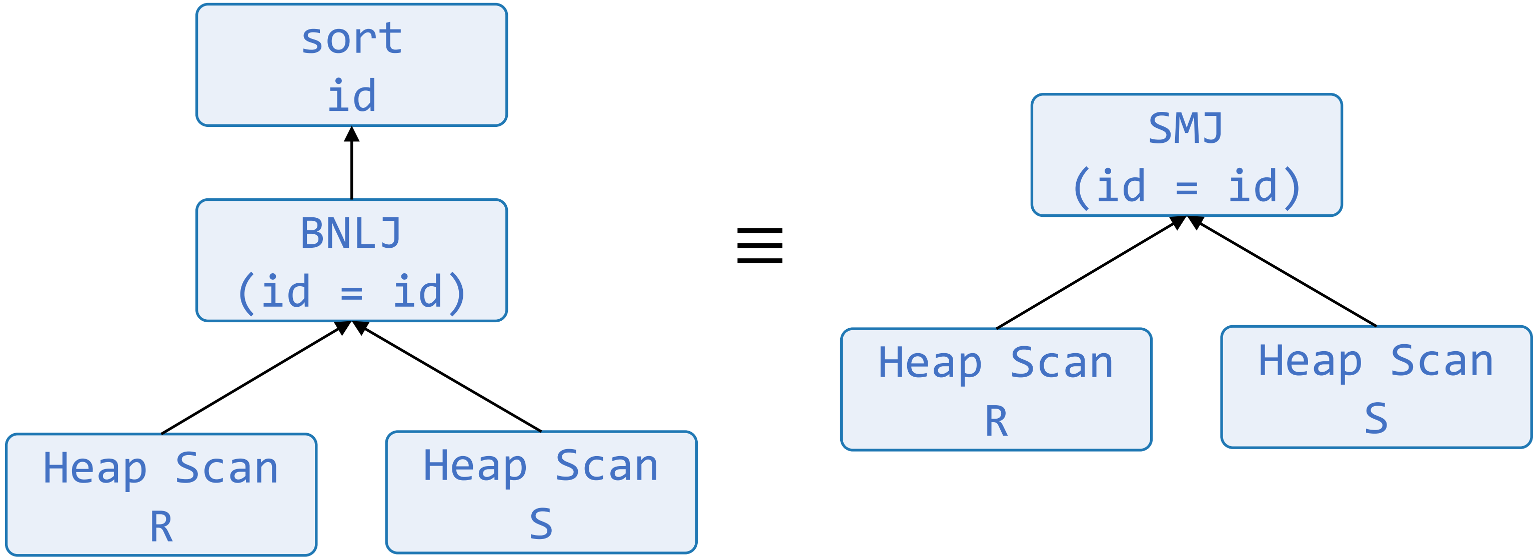
Massive Space! n joins $\approx 4^n$ plans!

Prune (Heuristics)
- Avoid plans with cross products
- Consider only left-deep join trees

Base table access with *selections* and *projections*
- Heap scan
- Index scan

Equijoins
- Page Nested Loop
- Block Nested Loop
- Index Nested Loop
- Sort-Merge Join
- Grace Hash Join

Theta-Joins
- Block Nested Loop



Physical Equivalences

Cost Estimation

Cost Estimation	How to compare two plans? Which plan is better?	What is the cost of a plan?	How to estimate the cost of a plan without executing it?	Update statistics periodically in a catalog Use crude but practical formulae • Selectivity to estimate intermediate results

#IO + *CPU-factor* * #tuples

Cost of each operator in plan

- IO cost of sequential scan, index scan, joins, etc., when we know input size
- Catalog keeps track of
  - Base table size (for leaf operators)
  - Index sizes

Estimate result size for each operator

- Operator output is downstream operator's input size
- For selections, and joins, estimate based on how much a selection condition reduces the size of the input table: *Selectivity*.

Cost Estimation

Statistics and Catalogs

Statistic	Meaning
tuples	# of tuples in a table (cardinality)
pages	# of disk pages in a table
Low($A_1, \dots, A_n$)	min value in a column A_i
High($A_1, \dots, A_n$)	max value in a column A_i
Keys($A_1, \dots, A_n$)	# of distinct values in a column A_i
index_height($I_1, \dots, I_k$)	the height of an index I_i
index_pages($I_1, \dots, I_k$)	# of disk pages in an index I_i

- *Catalogs* updated periodically.
  - Too expensive to do on every update
  - Crude estimates anyway!
- Modern systems keep finer-grained info on the distribution of values (e.g. histograms, sketches, etc.)

```
ANALYZE tbl;
select * from pg_stats where tablename = 'R'
```

Result Size Estimation and Selectivity

```
select * from
R, S, ... where
p1 AND p2 …
```

- Maximum result size: product of input sizes (think $R \times S \times \cdots$)
- Each term $p_1, \ldots, p_n$ reduces the input by a factor
  - Reduction Factor = *Selectivity* = |Output|/|Input|
  - Result size = Maximum result size * Selectivity
- Simplifying assumptions
  - *Uniformity*: all values in a table are uniformly distributed
  - *Independence*: predicates are independent
- Selectivity ~ Probability

Selectivity by example

```sql
select * from R
where A = 3527;
```

Equality

$$\text{keys}(A) = 100$$

$$\text{sel} = \frac{1}{\text{keys}(A)} = 0.01$$

(Uniformity)

```sql
select * from R
where A > 3500;
```

Inequality

(Uniformity)

$$\text{Low}(A) = 3000; \text{High}(A) = 4000;$$

$$\text{sel} = \frac{\text{High}(A) - v}{\text{High}(A) - \text{Low}(A) + 1} \approx 0.5$$

```sql
select * from R
where A = 3527
AND B = 20;
```

Conjunction

$$\text{keys}(A) = 100; \text{keys}(B) = 10$$

$$\text{sel} = \frac{1}{\text{keys}(A)} \times \frac{1}{\text{keys}(B)} = 0.001$$

(Independence)

```sql
select * from R
where A = 3527
OR B = 20;
```

Disjunction (Don't Double Count!)

(Independence)

$$\text{keys}(A) = 100; \text{keys}(B) = 10$$

$$\text{sel} = \frac{1}{\text{keys}(A)} + \frac{1}{\text{keys}(B)} - \frac{1}{\text{keys}(A)} \times \frac{1}{\text{keys}(B)} = 0.109$$

```sql
select * from R
where B = C;
```

Column Equality

(Independence)

$$\text{keys}(B) = 10 = \{a, b, c, d, e, f, g, h, i, j\};$$
$$\text{keys}(C) = 2 = \{a, b\}$$

$$sel = sel(B = a \wedge C = a) + sel(B = b \wedge C = b)$$
$$+ sel(B = c \wedge C = c) + sel(B = d \wedge C = d) + \ldots$$

$$\text{sel} = \frac{1}{2} * \frac{1}{10} + \frac{1}{2} * \frac{1}{10} + 0 * \frac{1}{10} + 0 * \frac{1}{10} + \ldots$$

$$\text{sel} = \frac{1}{10} = \frac{1}{\max(\text{keys}(B), \text{keys}(C))}$$

Selectivity by example

```
select * from
R, S
where R.A = S.A
```

Join Selectivity

$\text{keys}(R.A) = 100; \#\text{tuples} = 500$
$\text{keys}(S.A) = 50; \#\text{tuples} = 200$
Equivalent to Column Equality over $|R| \times |S|$

$$\text{sel} = \frac{1}{\max(\text{keys}(A), \text{keys}(B))}$$

```
select * from R
where A != 3500;
```

NOT

$\text{keys}(A) = 100$

$$\text{sel}(A = 3500) = \frac{1}{100}$$

$\text{sel}(A! = 3500) = 1 - \text{sel}(A = 3500) = 0.99$

What if we don't have any estimates? 1/10 is the Selinger way

Selectivity by example

```
/* default selectivity estimate for equalities such as "A = b" */
#define DEFAULT_EQ_SEL  0.005

/* default selectivity estimate for inequalities such as "A < b" */
#define DEFAULT_INEQ_SEL  0.3333333333333333

/* default selectivity estimate for range inequalities "A > b AND A < c" */
#define DEFAULT_RANGE_INEQ_SEL  0.005

/* default selectivity estimate for multirange inequalities "A > b AND A < c" */
#define DEFAULT_MULTIRANGE_INEQ_SEL 0.005

/* default selectivity estimate for pattern-match operators such as LIKE */
#define DEFAULT_MATCH_SEL   0.005

/* default selectivity estimate for other matching operators */
#define DEFAULT_MATCHING_SEL    0.010

/* default number of distinct values in a table */
#define DEFAULT_NUM_DISTINCT  200
```

Postgres Selectivities

Search Algorithm

Search Strategy	How do we find the lowest-cost plan?	How to search efficiently without full enumeration?	Structure search to avoid sub-optimal plan spaces.	Use *Dynamic Programming* + maintain "interesting orders" • Assume principle of optimality

What is DP? Break down a problem into simpler subproblems assuming the optimal solution to the overall problem depends upon the optimal solution to its subproblems.

Base case ($n = 1$ Relation)

Queries with σ, π, and Group By/Aggregation:

- estimate cost of every available access method (e.g. heap scan/index scan…)
- Choose/store the min cost and its plan
- Selects, projects (on-the-fly) so can be ignored
- Results pipelined into grouping/aggregation (hashing or sorting)

Induction case ($n = k + 1$ relations)

Queries with ⋈ on 2 or more relations:

- estimate cost for every
  - Order of left-deep plan
  - Join algorithm used in each join

The Search Strategy: Dynamic Programming

Access methods costs for relation $\boldsymbol{R}$ with index $\boldsymbol{I}$

- Heap file seq scan:

$$\#\mathrm{pages}(\mathrm{R})$$

- Primary key B+ tree index matching equality selection:

$$(\mathrm{height}(I)\ +\ 1)\ +\ 1$$

- Clustered index I matching selection with selectivity **sel** :

$$(\#\mathrm{pages}\ (I) + \ \#\mathrm{pages}\ (R)\ )\ *\ \mathrm{sel}$$

- Non-clustered index I matching selection:

$$(\#\mathrm{pages}(I) + \ \#\mathrm{tuples}(R))\ *\ \mathrm{sel}$$

Base case: cost of $n\ =\ 1$ relation plans

```sql
select name from movies
where rating = 9;
```

$$\text{sel} = \frac{1}{\text{keys(I)}} = \frac{1}{10};$$
$$\#\text{pages(R)} = 500;$$
$$\#\text{pages(I)} = 50;$$
$$\#\text{tuples(R)} = 50000$$

Access methods costs for movies ***R*** with index ***I*** *on rating*

- Heap file seq scan:

$$\#\text{pages(R)} = 500$$

- Clustered index I matching selection with selectivity **sel** :

$$\left(\#\text{pages}\ (I) + \ \#\text{pages}\ (R)\right) * \ \text{sel} = (50 + 500) * \frac{1}{10} = 55$$

- Non-clustered index I matching selection:

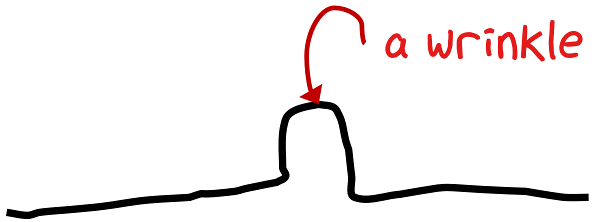
$$\left(\#\text{pages}(I) + \ \#\text{tuples}(R)\right) * \ \text{sel} = (50 + 50000) * \frac{1}{10} = 5005$$

An Example

Induction case: cost of $n = k + 1$ relation plans

Enumerate "relevant" left-deep plans over $n = k + 1$ relations in $k + 1$ passes

- Pass 1 (*Base Case*): Find best plan for each single relation
- Pass $k + 1$ (*Inductive Step*): Find best way to join result of a k-relation plan (*as outer*) to the $k + 1^{th}$ relation.



Assumption: Optimal result has optimal substructure

The best left-deep plan is composed of best decisions on the subplans

The best for joining R, S,T is one of these 3:

- (The best plan for joining R,S) ⋈ T
- (The best plan for joining T, S) ⋈ R
- (The best plan for joining R,T) ⋈ S

For each subset of relations, keep:

- cheapest plan overall

Enumerate "relevant" left-deep plans over $n = k + 1$ relations in $k + 1$ passes

- Pass 1 (*Base Case*): Find best plan for each single relation
- Pass $k + 1$ (*Inductive Step*): Find best way to join result of a k-relation plan (*as outer*) to the $k + 1^{th}$ relation.

What makes an order interesting?

An intermediate result has an "interesting order" if it is ***sorted*** by anything we can use later in the query

- ORDER BY attributes
- GROUP BY attributes
- Join attributes of potential subsequent merge joins

For each subset of relations, keep:

- cheapest plan overall
- cheapest plan overall for each ***"interesting order"***

Induction case: cost of $n = k + 1$ relation plans

Enumerating plans – the System R/Selinger way

Divide query into parts

Part 1: Dynamic Programming for ***select-project-join*** (SPJ).

- Avoid cross-products – consider a $k + 1$ join/product with a k-relation plan if:
  - There is a join condition
  - There are no more where clause predicates

Part 2: Order By, Group BY, Aggregation

- Might get an "interestingly ordered" plan
- Or add additional sort/hash operator

Query plan search even with pruning is still $O(e^n)$

- There is a lot more to learn: not the whole truth, but it's a good foundation:
  - The Postgres DP-optimizer also considers bushy plans!
  - Many other techniques: genetic optimizer, RL-optimizers, etc.
  - Still an active field: It's a hard problem!
- Better queries or better DBMS tuning
  - Why did the optimizer choose this terrible plan?
  - How can I help it to select a better one?
- A good perspective for many CS problems
  - Many problems benefit from a declarative/constrained specification and an optimizer to determine the best implementation

Why learn more about optimizers?