# 6.5830 Lecture 10

Jacob Lawrence, The Builders, 1947

## Query Optimization

## October 12, 2022

# Join Algo Summary

| Algo | I/O cost | CPU cost | In Mem? |
|---|---|---|---|
| Nested loops | \|R\|+\|S\| | O({R}x{S}) | R in mem |
| Nested loops | {S}\|R\| + \|S\| | O({R}x{S}) | No |
| Index nested loops (R index) | \|S\| + {S}c (c = 1 or 2) | O({S}log{R}) | No |
| Block nested loops | \|S\| + B\|R\| (B=\|S\|/M) | O({R}x{S}) | No |
| Sort-merge | \|R\|+\|S\| | O({S}log{S}) | Both |
| Hash (Hash R) | \|R\|+\|S\| | O({S} + {R}) | R in mem |
| Blocked hash (Hash S) | \|S\| + B\|R\| (B=\|S\|/M) | O({S} + B{R}) (*) | No |
| External Sort-merge | 3(\|R\| + \|S\|) | O(P x {S}/P log {S}/P) | No |
| Simple hash | P(\|R\|+\|S\|) (P=\|S\|/M) | O({R} + {S}) | No |
| Grace hash | 3(\|R\| + \|S\|) | O({R} + {S}) | No |

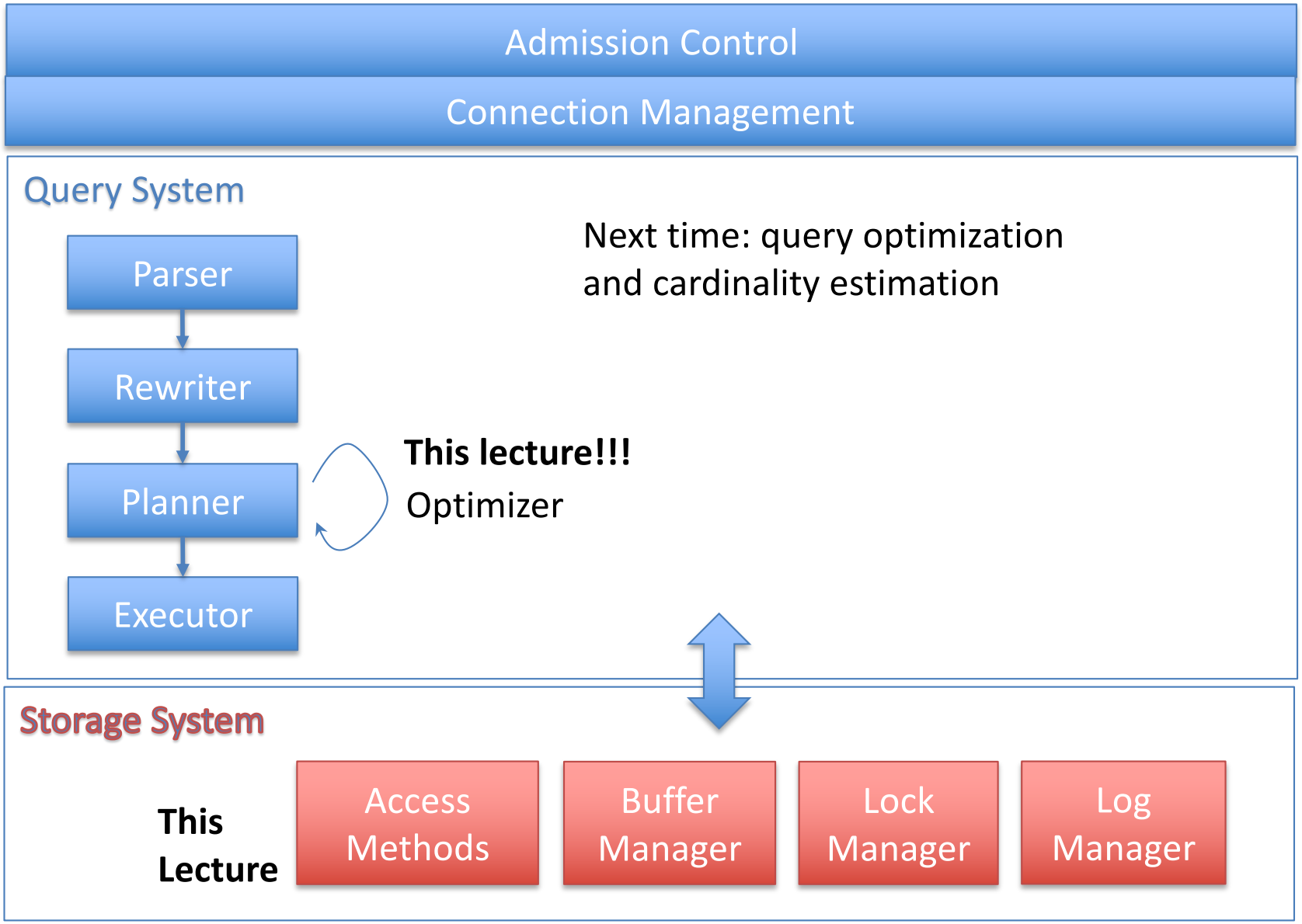
Grace hash is generally a safe bet, unless memory is close to size of tables, in which case simple can be preferable

Extra cost of sorting makes sort merge unattractive unless there is a way to access tables in sorted order (e.g., a clustered index), or a need to output data in sorted order (e.g., for a subsequent ORDER BY)

# Postgres Demo

- Try running joins with hash vs merge join

# Database Internals Outline



# Query Optimization Objective

- Find the query plan of minimum cost
  - Many possible cost functions, as we've discussed
- Requires a way to:
  - Evaluate cost of a plan
  - Enumerate (iterate through) plan options

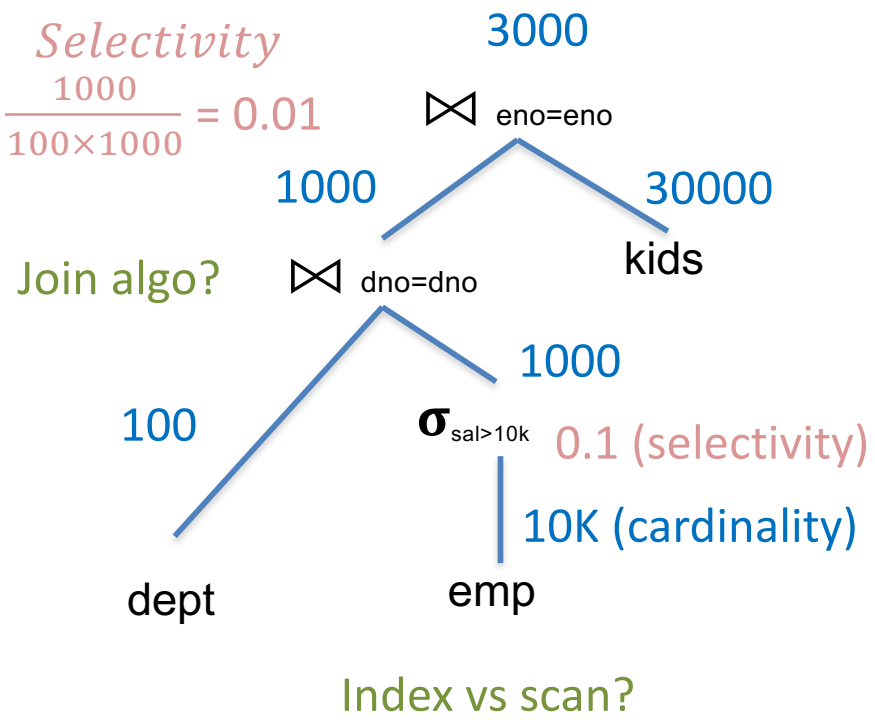
# Cost Estimation

- Cost Plan = ∑(Cost Plan Operators)
- Cost Plan Operator ∝ Size of Operator Input

- Determining Size of Operator Input
  - For base tables, equal to size on disk
    - Tables with indexes may support predicate push down
  - For other operators, equal to “selectivity” x size of children
    - Selectivity is fraction of input size that the operator emits
    - Join selectivity defined relative to the size of the cross product

# Example (Lec 5)

SELECT * FROM emp, dept, kids
WHERE sal > 10k
AND emp.dno = dept.dno
AND emp.eid = kids.eid




100 tuples/page
10 pages RAM
10 KB/page

|deptl = 100 records = 1 page = 10 KB
|empl = 10K = 100 pages = 1 MB
|kidsl = 30K = 300 pages = 3 MB

Steps:
For each plan alternative:

1. Estimate sizes of relations
2. Estimate selectivities
3. Compute intermediate sizes
4. Evaluate cost of plan operations

5. Select best plan

Steps:

1. Estimate sizes of relations
2. Estimate selectivities
3. Compute intermediate sizes
4. Evaluate cost of plan operations
5. Find best overall plan

# Selinger Statistics

**NCARD(R)** - "relation cardinality" - number of records in R

**TCARD(R)** - # pages R occupies

**ICARD(I)** - # keys (distinct values) in index I

**NINDX(I)** - pages occupied by index I

Min and max keys in indexes

Modern databases use much more sophisticated stats – will look at Postgres

Steps:

1. Estimate sizes of relations
2. Estimate selectivities
3. Compute intermediate sizes
4. Evaluate cost of plan operations
5. Find best overall plan

# Selinger Selectivities

F(pred) = Selectivity of predicate = Fraction of records that a predicate does not filter

**NCARD(R)** - "relation cardinality" - number of records in R

**TCARD(R)** - # pages R occupies

**ICARD(I)** - # keys (distinct values) in index I

**NINDX(I)** - pages occupied by index I

Min and max keys in indexes

**Predicate types**

1. col = val

F = 1/ICARD() *(if index available)*

F = 1/10 otherwise

Modern DBs use fancier stats!

2. col > val

(max key - value) / (max key - min key) *(if index available)*

1/3 otherwise

3. col1 = col2

1/MAX(ICARD(col1), ICARD(col2)) *(if index available)*

1/10 otherwise

Steps:

1. Estimate sizes of relations
2. Estimate selectivities
3. Compute intermediate sizes
4. Evaluate cost of plan operations
5. Find best overall plan

# Complex Predicates

- P1 and P2

  F(P1) x F(P2)

- P1 or P2

  1 – P(neither predicate is satisfied) =

  1 – (1-F(P1)) x (1-F(P2))

  Note uniformity assumption

Steps:

1. Estimate sizes of relations
2. Estimate selectivities
3. Compute intermediate sizes
4. Evaluate cost of plan operations
5. Find best overall plan

# Intermediate Sizes

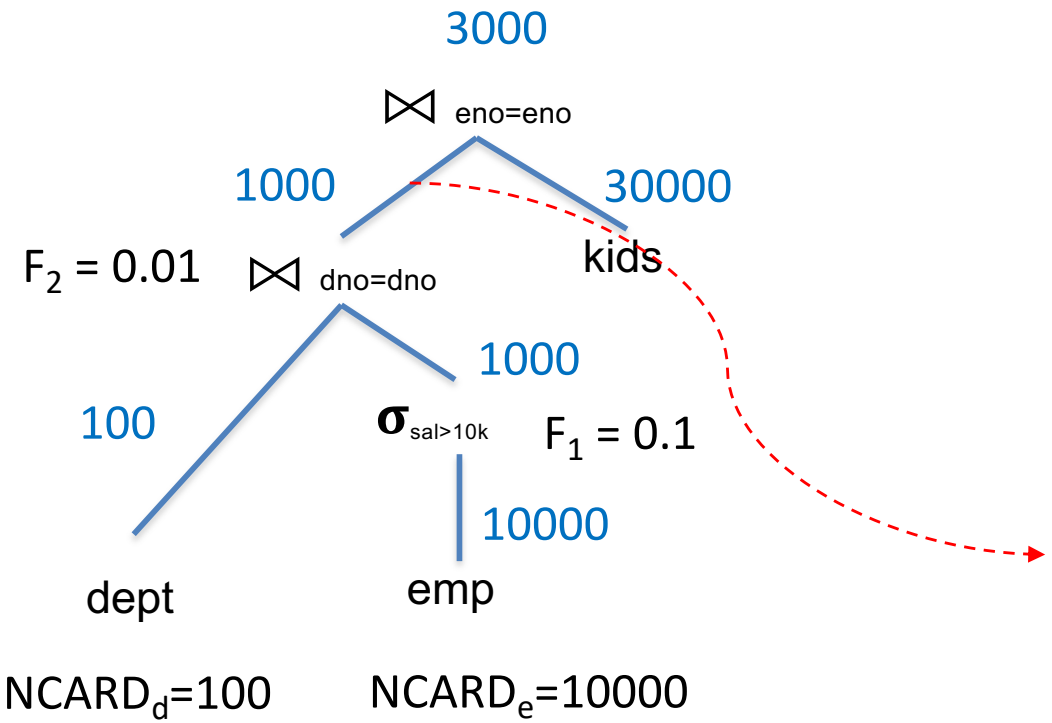
**NCARD(R)** - "relation cardinality" - number of records in R

**TCARD(R)** - # pages R occupies

**ICARD(I)** - # keys (distinct values) in index I

**NINDX(I)** - pages occupied by index I

Min and max keys in indexes




$$NCARD_d \times NCARD_e \times F_1 \times F_2 = 100 \times 10000 \times 0.1 \times 0.01 = 1000$$

Steps:

1. Estimate sizes of relations
2. Estimate selectivities
3. Compute intermediate sizes
4. Evaluate cost of plan operations
5. Find best overall plan

# Cost of Base Table Operations

Cost = pages read + weight x (records evaluated)

**NCARD(R)** - "relation cardinality" - number of records in R

**TCARD(R)** - # pages R occupies

**ICARD(I)** - # keys (distinct values) in index I

**NINDX(I)** - pages occupied by index I

Min and max keys in indexes

W: weight of CPU operations

**Equality predicate with unique index**: 1 + 1 + W

Heap File lookup

B+Tree lookup

Predicate evaluation

**Clustered index, range w/ selectivity F**: F x (NINDX + TCARD) + W x (tuples read)

One I/O per page

**Unclustered index, range w/ selectivity F** : F x (NINDX + NCARD) + W x (tuples read)

One I/O per record

**Seq (segment) scan**: TCARD + W x (NCARD)

Steps:

1. Estimate sizes of relations
2. Estimate selectivities
3. Compute intermediate sizes
4. Evaluate cost of plan operations
5. Find best overall plan

# Cost of Joins

**NCARD(R)** - "relation cardinality" - number of records in R

**TCARD(R)** - # pages R occupies

**ICARD(I)** - # keys (distinct values) in index I

W: weight of CPU operations

## NL(A,B,pred)

$$\text{Cost}(A) + \text{NCARD}(A) \times \text{Cost}(B)$$

Outer Plan  Inner Plan

Selinger only considers "left deep" plans, i.e., B is always a base table $T_{left}$

In an index on $T_{left}$, $\text{Cost}(B) = 1 + 1 + W$

If no index, $\text{Cost}(B) = \text{TCARD}(T_{left}) + W \times \text{NCARD}(T_{left})$

Cost(A) is just cost of outer subtree

Steps:

1. Estimate sizes of relations
2. Estimate selectivities
3. Compute intermediate sizes
4. Evaluate cost of plan operations
5. Find best overall plan

# Cost of Joins

**Merge(A,B,pred)**

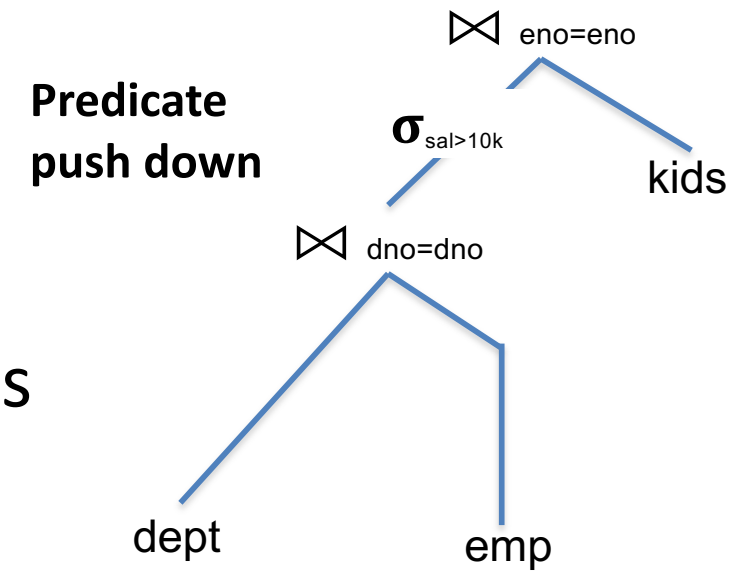
Cost(A) + Cost(B) + sort cost

If either table is a base table, cost is just the sequential scan cost

Steps:
1. Estimate sizes of relations
2. Estimate selectivities
3. Compute intermediate sizes
4. Evaluate cost of plan operations
5. Find best overall plan

# Enumerating Plans

- Selinger combines several heuristics with a search over join orders
- Heuristics
  - Push down selections
  - Don't consider cross products
  - Only "left deep" plans
    - Right side of all joins is base relation
- Still have to order joins!

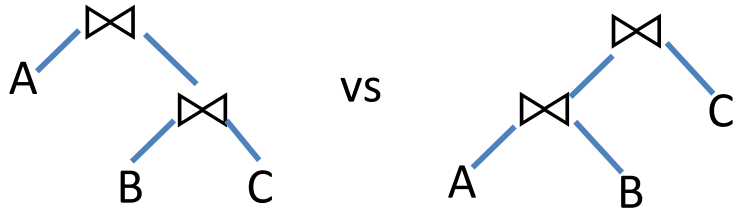


# Join ordering

- Suppose I have 3 tables, A ⋈ B ⋈ C
  - Predicates between all 3 (no cross products)
- How many orderings?

| | | |
|---|---|---|
| ABC | A(BC) | (AB)C |
| ACB | A(CB) | (AC)B |
| BAC | B(AC) | (BA)C |
| BCA | B(CA) | (BC)A |
| CAB | C(AB) | (CA)B |
| CBA | C(BA) | (CB)A |

n!




This plan is not left deep!

Left deep plans are all of the form (…(((AB)C)D)E)…)

n! left deep plans

10! = 3.6 M

15! = 1.3 T

Can we do better?

# Dynamic Programming Algorithm

- **Idea**: compute the best way to join each sub-plan, from smallest to largest
  - Don't need to reconsider subplans in larger plans
- For example, if the best way to join ABC is (AC)B, that will always be the best way to join ABC, whenever* these three relations occur as a part of a subplan.

* *Except when considering interesting orders*

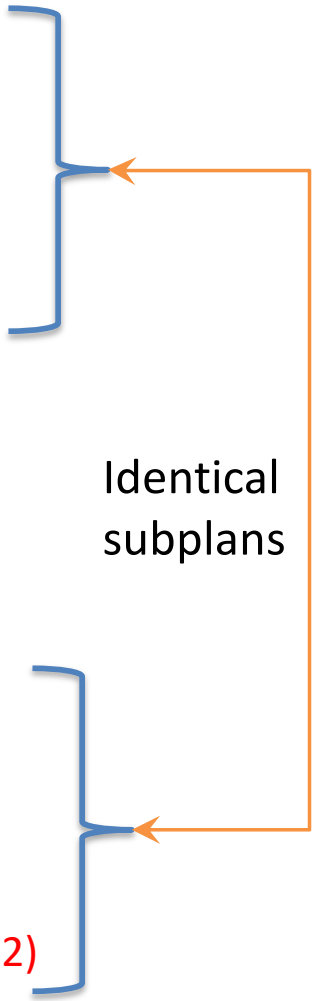
# Postgres example

*explain select * from emp join kids using (eno);*

```
Hash Join  (cost=347292.59..498019.60 rows=3000001 width=36)
  Hash Cond: (kids.eno = emp.eno)
  ->  Seq Scan on kids  (cost=0.00..49099.01 rows=3000001 width=18)
  ->  Hash  (cost=163696.15..163696.15 rows=10000115 width=22)
        ->  Seq Scan on emp  (cost=0.00..163696.15 rows=10000115 width=22)
```

*explain select * from dept join emp using(dno) join kids using (eno);*

```
Hash Join  (cost=350376.61..556245.96 rows=3000001 width=40)
  Hash Cond: (emp.dno = dept.dno)
  ->  Hash Join  (cost=347292.59..498019.60 rows=3000001 width=36)
        Hash Cond: (kids.eno = emp.eno)
        ->  Seq Scan on kids  (cost=0.00..49099.01 rows=3000001 width=18)
        ->  Hash  (cost=163696.15..163696.15 rows=10000115 width=22)
              ->  Seq Scan on emp  (cost=0.00..163696.15 rows=10000115 width=22)
  ->  Hash  (cost=1443.01..1443.01 rows=100001 width=8)
        ->  Seq Scan on dept  (cost=0.00..1443.01 rows=100001 width=8)
```



# Selinger Algorithm

R ←set of relations to join

For i in {1...|R|}:

  for S in {all length i subsets of R}:

    $optcost_s = \infty$

    $optjoin_S = \emptyset$

    for a in S:  //a is a relation

      $c_{sa}$ = $optcost_{s-a}$ +

        min. cost to join (S-a) to a +

        min. access cost for a

      if $c_{sa}$ < $optcost_s$ :

        $optcost_s = c_{sa}$

        $optjoin_s$ = optjoin(S-a) joined optimally w/ a

***Cached in previous step!***

# Example

4 Relations: ABCD

Optjoin:

A = best way to access A

(e.g., sequential scan,

or predicate pushdown into index...)

B = " " " " B

C = " " " " C

D = " " " " D

{A,B} = AB or BA

{A,C} = AC or CA

{B,C} = BC or CB

{A,D}

{B,D}

{C,D}

| Relations | Best Plan | Cost |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

**Dynamic Programming Table**

# Example (con't)

| Relations | Best Plan | Cost |
|---|---|---|
| A | Index Scan | 5 |
| B | Seq Scan | 15 |
| … | | |
| {A,B} | BA | 75 |
| {A,C} | AC | 12 |
| {B,C} | CB | 22 |
| .. | | |
| | | |
| | | |
| | | |
| | | |

Optjoin

Already computed!

{A,B,C} = remove A: compare A({B,C}) to ({B,C})A
remove B: compare ({A,C})B to B({A,C})
remove C: compare C({A,B}) to ({A,B})C

{A,C,D} = …

{A,B,D} = …

{B,C,D} = …

…

{A,B,C,D} = remove A: compare A({B,C,D}) to ({B,C,D})A
remove B: compare B({A,C,D}) to ({A,C,D})B
remove C: compare C({A,B,D}) to ({A,B,D})C
remove D: compare D({A,C,C}) to ({A,B,C})D

# Complexity

- Have to enumerate all sets of size 1…n

$$\binom{n}{1} + \binom{n}{2} \ldots + \binom{n}{n}$$

- Number of subsets of set of size n =

  |power set of n| =

  $2^n$ (here, n is number of relations)

Equivalent to all binary strings of length N, where a 1 in the ith position indicates that relation i is included:

001, 010, 100, … , 011, 111

# Complexity (cont.)

$2^n$ Subsets

How much work per subset?

Have to iterate through each element of each subset, so this at most n

$n2^n$ complexity (vs n!)

n=12 ➔ 49K vs 479M

# Interesting Orders

- Some query plans produce data in sorted order – E.g scan over a primary index, merge-join
  – Called *interesting order*
- Next operator may use this order – E.g. can be another merge-join
- For each subset of relations, compute multiple optimal plans, one for each interesting order
- Increases complexity by factor k+1, where k=number of interesting orders

# Summary

- Selinger Optimizer is the foundation of modern cost-based optimizers
  - Simple statistics
  - Several heuristics, e.g., left-deep
  - Dynamic programming algo for join ordering
- Easy to extend, e.g., with:
  - More sophisticated statistics
  - Fewer heuristics