Class 19: Joins II

Last time

Relation S, N = 500 pages $P_S = 80$
R, M = 1000 pages $P_R = 100$

Nested-Loop Joins

Simple $(P_R \cdot M) \cdot N + M$ w/ R outer

Page-oriented $M \cdot N + M$

Block-based $\frac{M \cdot N}{K} + M$ w/ K buffer

Index $M + M \cdot P_R \cdot ($index_access_cost + data_access_cost$)$

hash ~1.2 B⁺-tree 2-4

clustered: 1 I/O per page of matching tuples

unclustered: 1 I/O per matching tuple

Sort-Merge Joins

$3 \cdot (M+N)$ if $B > \sqrt{M}$ where M is # pages of the larger relation

$M+N$ if $B > N$ where N corresponds to the smallest relation

Today

→ Hash Joins
→ General Join Conditions
→ Aggregates

Hash Joins

→ Use a hash function $\underline{h}$ to create partitions of both relations $\boxed{\text{hashing (building)}}$

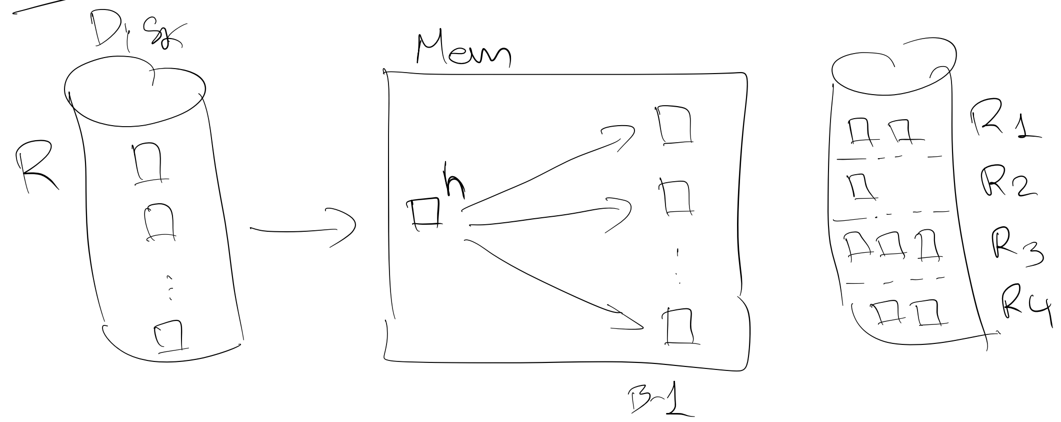
→ match tuples only between the corresponding partitions $\boxed{\text{Probing (matching)}}$

B buffers $\quad R \underset{i=j}{\bowtie} S$

h hash function

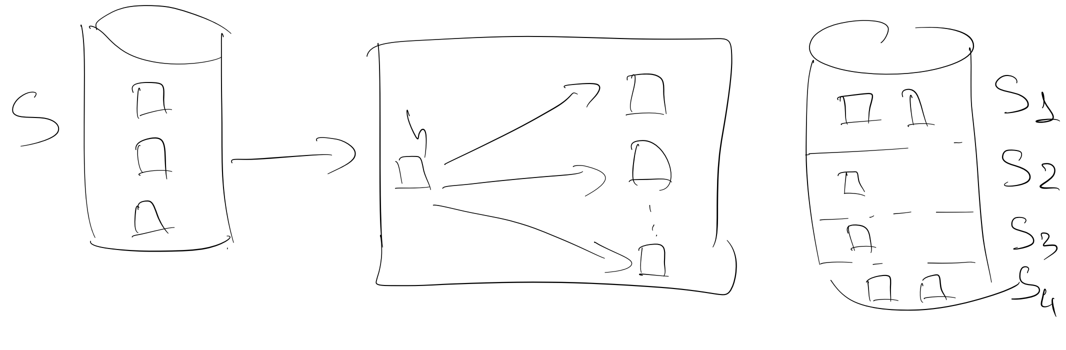
```
building {
    ∀ r ∈ R
        read r and add it to buffer $h(r_i)$
    ∀ s ∈ R
        read S and add it to buffer $h(s_j)$
}

matching {
    for $l = 1, 2, \dots K$
        ∀ r ∈ $R_l$
            read r and insert into in-memory HT using $h_2(r_i)$
        ∀ s ∈ $S_l$
            read S and probe HT using $h_2(s_j)$
            if match found add ⟨r,S⟩ to the result
        clear hash table from memory to proceed
        with next pair of partitions
}
```

Building



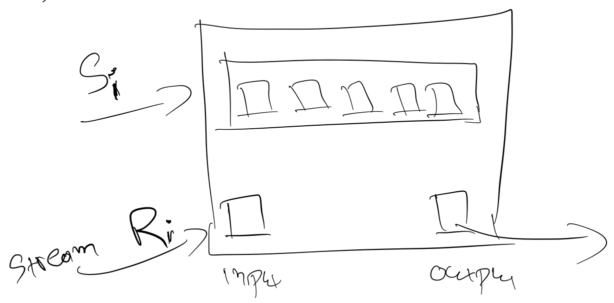
cost

$2 \cdot M$



$2 \cdot N$

Matching



read every partition once

in-memory HT w/ h2 ($\neq h$)

Search in S_i as we stream R_i

Cost: $M + N$

total cost of Hash Join $= 3(M+N) = \boxed{4500} \to \boxed{45}$

Memory Requirements

→ enough buffer for the largest partition of the smaller relation (S)

→ Input page for the other relation

→ Output page

→ a few pages of hash metadata

Fudge factor f (for example $f = 1.04$)

if h → uniform

size of a partition $\sim \frac{N}{B-1}$

$$B > \frac{f \cdot N}{B-1} + 2 \approx > B > \sqrt{f \cdot N}$$

What if not enough memory? (for Si to fit in memory)

→ apply the same algorith recursively

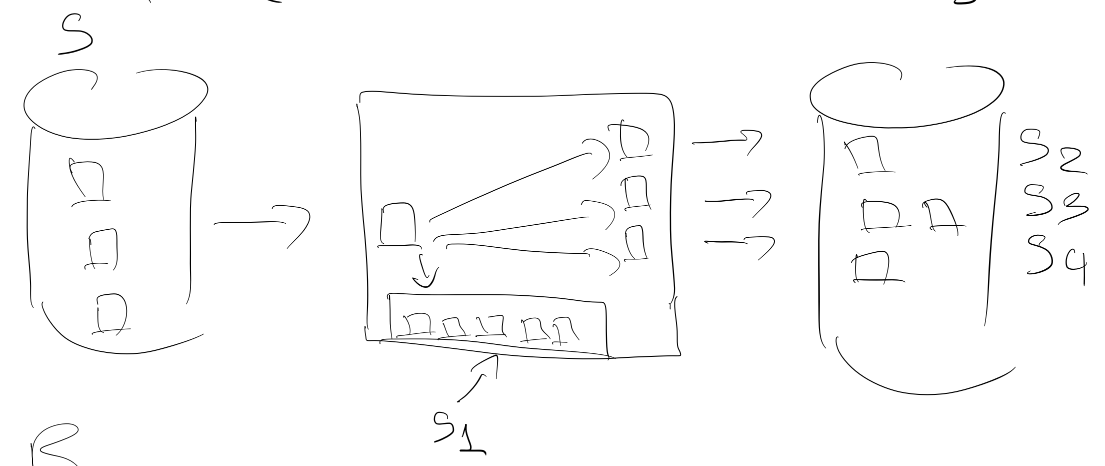
→ read, repartition S_i, R_i with h_3 ($\neq h_2, \neq h$)

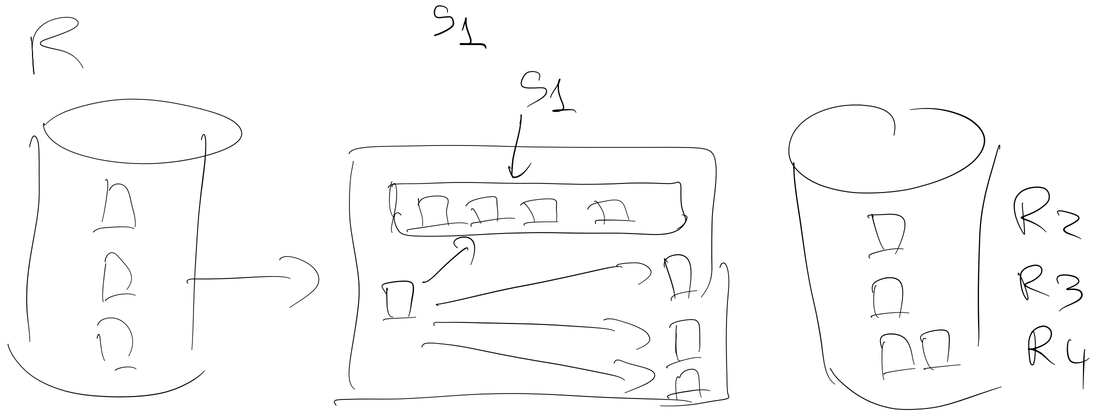
→ matching per subpartition (mem. is enough)

* if not, again recursion

What if we have more memory?

<u>Hybrid</u>
<u>Hash</u>
<u>Join</u>





<u>Cost</u>

→	hashing S	$N + N - \text{sizeof}(S_1)$
→	hashing R	$M + M - \text{sizeof}(R_1)$
→	matching	$M - \text{sizeof}(R_1) + N + \text{sizeof}(S_1)$
	total	$3(M+N) - 2(\text{sizeof}(S_1) + \text{sizeof}(R_1))$

$B = 300$
$M = 1000$
$N = 500$

$$3(1000+500) - 2(500+250) = 4500 - 1500 = \boxed{3000}$$

$\boxed{6s}$

if $B = 600$

read S <u>once</u> + build hash table

Scan R <u>once</u> prob S on-the-fly

	Hash Join	vs	SMJ
cost	$3(M+N)$		$3(M+N)$
mem. req.	$B > \sqrt{f \cdot N}$ ← smaller		$B > \sqrt{M}$ ← larger
	$B > \sqrt{1.04 \cdot 500} = 23$		$B > \sqrt{1000} = 32$
$B (> m.r.) < N$	$3(M+N) - 2(sizeof(R_1) + sizeof(S_1))$		$3(M+N)$
$B > N$	$M+N$		$M+N$
output	___		sorted
if input sorted	$3(M+N)$		$M+N$

BUT sensitive to data skew

(a) equality joins on several attributes

(b) inequality joins

→ (a) for INLJ we need index with all attributes in join conditions

→ sort / hash use combination of all attributes

→ (b) INLJ w/ B^+-Tree (not Hash Index)

HJ / SMJ cannot work

Block NLJ the best approach

Set

UNION / EXCEPT (set difference)

→ Sorting

→ sort S + R on all attributes

→ merging → discard duplicates (UNION)

→ set_difference

→ hashing

→ partition R+S

→ ∀ S-part probe corr. R-part

→ discard duplicates (UNION)

→ set-difference

→ Intersection → special case of Join

Equality across all attributes

Aggregation

→ SELECT AVG(sal) FROM E

→ SCAN once

→ GROUP BY

<age, avg_salary>

hash (age) ⟶ <age, salary, count>

sort (age) Calculate "running info" of aggregation on-the-fly

→ if we have an index on <Group_by, select, where>

can use only the index $\boxed{\text{WAY FASTER}}$

→ Buffering

many thing in parallel

tough to estimate what is cached by BP

SNLJ B>N ✓.

B<N LRU → sequential flooding.

MRU ✓.

INLJ → sort the outer relation