Class 16 : 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 $M + M \cdot p_R \cdot (\text{index\_access\_cost} + \text{data\_access\_cost})$

index_access_cost: hash ~1.2, B^+-tree 2–4

data_access_cost: 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 $\boxed{\text{Probing (matching)}}$

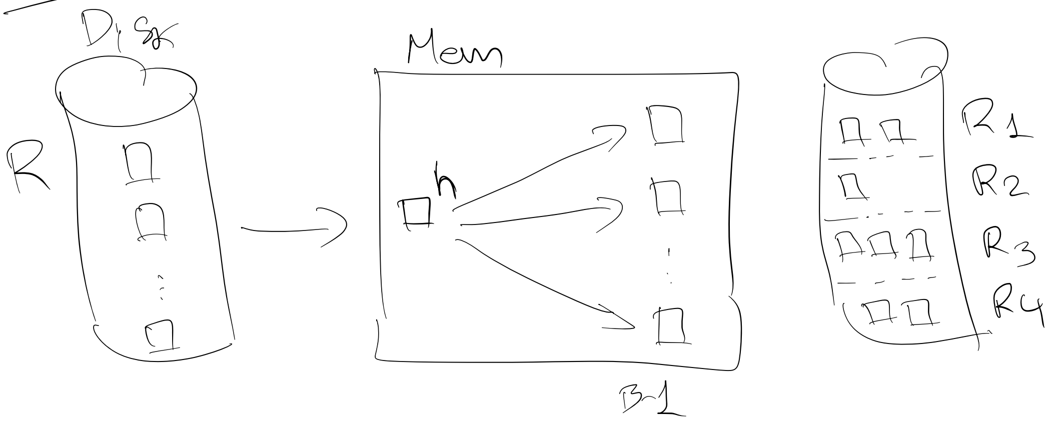
B buffers

$R \bowtie_{i=j} S$

h hash function

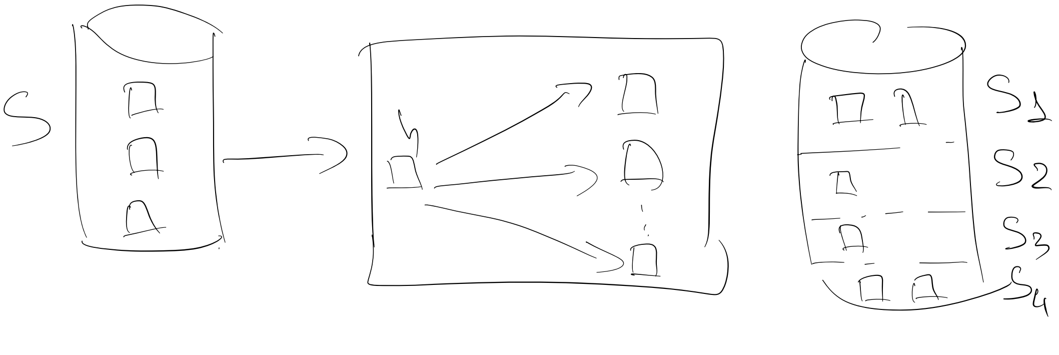
```
building
∀ r ∈ R
    read r and add it to buffer $h(r_i)$
∀ s ∈ R
    read S and ~~add~~ it to buffer $h(S_j)$

matching
for l=1,2,.... 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



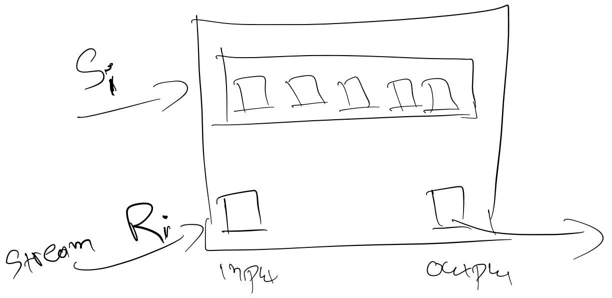
cost

$2 \cdot M$



$2 \cdot N$

Matching



read every partition once

in-memory HT w/ h2 (≠h)

Search in S_i as we stream R_i

Cost: $M + N$

total cost of Hash Join $= 3(M+N) = \boxed{4500} \rightarrow \boxed{9s}$

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 $h \rightarrow$ uniform

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

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

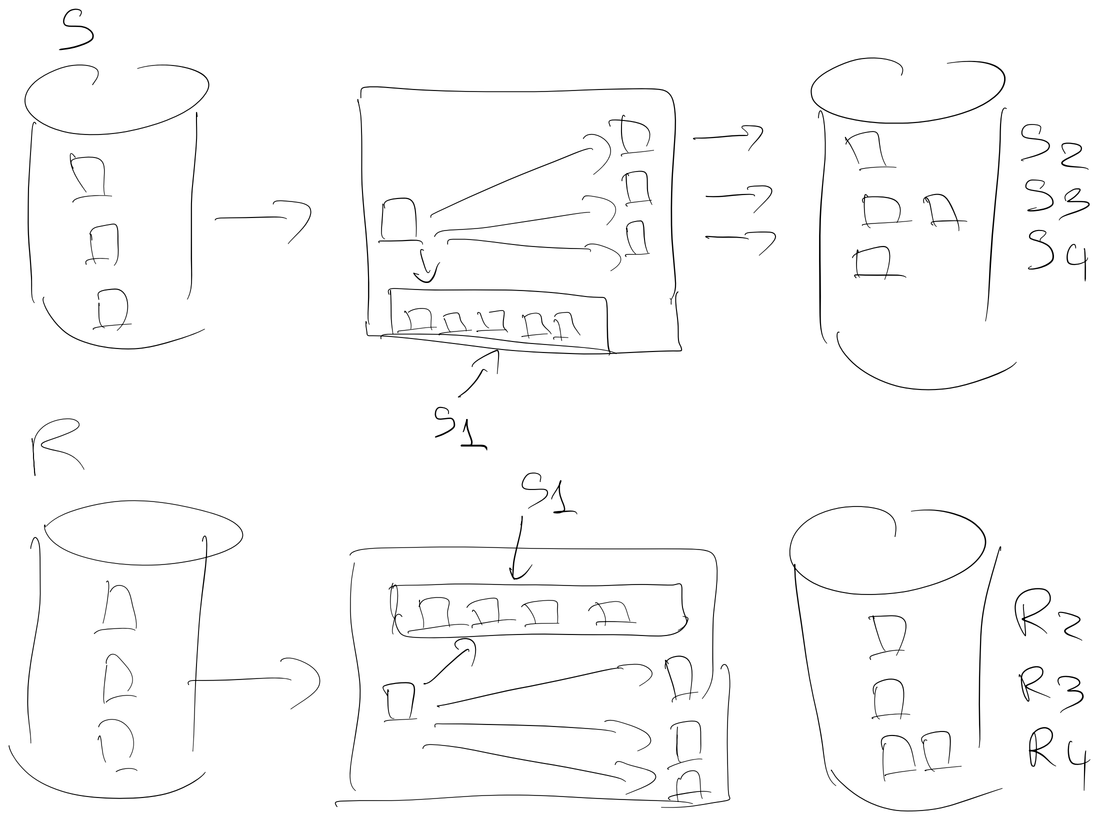
What if not enough memory? (for S_i to fit in memory)

- → apply the same algorithm 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?

Hybrid Hash Join



Cost

$\to$ hashing S $\quad N + N - \text{sizeof}(S_1)$

$\to$ hashing R $\quad M + M - \text{sizeof}(R_1)$

$\to$ matching $\quad M - \text{sizeof}(R_1) + N + \text{sizeof}(S_1)$

total $\quad 3(M+N) - 2(\text{sizeof}(S_1) + \text{sizeof}(R_1))$

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

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

$\boxed{6s}$

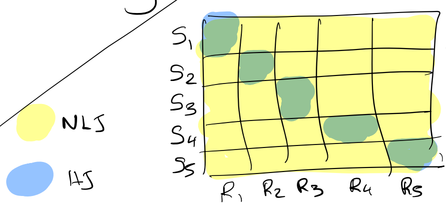
if $B = 600$

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

scan R <u>once</u> probe S on-the-fly

HJ vs NLJ

Work done during matching



NLJ compares all r∈R with all s∈S while HJ only r∈Ri wit s∈Si

	Hash Join	vs	SMJ
cost	$3(M+N)$		$3(M+N)$
memory requirement (m.r.)	$B > \sqrt{f \cdot N}$ ← smaller		$B > \sqrt{M}$ ← larger
	example: $B > \sqrt{1.04 \cdot 500} = 23$		$B > \sqrt{1000} = 32$
$\sqrt{fN} \leq B < \sqrt{M}$	HJ exploits additional mem $3(M+N) - 2(\text{sizeof}(R_1) - \text{sizeof}(S_1))$		needs additional passes (cost is $>3(M+N)$)
$\sqrt{M} \leq B < N$			$3(M+N)$
$B > N$	$M+N$		$M+N$
output	___		sorted
if input sorted	$3(M+N)$		$M+N$

BUT sensitive to data skew

(a) equality join 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

→ refinement also applies

→ 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 ✓.

BNLJ replacement policy has no impact

INLJ → sort the outer relation