$R \bowtie S$

Two pass algorithms

organize R. S, (sortig)

Pass 1 : organize data

corresponding org

Pass 2 : Use org. perform op

Two-Pass Algorithms Based on Sorting

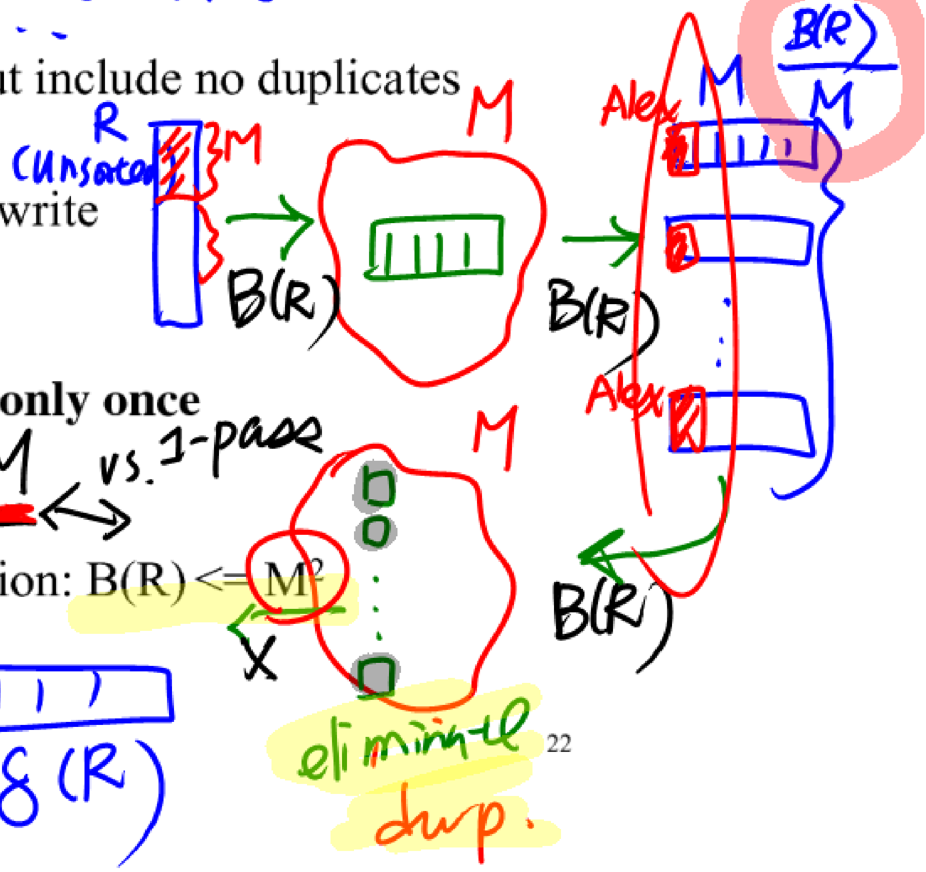
$\frac{B(R)}{M} \le M$

Duplicate elimination δ(R) Select distint

- Simple idea: like sorting, but include no duplicates
- Step 1: sort runs of size M, write
  - Cost: 2B(R)
- Step 2: merge M-1 runs,

  but **include each tuple only once**
  - Cost: B(R) $B(\delta(R)) \le M$ ⟷ vs. 1-pass
- Total cost: 3B(R), Assumption: B(R) <= M²

Just like merge sorting based.



δ(R)

Q: What can $\boxed{\text{sorting}}$ help? And, how?

- Selection?
- Projection?
- Set operations?
- Join?
- Duplicate elimination?
- Grouping?

R ∩ S , R ∪ S . ✓

distinct/or not.

① default of Det: ∃
select No
from eliminate →
where

② select Det:
[...
from
$\boxed{\text{Intersect}}$ Eliminat
select dup.
... sorting

Skip

Two-Pass Algorithms Based on Sorting

Grouping: $\gamma_{city,\ sum(price)}(R)$ Group-By. Exactly like $\delta(R)$ $\Rightarrow \equiv$ Sorting.

- Same as before: sort, then compute the sum(price) for each group
- As before: compute sum(price) during the merge phase. same!
- Total cost: 3B(R)
- Assumption: $B(R) <= M^2$

Two-Pass Algorithms Based on Sorting

Sorting of R, S

Binary operations: R ∩ S, R U S, R – S

- Idea: sort R, sort S, then do the right thing
- A closer look:
  - Step 1: split R into runs of size M, then split S into runs of size M. Cost: 2B(R) + 2B(S)
  - Step 2: merge *all* x runs from R; merge all y runs from S; ouput a ~~tuple on a case~~ by cases basis ($x + y \le M$)
- Total cost: 3B(R)+3B(S)
- Assumption: $B(R)+B(S) \le M^2$

Two-Pass Algorithms Based on Sorting

Join R⋈S

merge R

merge S

separately.

①
- Start by sorting both R and S on the join attribute:
  - Cost: 4B(R)+4B(S) (because need to write to disk)
- Read both relations in sorted order, match tuples
  - Cost: B(R)+B(S)
- Difficulty: many tuples in R may match many in S
  - If at least one set of tuples fits in M, we are OK
  - Otherwise need ~~nested loop~~, higher cost
- Total cost: 5B(R)+5B(S) higher
- Assumption: $B(R) <= M^2$, $B(S) <= M^2$ easier.

like sorting of R. S.

② Total cost: 3B(R) + 3B(S)

Assumption: $B(R) + B(S) <= M^2$

Q: Why is sorting-based "two" pass?

- Pass 1? Organize by sorting into "runs"
- Pass 2?

join the corresponding neighborhood of each run

$R \rightarrow R_1, \ldots, R_{10}$

$S \rightarrow S_1, \ldots, S_{20}$

$\rightarrow$ merge

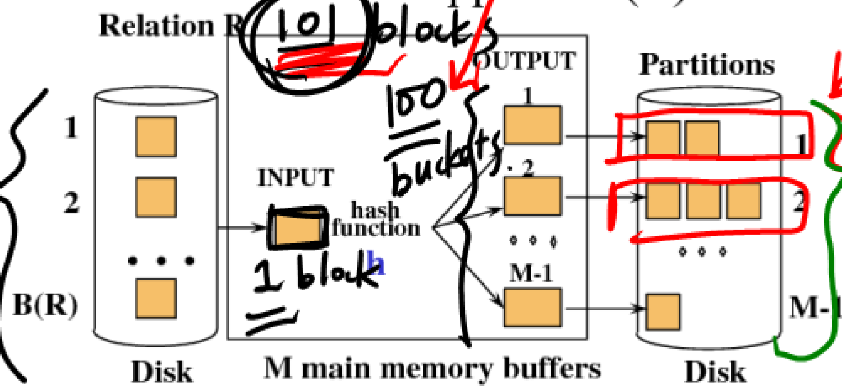
sorting

Two Pass Algorithms Based on Hashing

use as many buckets as possible

⇒ smaller buckets!

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. B(R)/M



buc ⇒ $\frac{1000 \text{ blocks}}{100 \text{ buc}} = 10$

$\frac{B(R)}{M}$ easiest

M

- Does each bucket fit in main memory ?
  - Yes if B(R)/M <= M, i.e. B(R) <= M^2

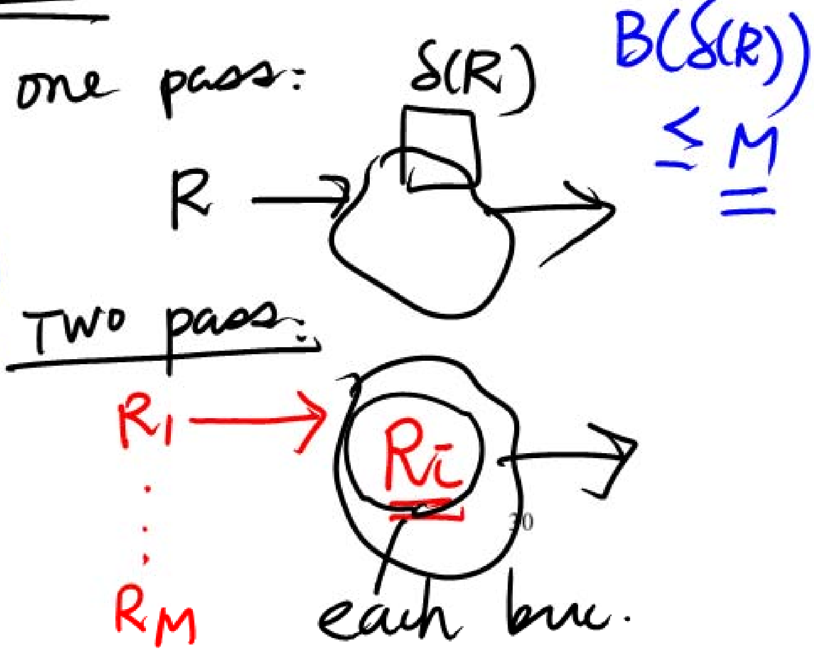
bucket $\leq M$ ⇒ $\frac{B(R)}{M} \leq M$ ⇒ $B(R) \leq M^2$

Q: What can hashing help? And, how?

- Selection?
- Projection?
- Set operations?
- Join?
- Duplicate elimination?
- Grouping?

Hash Based Algorithms for δ

- Recall: $\delta(R)$ = duplicate elimination
- Step 1. Partition R into buckets $R \to R_1 \dots R_M$
- Step 2. Apply δ to each bucket (may read in main memory)
- Cost: 3B(R)
- Assumption: $B(R) \le M^2$



Skip

Hash Based Algorithms for γ

- Recall: $\gamma(R)$ = grouping and aggregation
- Step 1. Partition R into buckets
- Step 2. Apply γ to each bucket (may read in main memory)

- Cost: $3B(R)$
- Assumption: $B(R) \le M^2$

Hash-based Join

- $R \bowtie S$
- Simple version: *main memory hash-based join*
  - Scan S, build buckets in main memory
  - Then scan R and join

- Requirement: min(B(R), B(S)) <= M

$T_{11} \bowtie T_{22}$

$T_{11}.P > T_{22}.P$

Equality

Partitioned Hash Join

Foreign key

$M = 100$

$R \bowtie S$

- Step 1:
  - Hash S into M buckets
  - send all buckets to disk
- Step 2
  - Hash R into M buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

same Hash func

1 $S \rightarrow S_1 \ldots S_{100}$

2 $R \rightarrow R_1 \ldots R_{100}$

3 $R_i \bowtie S_i$

$R_1 \bowtie S_1$

$R_2 \bowtie S_2$ ...

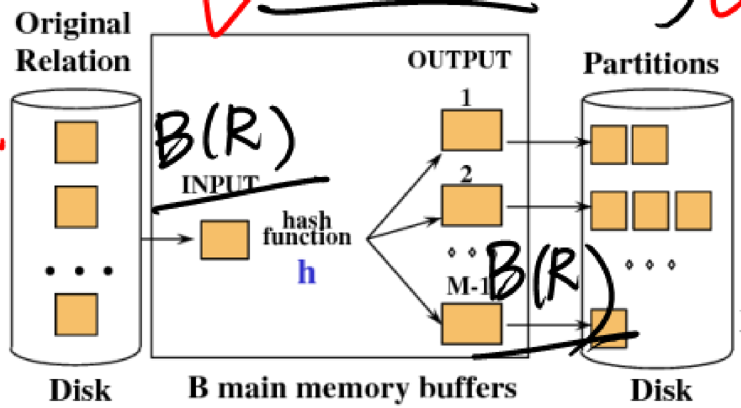
$R_{100} \bowtie S_{100}$

Hash

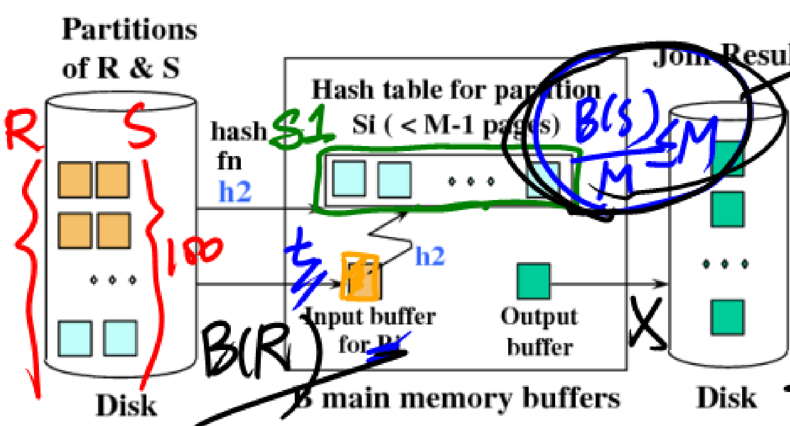
① variance "H" ② one full buc in mem.

Partitioned Hash-Join

(S) R



- Partition both relations using hash fn **h**: R tuples in partition i will only match S tuples in partition i.



- Read in a partition of R, hash it using **h2 (<> h!)**. Scan matching partition of S, search for 180 matches.

Partitioned Hash Join

- Cost: 3B(R) + 3B(S)
- Assumption:

  At least one full bucket of the smaller rel must fit in memory: min(B(R), B(S)) <= M^2

one bucket of small rel. fits in mem

Partitioned Hash Join

- Cost: 3B(R) + 3B(S)
- Assumption:

  At least one full bucket of the smaller rel must fit in memory: min(B(R), B(S)) <= M^2

one bucket of small rel. fits in mem

Two Pass

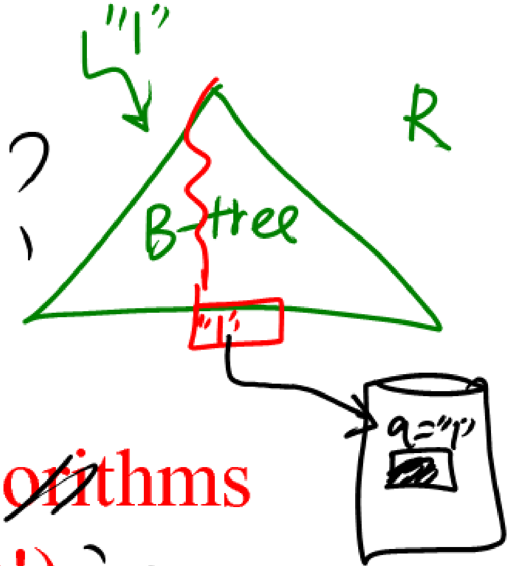
Pass 1: Hash (organize)

$R_i, S_i \;\; \forall i = 1 \ldots M$

Pass 2: $\underline{R_i} \bowtie \underline{S_i}$

Selection $\sigma_{R.a=\text{"1"}}(R)$ cost?

search — we talked in Index.



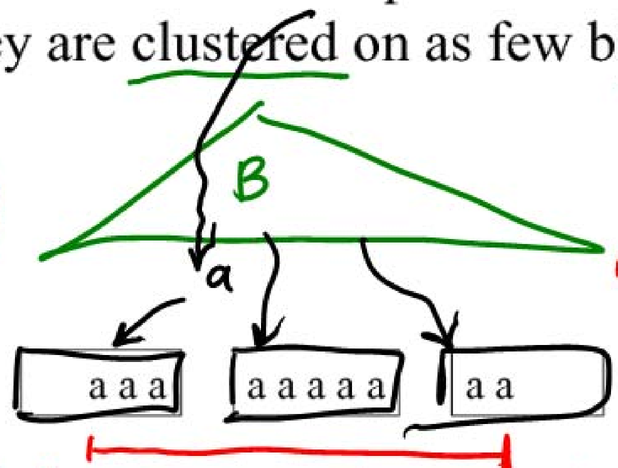
Index-based algorithms

(zero-pass!) join

Indexed Based Algorithms

- In a clustered index all tuples with the same value of the key are clustered on as few blocks as possible

$\sigma_{A=a}(R)$



in mem.! (free)

R: ordered by attr A.

blocks of A="a"

Statistic Parameter $V(R, \text{"name"}) = 5000$

Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$
- Clustered index on a: cost $B(R)/V(R,a)$ $V(R, dept) = 4$
- Unclustered index on a: cost $T(R)/V(R,a)$

R : 1000 blocks. 5000 tuples

R. dept = { CS, EE, ME, bio }

the blocks for $\sigma_{dept=\text{"cs"}}$

(clustered) $= \frac{1000}{4} = 250$

non-clustered

(tuple = block)

$\frac{5000}{4} = 1250$

$T(R) = B(R)$

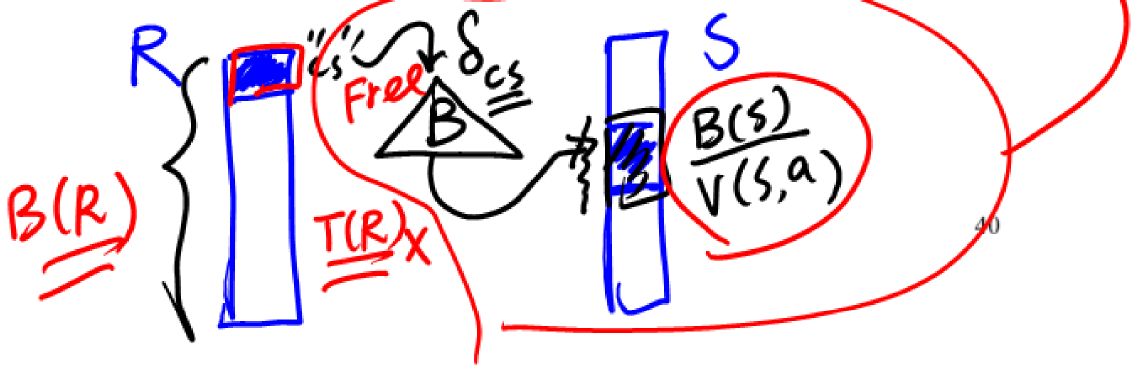
Index Based Selection

- Example: $B(R) = 2000$, $T(R) = 100{,}000$, $V(R, a) = 20$, compute the cost of $\sigma_{a=v}(R)$
- Cost of table scan:
  - If R is clustered: $B(R) = 2000$ I/Os
  - If R is unclustered: $T(R) = 100{,}000$ I/Os
- Cost of index based selection:
  - If index is clustered: $B(R)/V(R,a) = 100$
  - If index is unclustered: $T(R)/V(R,a) = 5000$
- Notice: when $V(R,a)$ is small, then unclustered index is useless

Index Based Join

Zero

- R $\bowtie$ S
- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S
- Assume R is clustered. Cost:
  - If index is clustered: B(R) + T(R)B(S)/V(S,a)
  - If index is unclustered: B(R) + T(R)T(S)/V(S,a)



Average SQLLite Score: 3.2

Average SQL Tuning Score: 3.65

Suggestions	Count
keep it!	31
more engaging lectures	12
more php/sql demos	9
stop it!	9
no suggestion	6
Combine into 1 lecture	3
hot topics in db field	3
more integration, no standalone ST lectures	3
topic comparing production RDBMS	3
topic on noSQL	3
topic on something other than SQL	3
topic on web crawling	3
vote on topics before hand	3
enum exam topics from ST lectures	2
topic on massively scalable DBs	2
easier topics	1
have kevin teach ST lectures	1
lecture from industry	1
more famous speakers	1
more hands one topics	1
more variety	1
move ST lectures up	1
no ST lecture on Fridays	1
remove SQLLite	1
topic on DB's behind facebook, twitter	1
topic on hash tables	1
topic on OODBMS	1
topic on Oracle	1
topic on speeding up sql	1
Use Previous Projects	1