RDBMS and SQL

Madhavan Mukund

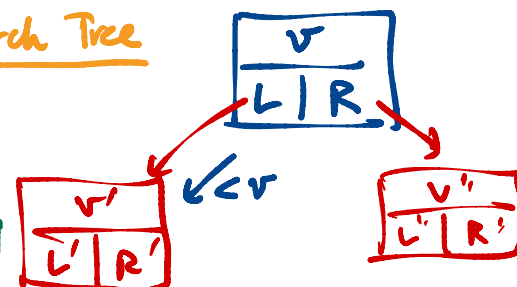
https://www.cmi.ac.in/~madhavan

Lecture 10, 6 November 2025

Recap

- Store tables on disk
- Transfer data in blocks — seek, transfer
- Maintain index

Search Tree

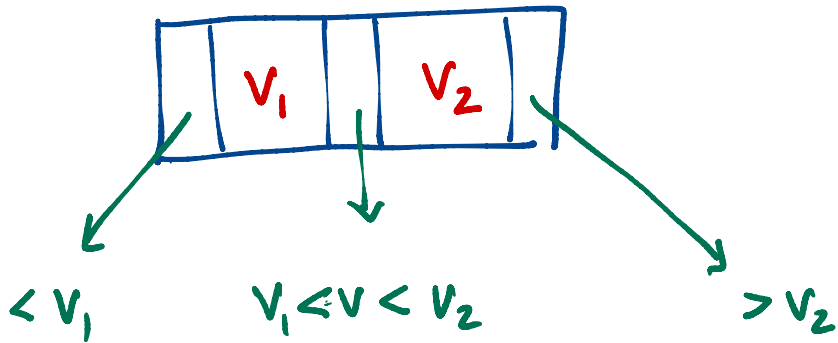


k levels → $2^0 + 2^1 + \ldots + 2^{k-1}$

$= 2^k - 1$ — size N

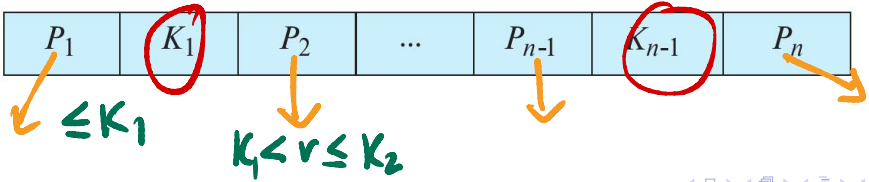
$k = \log N$

Generalize to 3-ary trees



Search trees

- Binary search trees
  - Binary search on dynamic data
  - Balanced tree has logarithmic height
- Block-based access
  - Binary tree node has one search key value, two pointers
  - Block can hold much more
- Generalize to multiple key values, multiple pointers



1 TB disk — 10^{12}

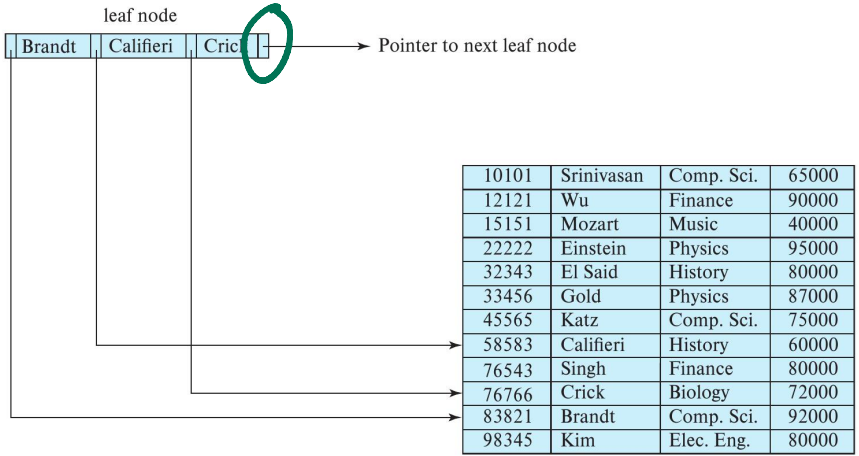
$\Rightarrow$ 4096 Byte blocks $\approx 10^9$ blocks

$\downarrow$

$\log_2 10^9 \approx 30$

B+ trees

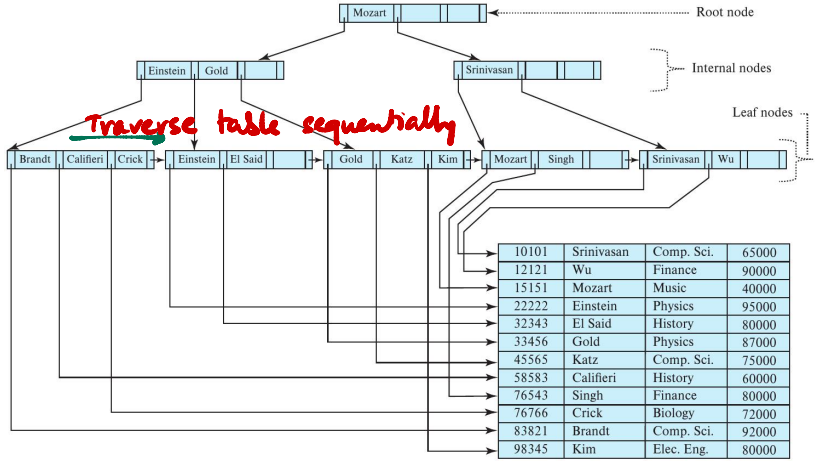
- Leaf nodes form a dense index — linked list of leaves, each one block



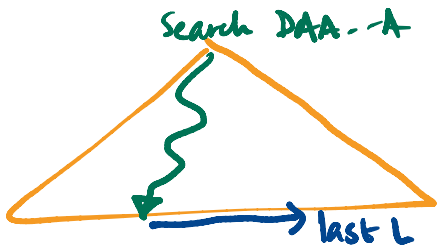
instructor file

B+ trees

- Leaf nodes form a dense index — linked list of leaves
- Non-Leaf nodes form a sparse index



- Name = "Brandt"
- Name starts with D – L Range of values



B+ trees

- Leaf nodes form a dense index — linked list of leaves
- Non-leaf nodes form a sparse index
- Constraints — assume n keys and pointers can fit in a block
  - Each leaf has at least $\lceil (n-1)/2 \rceil$ key values
  - Each non-leaf has at least $\lceil n/2 \rceil$ pointers
  - Height of the tree is proportional to $\log_{n/2}(n)$

Query processing

- Translate the query from SQL into relational algebra
- Evaluate the relational algebra expression

Query processing

- Translate the query from SQL into relational algebra
- Evaluate the relational algebra expression
- Challenges
  - Many equivalent relational algebra expressions

    $\sigma_{salary<75000}(\pi_{salary}(instructor))$ vs $\pi_{salary}(\sigma_{salary<75000}(instructor))$
  - Many ways to evaluate a given expression

Query processing

- Translate the query from SQL into relational algebra
- Evaluate the relational algebra expression
- Challenges
  - Many equivalent relational algebra expressions

    $\sigma_{salary<75000}(\pi_{salary}(instructor))$ vs $\pi_{salary}(\sigma_{salary<75000}(instructor))$
  - Many ways to evaluate a given expression
- Query plan
  - Annotate the expression with a detailed evaluation strategy key values
    - Use index on *salary* to find instructors with $salary < 75000$
    - Or, scan entire relation, discard rows with $salary \geq 75000$

Query optimization

- Choose plan with lowest cost
- Maintain database catalogue — number of tuples in each relationn, size of tuples, ...
- Assess cost in terms of disk access and transfer, CPU time, ...
- For simplicity, ignore in-memory costs (CPU time), restrict to disk access

Query optimization

- Choose plan with lowest cost
- Maintain database catalogue — number of tuples in each relationn, size of tuples, ...
- Assess cost in terms of disk access and transfer, CPU time, ...
- For simplicity, ignore in-memory costs (CPU time), restrict to disk access
- Disk accesses
  - Relation r occupies b_r blocks
  - Disk seeks — time t_S per seek
  - Block transfers — time t_T per transfer

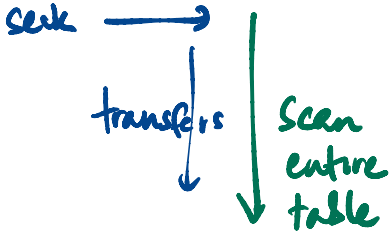
Query optimization

- Choose plan with lowest cost
- Maintain database catalogue — number of tuples in each relationn, size of tuples, ...
- Assess cost in terms of disk access and transfer, CPU time, ...
- For simplicity, ignore in-memory costs (CPU time), restrict to disk access
- Disk accesses
  - Relation r occupies b_r blocks
  - Disk seeks — time t_S per seek
  - Block transfers — time t_T per transfer
- Other factors — buffer management etc

Selection

- Linear search

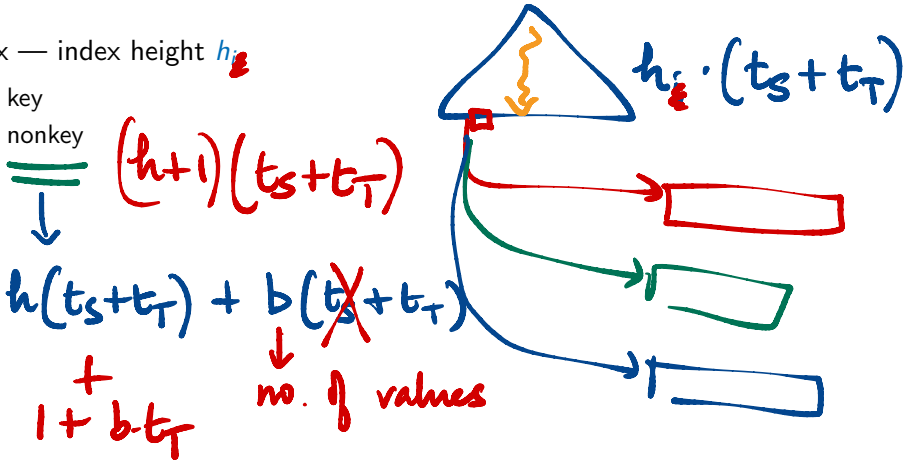
$t_S + b_r \cdot t_T$



$\sigma_\theta(r)$

Selection

- Linear search
- Clustering index — index height h_i
  - Equality on key
  - Equality on nonkey



Selection

- Linear search
- Clustering index — index height h_i — table is ordered by this
  - Equality on key
  - Equality on nonkey
- Secondary index (key, non-key)
  - $h(t_S + t_T) + b(t_S + t_T)$
  - Same as before

Selection

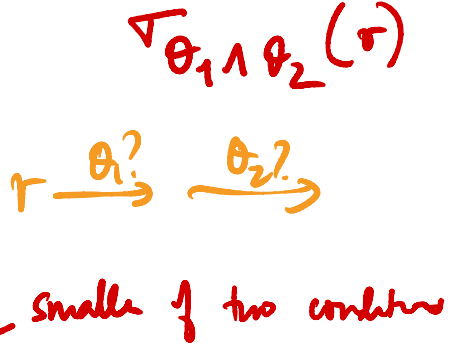
- Linear search
- Clustering index — index height h_i
  - Equality on key
  - Equality on nonkey
- Secondary index (key, non-key)
- Clustering index, comparison on A
  - Sorted on A
  - Not sorted on A

$v_1 \leq A \leq v_2$ = size n blocks

$h(t_S + t_T) + t_S + n \cdot t_T$

Selection

- Linear search
- Clustering index — index height h_i
  - Equality on key
  - Equality on nonkey
- Secondary index (key, non-key)
- Clustering index, comparison on A
  - Sorted on A
  - Not sorted on A
- Boolean combinations
  - Conjunctive selection using one index
  - Conjunctive selection using composite index
  - Disjunction, negation, ...



$$J_{Join} = \nabla_{\theta} (r_1 \times r_2)$$

Sorting

- In-memory sorting vs sorting on disk

Sorting

- In-memory sorting vs sorting on disk
- Merging sorted lists — varieties

Merging — Combine 2 sorted lists into 1

Duplicates — Eliminate — Union

— Keep only duplicates — Intersect

— Set difference

Sorting

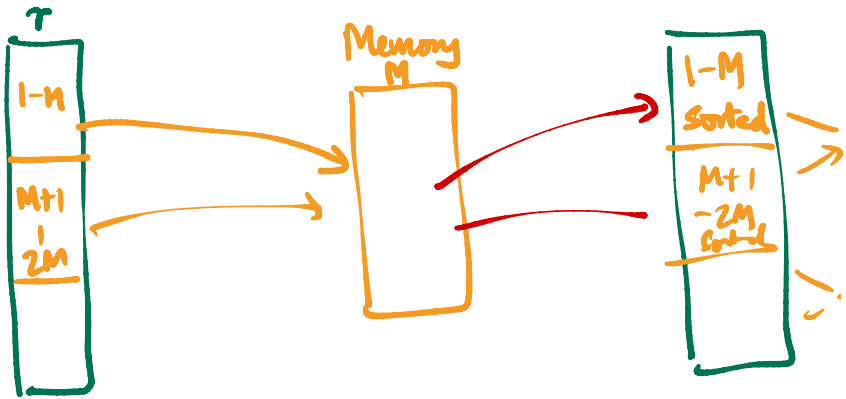
- In-memory sorting vs sorting on disk
- Merging sorted lists — varieties
- Traditional merge sort

External merge sort

- N records, b_r blocks, M blocks in memory

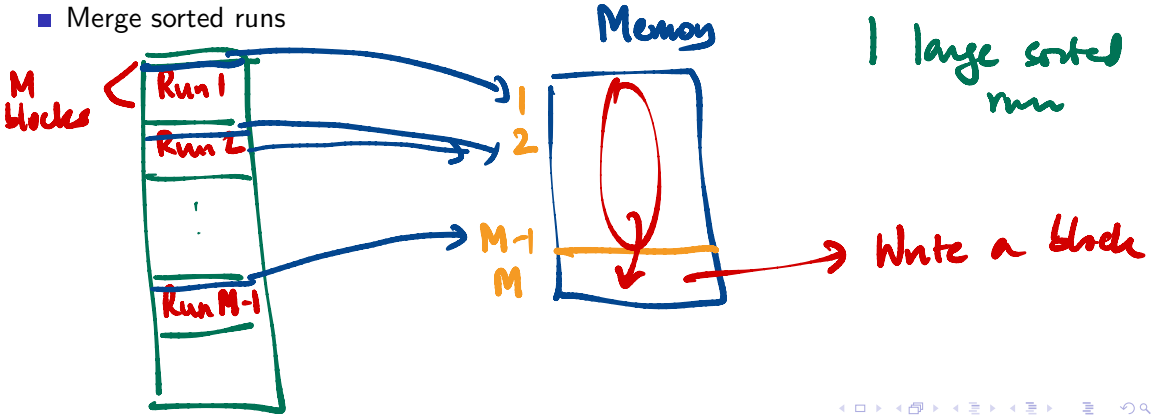
External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M



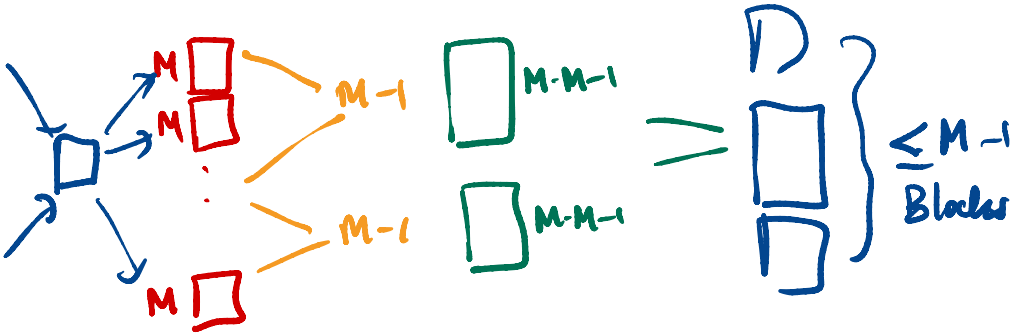
External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs



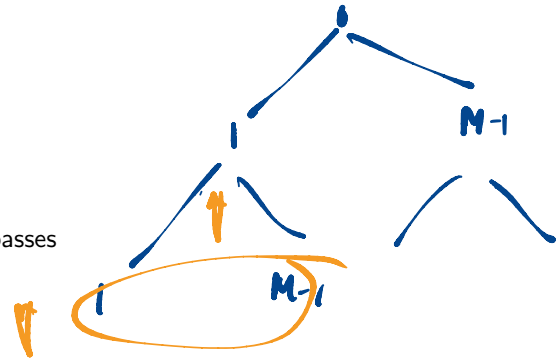
External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs



External merge sort

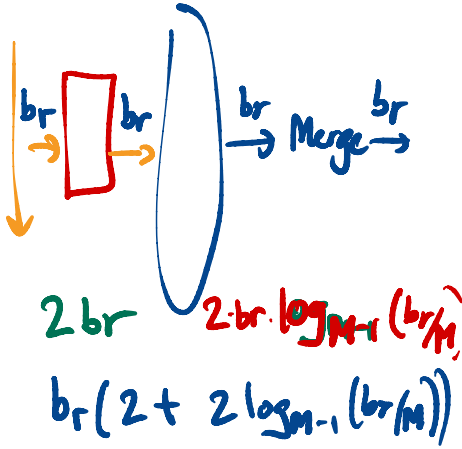
- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs
- Complexity
  - b_r/M sorted runs, $\lceil \log_{M-1}(b_r/M) \rceil$ merge passes



External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs
- Complexity
  - b_r/M sorted runs, $\lceil \log_{M-1}(b_r/M) \rceil$ merge passes
  - Block transfers — $b_r\ (2\lceil \log_{M-1}(b_r/M) \rceil + 1)$
    - Why not $b_r\ (2\lceil \log_{M-1}(b_r/M) \rceil + 2)$?

Last round of writing is ignored - "pipeline"



External merge sort

- N records, b_r blocks, M blocks in memory
- Compute sorted runs of size M
- Merge sorted runs
- Complexity
  - b_r/M sorted runs, $\lceil \log_{M-1}(b_r/M) \rceil$ merge passes
  - Block transfers — $b_r \ (2\lceil \log_{M-1}(b_r/M) \rceil + 1)$
    - Why not $b_r \ (2\lceil \log_{M-1}(b_r/M) \rceil + 2)$?
  - Block seeks — $2\lceil b_r/M \rceil + b_r \ (2(\lceil \log_{M-1}(b_r/M) \rceil - 1)$

Computing joins

- Running example
  - $Student \bowtie Takes$

Computing joins

- Running example
  - *Student* $\bowtie$ *Takes*
  - *Student* — 5000 rows, 100 blocks — 50 rows / block
  - *Takes* — 10000 rows, 400 blocks — 25 rows / block

Nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)

Nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation

for each row in r
  for each row in s

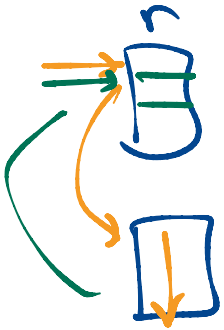
Nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + n_r \cdot b_s$

Nested-loop join

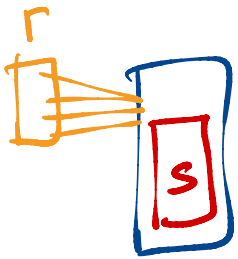
- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + n_r \cdot b_s$
  - Block seeks: $b_r + n_r$ — inner relation read sequentially

$= $ seeks of s



Nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + n_r \cdot b_s$
  - Block seeks: $b_r + n_r$ — inner relation read sequentially
  - Special case: smaller relation fits in memory



Block nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)

Block nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation

Block nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + b_r \cdot b_s$

$n_r \cdot b_s$

Block nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Block transfers: $b_r + b_r \cdot b_s$
  - Block seeks: $b_r + b_r = 2b_r$

N_r

Indexed nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)

Indexed nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation

for each row in r
search for A in s

Indexed nested-loop join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
  - Total cost: $b_r(t_T + t_S) + n_r * c$
    - c is cost of single selection on s

Merge join

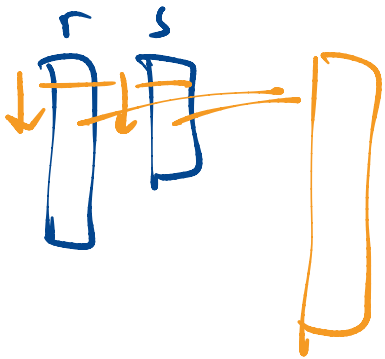
- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)

Merge join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
    - Assume relations are sorted (add cost of sorting)

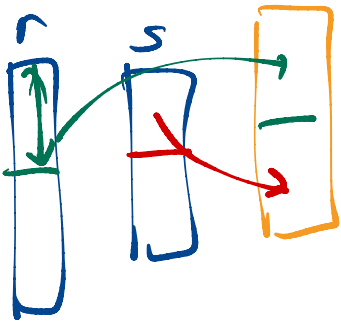
Merge join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
    - Assume relations are sorted (add cost of sorting)
  - Block transfers: $b_r + b_s$



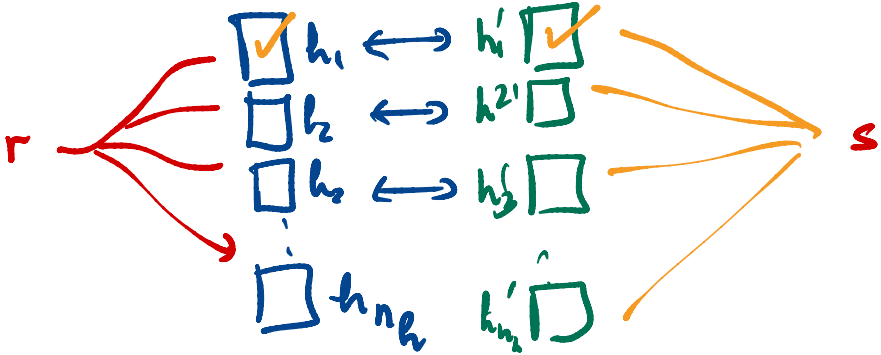
Merge join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Complexity
  - $r \bowtie_\theta s$ — r is outer relation, s is inner relation
    - Assume relations are sorted (add cost of sorting)
  - Block transfers: $b_r + b_s$
  - Block seeks: $\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil$
    - Read a chunk of blocks b_b at a time



Hash join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Hash function on join attribute A, n_h output values



Hash join

- (5000 rows, 100 blocks) *Student* $\bowtie$ *Takes* (10000 rows, 400 blocks)
- Hash function on join attribute A, n_h output values
- Join each pair of hash buckets — build index and probe