CSE462/562: Database Systems (Fall 24)

Lecture 16: Index Scan and Cost Analysis

11/5/2024

University at Buffalo
Department of Computer Science
and Engineering
School of Engineering and Applied Sciences

Last updated: 10/9/2024 3:00 PM

Revisiting simple selection

- $\sigma_p R$
- Notations
  - t_S = seek time in I/O, t_T = page transfer time in I/O
- Linear scan works with any predicate p but has linear I/O cost
  - $c = t_S + N t_T$
  - 1 seek to the start of the file and N pages to read
- Can we do better for special predicate $p = x \in [L, R]$ if
  - we have a B-tree index over x?
  - and/or the file is sorted on x?
  - What about more general predicate p?

Simple selection: index scan

T: # of matching records
F: # of data entries per leaf page
N: # of pages with matching records

- If the file has a B-Tree index I over the search key,
- Basic idea:
  1. Find the first qualifying data entry in the tree
  2. Scan all the data entries and/or fetch the records as needed.

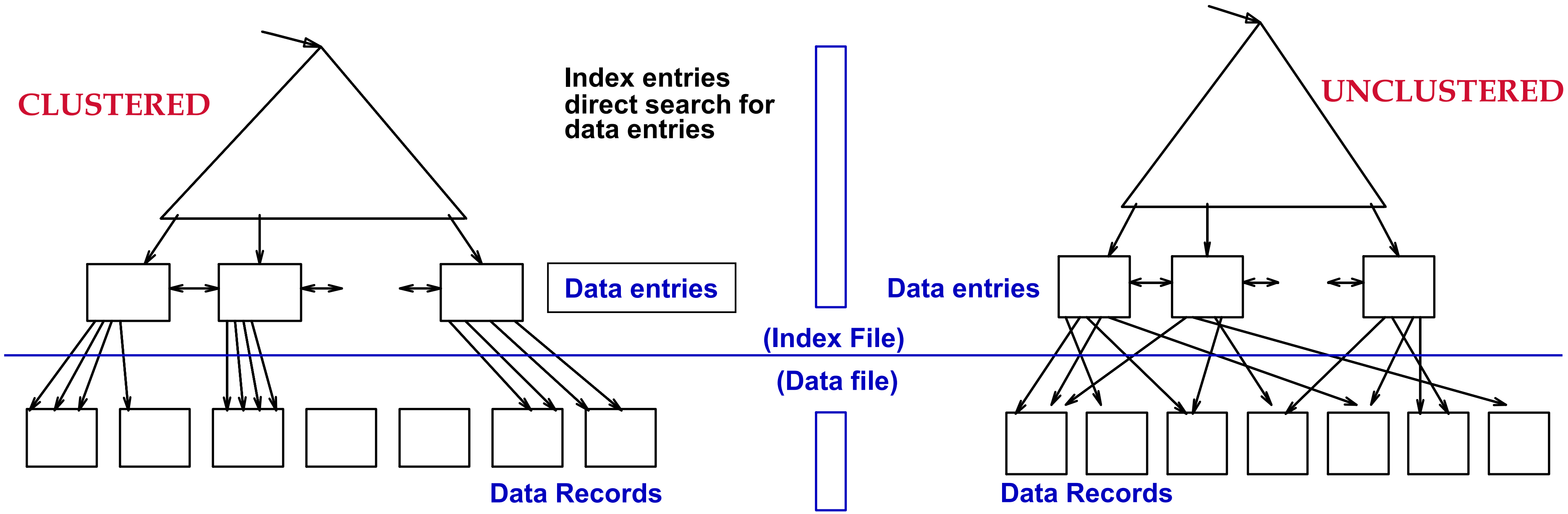
- Three alternatives of data entries:
  - Alternative 1: the record itself (with its key k) – always clustered
  - Alternative 2: <k, record ID of a matching record>
  - Alternative 3: <k, list of record IDs of matching records>

  - For alternative 2 & 3, the index can be clustered or unclustered

  - which can significantly impact the I/O efficiency of index scans

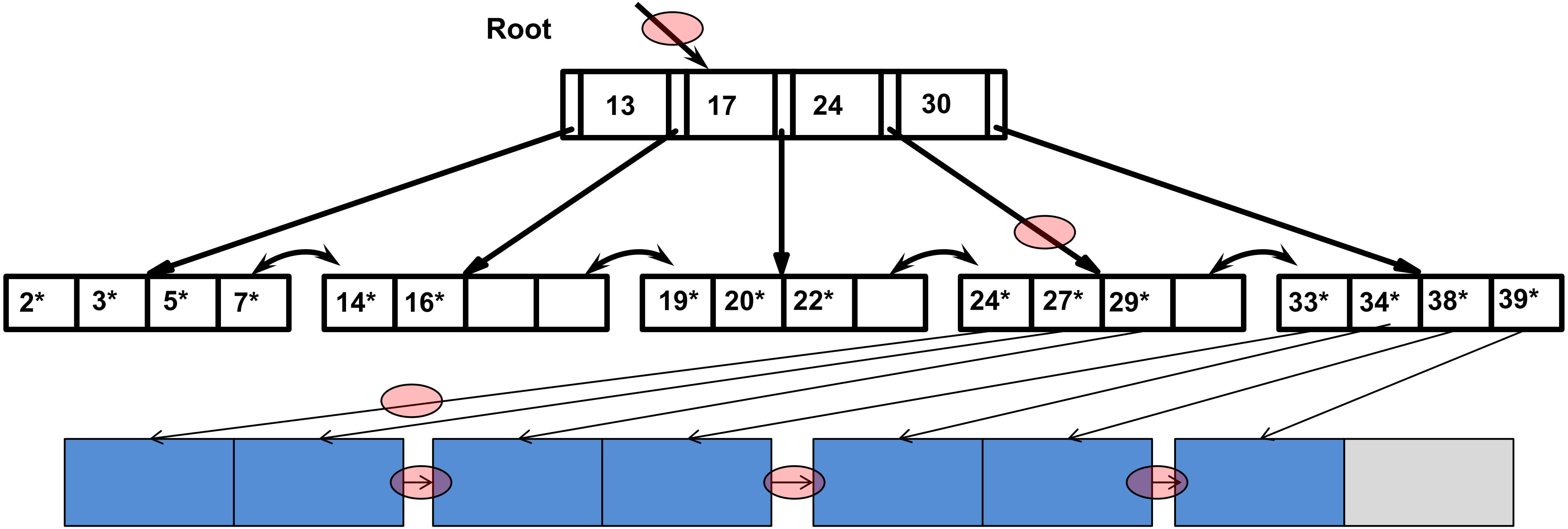
Data access cost using B-Tree

- Recall clustered vs. unclustered: if order of data records is the same as, or `close to’, order of index data entries, then called clustered index.
  - ***Cost of using B-Tree to access records varies a lot depending on whether it is clustered or not***



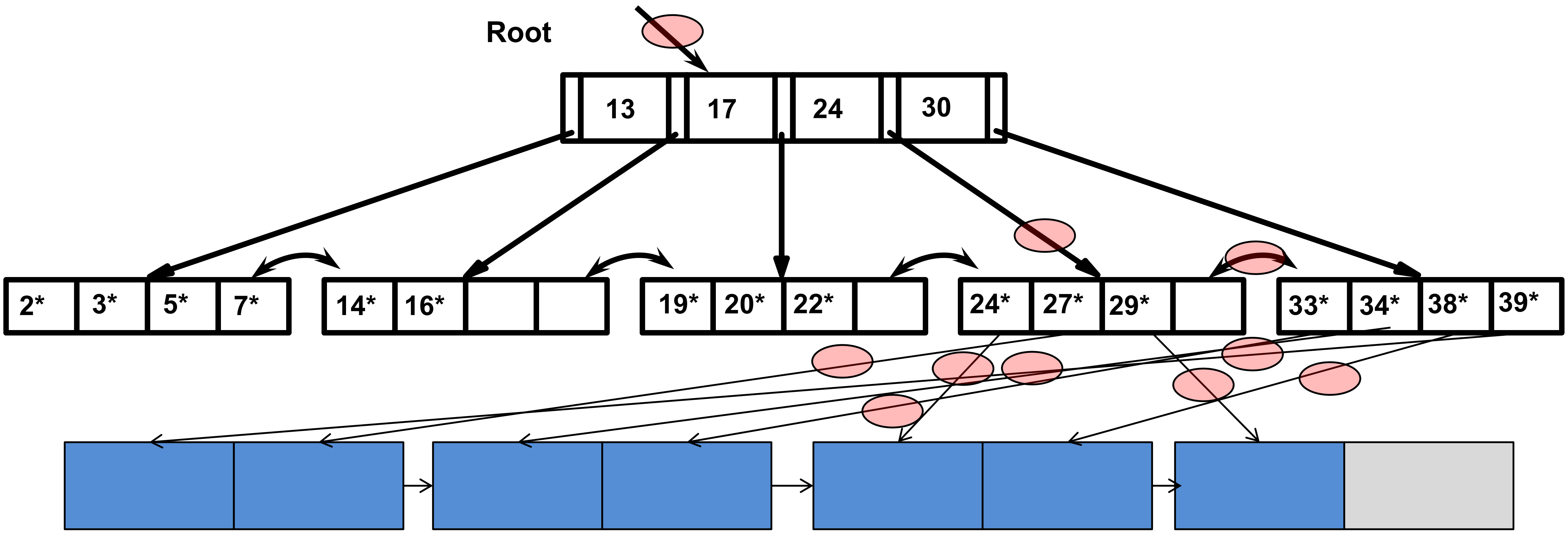
Cost of range scan with clustered B-Tree index

- All records with key >= 24. Clustered index with alternative 2.
  - 6 I/Os
    - 2 random I/O
    - 4 sequential I/O if heap file is laid out sequentially



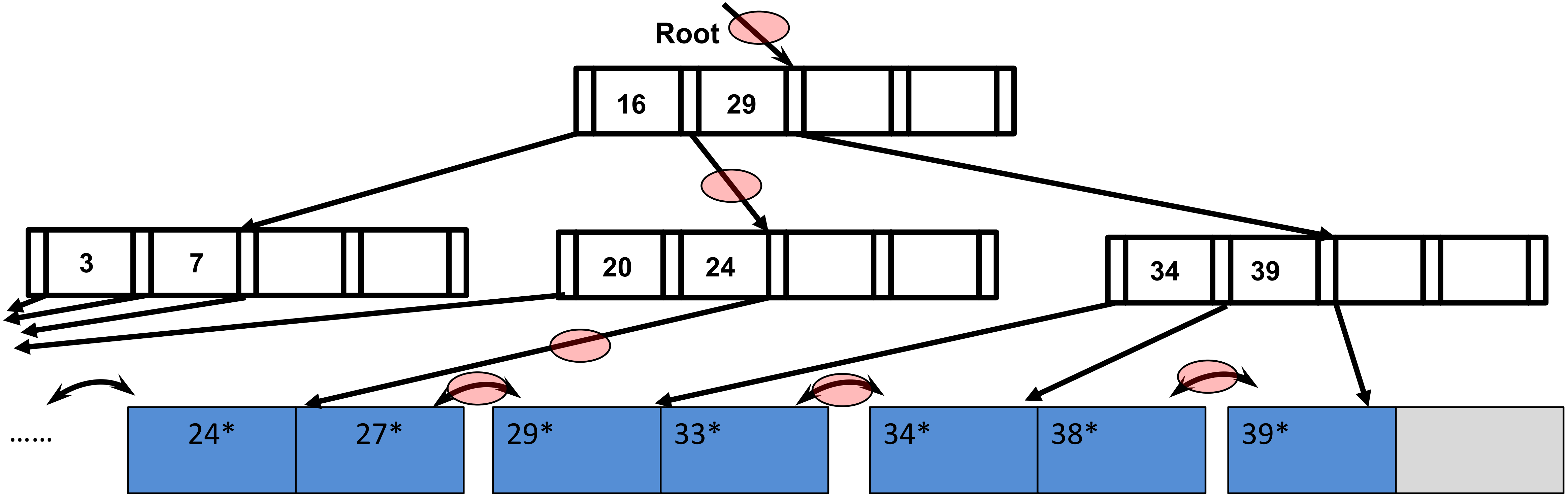
Cost of range scan with unclustered B-Tree index

- All records with key >= 24. Unclustered index with alternative 2.
  - 10 I/Os
    - All random I/Os



Cost of range scan with clustered B-Tree file

- All records with key >= 24. Clustered index with alternative 1.
  - 6 I/Os
    - 3 Random I/O
    - 3 Sequential I/O if the leaf level is sequential in the file



Recap on cost model

- Cost = $N_T \times t_T + S \times t_S$
  - N_T: number of pages read/written; S: number of random I/O
- Assumptions
  - Ignoring the buffer effect for random pages
    - Do consider the private workspace size M for the operators
  - Omitting the cost of transferring output to the user/disk
    - Common to any equivalent plan

Typical t_T and T_S

	HDD*	SSD†
t_T (ms)	0.1	0.01
t_S (ms)	4	0.09

- Notations: for relation R
  - T_R: number of records, N_R: number of pages in its heap file, B_R: (average) number of tuples per page
  - h_I: height of a B-tree index I over the file
  - M: private workspace size in pages
- Running example
  - $t_S = 4\ ms$, $t_T = 0.1\ ms$, *4000-byte page*
  - Student: $\mathrm{R(sid: int, name: varchar(19), login: varchar(19), major: char(2), adm\_year: int)}$
    - 50 bytes/tuple, $B_R = 80$, $T_R = 40{,}000$, $N_R = 500$
  - Enrollment: $\mathrm{E(\underline{sid: int, semester: char(3), cno: int}, grade: double)}$
    - 20 bytes/tuple, $B_E = 200$, $T_E = 200{,}000$, $N_E = 1000$

Simple selection: index scan

T: # of matching records
F: # of data entries per leaf page
N: # of pages with matching records

- If the file has a B-Tree index I over the search key, assuming alternative 2 for data entries
  - cost varies depending on whether it's clustered
- Assuming selectivity is $s = 0.1$, the number of matching records is T and the number of pages with matching records is N, assume $h = 3$
  cost =
  - $h_I \times (t_T + t_S)$ for finding qualifying data entries +
  - cost for retrieving the heap records
    - clustered: $t_S + N \times t_T \approx t_S + \lceil sN_R \rceil \times t_T$ (total $= 12.3 + 9 = 21.3\ ms$)
    - unclustered: $\left(\left\lceil \frac{T}{F} \right\rceil - 1\right) \times t_T + T \times (t_T + t_S)$
      $= \left(\left\lceil \frac{\lceil sT_R \rceil}{F} \right\rceil - 1\right) \times t_T + \lceil sT_R \rceil \times (t_T + t_S)$ (total = $12.3 + 16401.3 = 16413.3\ ms$)
      - can we do better?

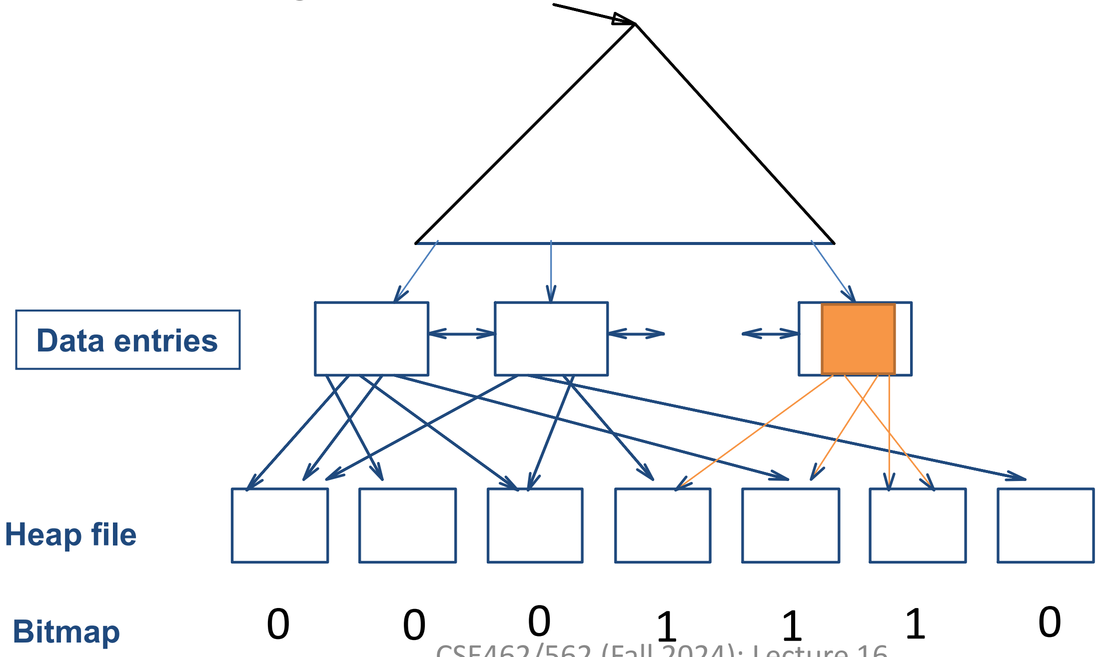
Trade-offs with B-Tree

- Clustered B-Tree
  - One per table
  - Both are good for large range scans, small range scans and point lookups
  - Alternative 2/3 (clustered index)
    - A bit easier to maintain – can be lax on the heap record order (“close to” the data entry order)
  - Alternative 1 (clustered file)
    - Harder to maintain – strictly clustered
      - Need to reorganize the leaf level to make sure they are sequential
    - Save space on data entries (no duplication of keys)
    - Might have larger tree height

- Unclustered B-Tree
  - Usually alternative 2/3
  - Easiest to maintain
  - Not very efficient when range scan covers too many records
    - Rule of thumb: Scan no more than a tiny fraction of rows
      e.g., 0.01% on 7200 rpm HDD, 0.1% on consumer-level Nand SSD
      (empirical value, it may vary depending on your DBMS and storage device)

Simple selection: bitmap index scan

- Refinement for unclustered index scan: bitmap index scan
    1. Initialize a bitmap with one bit for each page in the file (usually fits in mem even for a large file)
    2. Find the first qualifying data entry
    3. Scan all the data entries and mark all the unique pages with the matching records in the bitmap
    4. Scan all the pages with bit 1 (linear scan on page)
- Alternative: collect all RID in memory in step 3, sort and fetch tuples in RID order
    - more expensive unless RIDs fit in memory
    - might make sense for faster storage (thus CPU cost matters)



Simple selection: bitmap index scan

T: # of matching records
F: # of data entries per leaf page
N: # of pages with matching records

- Cost of bitmap index scan =
  - (tree search) $h \times (t_S + t_T) +$
  - (scan of data entries) $\left(\left\lceil\frac{T}{F}\right\rceil - 1\right) \times t_T +$ (assuming leaf level is consecutive from bulk loading)
  - (scan of data pages) $N \times (t_S + t_T)$ (when N is small and thus most involve random seeks) or
    $t_S + N \times t_T$ (when N is close to N_R and it's close to sequential scan)
- Example 1 (large selectivity): $s = 0.9, \mathrm{F} = 300, \mathrm{T} = \lceil s\mathrm{T_R} \rceil = 36000, \mathrm{N} = 500$ =>
  cost = $4.1 \times 3 + 0.1 \times \left(\lceil\frac{36000}{300}\rceil - 1\right) + 4 + 0.1 \times 500 = 78.2\ ms$ (unclustered)
  vs $4.1 \times 3 + 4 + 0.1 \times \lceil 0.9 \times 500 \rceil = 61.3\ ms$ (clustered)
- Example 2 (moderate selectivity): $s = 0.1, F = 300, T = \lceil sT_R \rceil = 4000, \mathrm{E[N]} \approx 500$ (think: why?)
  cost = $4.1 \times 3 + 0.1 \times \left(\lceil\frac{4000}{300}\rceil - 1\right) + 4 + 0.1 \times 500 = 67.6\ ms$ (unclustered)
  vs $4.1 \times 3 + 4 + 0.1 \times \lceil 0.1 \times 500 \rceil = 21.3\ ms$ (clustered)
- Example 3 (small selectivity): $s = 0.0001, F = 300, T = \lceil sT_R \rceil = 4, N = 4$
  cost = $4.1 \times 3 + 0.1 \times \left(\lceil\frac{4}{300}\rceil - 1\right) + 4.1 \times 4 = 28.7\ ms$ (unclustered)
  vs $4.1 \times 3 + 4 + 0.1 \times \lceil 0.0001 \times 500 \rceil = 16.4\ ms$ (clustered)
- Trade-offs:
  - Only slightly more expensive than a linear scan when selectivity is close to 1
  - Only slightly more expensive than a regular secondary index scan when selectivity is close to 0 (<< linear scan)
  - Only works poorly when the selectivity is moderate -- better off with clustered index
    - To show that, let $I_i = 1$ if page i has any matching record (an indicator variable) and assume uniform distribution in search key
    - $E[N] = \sum_{1 \le i \le N_R} E[I_i] = \sum_{1 \le i \le N_R} \Pr\{I_i = 1\} = N_R(1 - (1 - s)^{B_R})$

Analysis of B-Tree storage cost

- Suppose the usable page size is P (bytes), each record is r (bytes), the index key is k bytes, record ID or page number is q bytes, and N records in total in the heap file.
- Assume we use alternative 2 for the data entries.
- Bottom-up analysis:
  - Number of pages in the heap file: $M = \lceil \frac{N}{\lfloor P/r \rfloor} \rceil$.
  - Number of data entries: N (one per record)
  - Size of a data entry: $k + q$ bytes (without considering alignments)
  - Number of pages in leaf level:
    - $N' = \lceil \frac{N}{\lfloor P/(k+q) \rfloor} \rceil$
    - If the average leaf page utilization ratio is u:
$$N' = \lceil \frac{N}{\lfloor P * u/(k+q) \rfloor} \rceil$$
    - Let B be the number of data entries per leaf page
      - $B = \lfloor P * u/(k+q) \rfloor$

Analysis of B-Tree storage cost

fill factor: the default utilization ratio when bulk loading the tree

- Internal levels:
  - Fan-out/number of index entries per page $f = \left\lfloor \frac{P \times u - q}{k+q} \right\rfloor + 1$ (u is the average utilization ratio: [0.5, 1))
  - Number of entries in the index level right above the leaf level: N’ (one entry per leaf-level page)
  - Number of pages required in this level: N'/f
  - Number of entries in the level above: N'/f
  - Number of pages in the level above: N'/f^2
  - Recursively pages in each level:
    - N’, N’/f, N’/f^2 , N’/f^3 …. 1=N’/f^{h-1}
    - So $h = \lceil \log_f N' \rceil + 1 = \left\lceil \log_f \lceil \frac{N}{B} \rceil \right\rceil + 1$
    - total number of internal pages $1 + f + \ \ldots + f^{h-1} = \frac{f^h - 1}{f - 1} = O(N') = O(N/B)$
- Total number of pages in a B-Tree: $O(N') = O(\frac{N}{B})$

Exercises: cost analysis of B-tree index scans

- Example
  - page size = 4096 B
  - For Table A(x, y, z), record length = 64, sizeof(x) == 8 and sizeof(y) == 8.
    - number of records = $2^{20} = 1{,}048{,}576$
  - There're equal number of records with $x > 0$ and $x \leq 0$
  - There's $2^{10} = 1024$ records with $y = 1$
- Assumptions:
  - No page header overhead
  - record id and page id are both 8 bytes
  - no alignment padding needed for index and data entries, no record header overhead
  - Fill factor = 80% for all pages.
  - Ignore the caching effect of buffer pool -> each page access = 1 I/O

- Heap file:
  - Number of pages:
  - Cost of finding all records with $y = 1\ and\ x > 0$:
  - Cost of finding all records with $x = 1$:
  - Cost of insertion of a record:
  - Cost of deletion of all records with y = 1:

Exercises: cost analysis of B-tree index scans

- Example
  - page size = 4096 B
  - For Table A(x, y, z), record length = 64, sizeof(x) == 8 and sizeof(y) == 8.
    - number of records = $2^{20} = 1{,}048{,}576$
  - There're equal number of records with $x > 0$ and $x \leq 0$
  - There's $2^{10} = 1024$ records with $y = 1$
- Assumptions:
  - No page header overhead
  - record id and page id are both 8 bytes
  - no alignment padding needed for index and data entries, no record header overhead
  - Fill factor = 80% for all pages.
  - Ignore the caching effect of buffer pool -> each page access = 1 I/O

- B-tree file over (y), alt. 1:
  - Number of pages:
  - Cost of finding all records with $y = 1\ and\ x > 0$:
  - Cost of finding all records with $x = 1$:
  - Cost of insertion of a record:
  - Cost of deletion of all records with y = 1:

Exercises: cost analysis of B-tree index scans

- Example
  - page size = 4096 B
  - For Table A(x, y, z), record length = 64, sizeof(x) == 8 and sizeof(y) == 8.
    - number of records = $2^{20} = 1{,}048{,}576$
  - There're equal number of records with $x > 0$ and $x \leq 0$
  - There's $2^{10} = 1024$ records with $y = 1$
- Assumptions:
  - No page header overhead
  - record id and page id are both 8 bytes
  - no alignment padding needed for index and data entries, no record header overhead
  - Fill factor = 80% for all pages.
  - Ignore the caching effect of buffer pool -> each page access = 1 I/O

- B-tree file over (y), alt. 2 and clustered:
  - Number of pages:
  - Cost of finding all records with $y = 1\ and\ x > 0$:
  - Cost of finding all records with $x = 1$:
  - Cost of insertion of a record:
  - Cost of deletion of all records with y = 1:

Exercises: cost analysis of B-tree index scans

- Example
  - page size = 4096 B
  - For Table A(x, y, z), record length = 64, sizeof(x) == 8 and sizeof(y) == 8.
    - number of records = $2^{20} = 1{,}048{,}576$
  - There're equal number of records with $x > 0$ and $x \leq 0$
  - There's $2^{10} = 1024$ records with $y = 1$
- Assumptions:
  - No page header overhead
  - record id and page id are both 8 bytes
  - no alignment padding needed for index and data entries, no record header overhead
  - Fill factor = 80% for all pages.
  - Ignore the caching effect of buffer pool -> each page access = 1 I/O
- B-tree file over (y), alt. 2 and unclustered:
  - Number of pages:
  - Cost of finding all records with $y = 1\ and\ x > 0$:
  - Cost of finding all records with $x = 1$:
  - Cost of insertion of a record:
  - Cost of deletion of all records with y = 1:

Exercises: cost analysis of B-tree index scans

- Example
  - page size = 4096 B
  - For Table A(x, y, z), record length = 64, sizeof(x) == 8 and sizeof(y) == 8.
    - number of records = $2^{20} = 1{,}048{,}576$
  - There're equal number of records with $x > 0$ and $x \leq 0$
  - There's $2^{10} = 1024$ records with $y = 1$
- Assumptions:
  - No page header overhead
  - record id and page id are both 8 bytes
  - no alignment padding needed for index and data entries, no record header overhead
  - Fill factor = 80% for all pages.
  - Ignore the caching effect of buffer pool -> each page access = 1 I/O

- B-tree file over (y, x), alt. 2 and clustered:
  - Number of pages:
  - Cost of finding all records with $y = 1 \ and \ x > 0$:
  - Cost of finding all records with $x = 1$:
  - Cost of insertion of a record:
  - Cost of deletion of all records with y = 1:

Exercises: cost analysis of B-tree index scans

- Example
  - page size = 4096 B
  - For Table A(x, y, z), record length = 64, sizeof(x) == 8 and sizeof(y) == 8.
    - number of records = $2^{20} = 1{,}048{,}576$
  - There're equal number of records with $x > 0$ and $x \leq 0$
  - There's $2^{10} = 1024$ records with $y = 1$
- Assumptions:
  - No page header overhead
  - record id and page id are both 8 bytes
  - no alignment padding needed for index and data entries, no record header overhead
  - Fill factor = 80% for all pages.
  - Ignore the caching effect of buffer pool -> each page access = 1 I/O

- B-tree file over (y, x), alt. 2 and unclustered:
  - Number of pages:
  - Cost of finding all records with $y = 1\ and\ x > 0$:
  - Cost of finding all records with $x = 1$:
  - Cost of insertion of a record:
  - Cost of deletion of all records with y = 1: