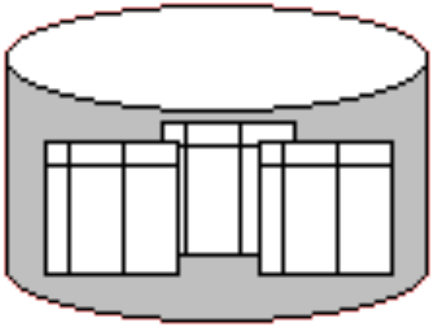
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Συστήματα Διαχείρισης Βάσεων Δεδομένων

Φροντιστήριο 2: File Organization Examples

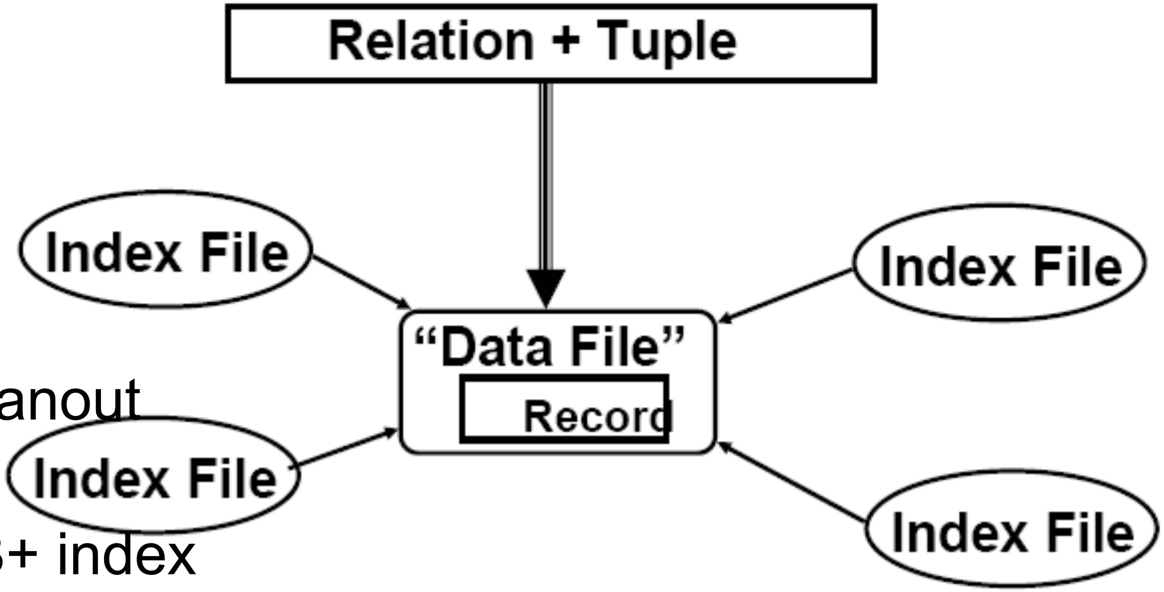
Δημήτρης Πλεξουσάκης
Τμήμα Επιστήμης Υπολογιστών

FILE ORGANIZATION EXAMPLES



Comparison of File Organizations

- Heap
  - unordered file
- Sorted (Sequential) Files
  - ordered by some field(s)
- Clustered B+ index
  - balanced tree with high fanout
- Heap file with unclustered B+ index
  - file is unordered
- Heap file with unclustered hash index
  - "calculate" position of the record



Heap Files

- Rows appended to end of file as they are inserted
  - Hence the file is unordered

- Deleted rows create gaps in file
  - File must be periodically compacted to recover space

Transcript Stored as a Heap File

666666	MGT123	F1994	4.0
123456	CS305	S1996	4.0
987654	CS305	F1995	2.0

page 0

717171	CS315	S1997	4.0
666666	EE101	S1998	3.0
765432	MAT123	S1996	2.0
515151	EE101	F1995	3.0

page 1

234567	CS305	S1999	4.0
878787	MGT123	S1996	3.0

page 2

Heap File - Performance

- Assume file contains *F* pages

- Inserting a row (with duplicate checking):
  - Access path is scan
  - Avg. *F/2* page transfers if row already exists
  - *F+1* page transfers if row does not already exist

- Deleting a row:
  - Access path is scan
  - Avg. *F/2+1* page transfers if row exists
  - *F* page transfers if row does not exist

Heap File - Performance

- Query
  - Access path is scan
  - Organization efficient if query returns all rows and order of access is not important
    - `SELECT * FROM Transcript`
  - Organization inefficient if a *unique* row is requested
    - Avg. *F/2* pages read to get information from a single page

```
SELECT  T.Grade
FROM  Transcript T
WHERE  T.StudId=12345 AND  T.CrsCode =‘CS305’
                AND  T.Semester = ‘S2000’
```

Heap File - Performance

- Organization inefficient when a subset of rows is requested: *F* pages must be read

```
SELECT  T.CrsCode, T.Grade
FROM  Transcript T                   -- equality search
WHERE  T.StudId = 123456

SELECT  T.StudId, T.CrsCode
FROM  Transcript T                   -- range search
WHERE  T.Grade BETWEEN 2.0 AND 4.0
```

Sorted File

- Rows are sorted based on some attribute(s)
  - Access path is binary search
  - Equality or range query based on that attribute has cost log_2F to retrieve page containing first row
  - Successive rows are in same (or successive) page(s) and cache hits are likely
  - By storing all pages on the same track, seek time can be minimized
- Example - Transcript sorted on StudId :

```
SELECT  T.Course,
T.Grade
FROM  Transcript T
WHERE  T.StudId = 123456
```

```
SELECT  T.Course, T.Grade
FROM  Transcript T
WHERE T.StudId BETWEEN
      111111 AND 199999
```

Transcript Stored as a Sorted File

111111	MGT123	F1994	4.0
111111	CS305	S1996	4.0
123456	CS305	F1995	2.0

page 0

123456	CS315	S1997	4.0
123456	EE101	S1998	3.0
232323	MAT123	S1996	2.0
234567	EE101	F1995	3.0

page 1

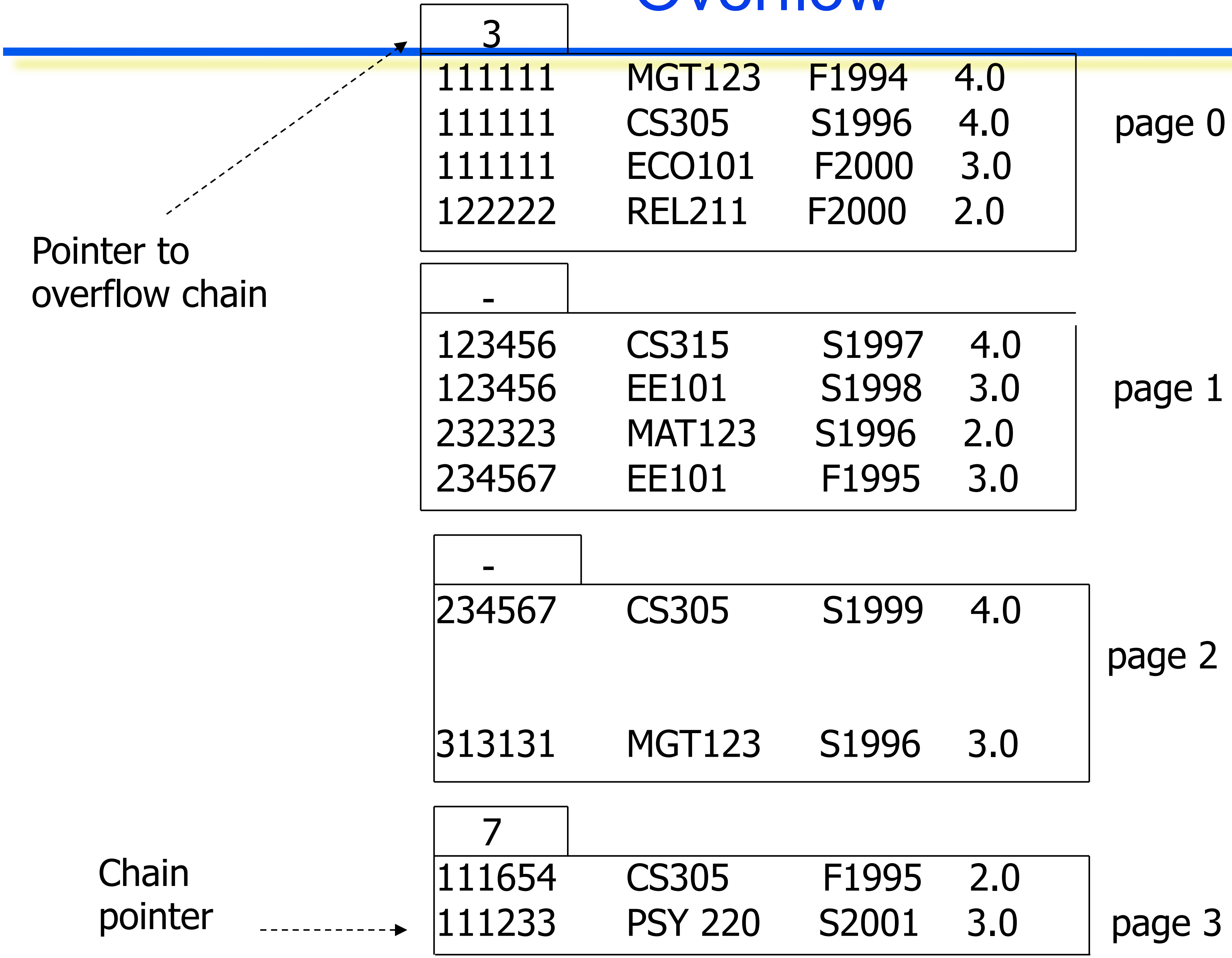
234567	CS305	S1999	4.0
313131	MGT123	S1996	3.0

page 2

Maintaining Sorted Order

- Problem: After the correct position for an insert has been determined, inserting the row requires (on average) *F/2* reads and *F/2* writes
- (Partial) Solution 1: Leave empty space in each page: *fillfactor*
- (Partial) Solution 2: Use overflow pages (*chains*)
  - Disadvantages:
    - Successive pages no longer stored contiguously
    - Overflow chain not sorted, hence cost no longer $log_2 F$

Overflow



3

111111	MGT123	F1994	4.0
111111	CS305	S1996	4.0
111111	ECO101	F2000	3.0
122222	REL211	F2000	2.0

page 0

Pointer to overflow chain

-

123456	CS315	S1997	4.0
123456	EE101	S1998	3.0
232323	MAT123	S1996	2.0
234567	EE101	F1995	3.0

page 1

-

234567	CS305	S1999	4.0
313131	MGT123	S1996	3.0

page 2

7

111654	CS305	F1995	2.0
111233	PSY 220	S2001	3.0

page 3

Chain pointer

Cost Model for Analysis

NOTATION:

- **B:** The number of data pages
- **R:** Number of records per page
- **F:** Fanout of B-tree (number of children for each non-leaf node)

NOTES:

- Average-case analysis; based on several simplistic assumptions (see text for more detail)
- CPU costs are ignored

I/O Cost: Heap File

B: Number of data pages (packed)
R: Number of records per page

	Heap File
Scan all records	B
Equality Search	0.5 B
Range Search	B
Insert	2
Delete	0.5 B + 1

Scan ½ file on average → Equality Search

Scan entire file since unordered → Range Search

Load last page, add, write out → Insert

Find page, delete record, write out → Delete

I/O Cost: Sorted File (on Search Key)

B: Number of data pages (packed)
R: Number of records per page

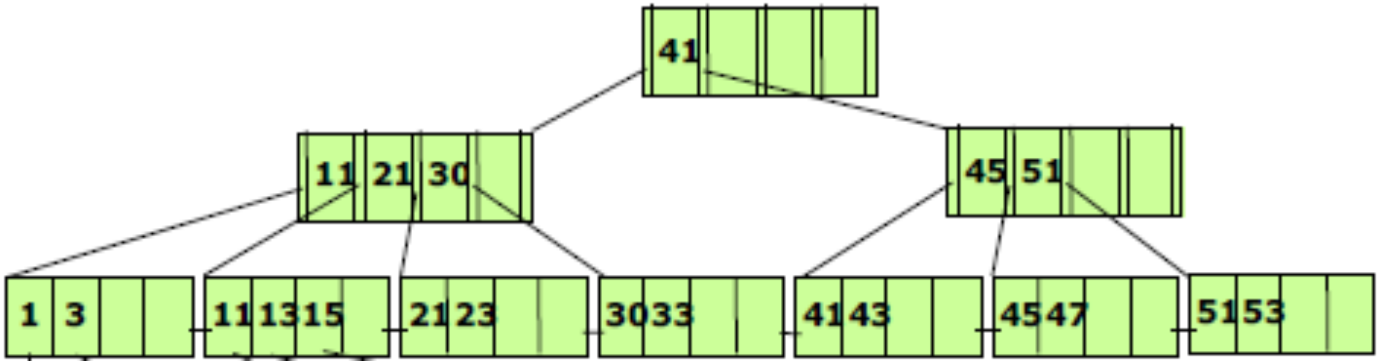
	Sorted File
Scan all records	B
Equality Search	$\log_2 B$
Range Search	$\log_2 B$ + # matching pages
Insert	$\log_2 B + B$
Delete	$\log_2 B + B$

Binary Search → Equality Search

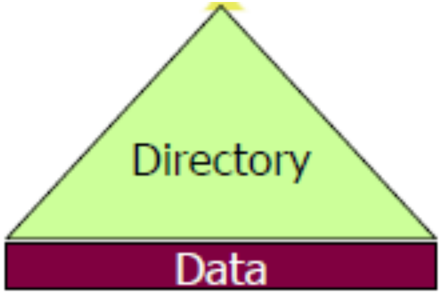
Find first, then sequentially retrieve → Range Search

$\log_2 B+0.5B+0.5B$: Assume search record in middle of file. To compact file, need to read in remaining 0.5B pages, adjust, then write out. → Insert, Delete

Clustered B+ Tree File Example (Alt #1)



- B+ Tree: Balanced trees (all paths from root to leaf have the same length) with data stored only in leaf nodes
- Alternative #1 index: Index and data records stored together. In the B+ tree index case, data records are stored in the leaf nodes.
- Clustered index: The order of data records in the file is the same as the order of data entries in the index
    - Alternative #1 is clustered **by definition**.



I/O Cost: Clustered B+ Tree File (Alt #1)

B: Number of data pages (packed)
R: Number of records per page
F: Fanout of B-Tree

Assume 67% occupancy

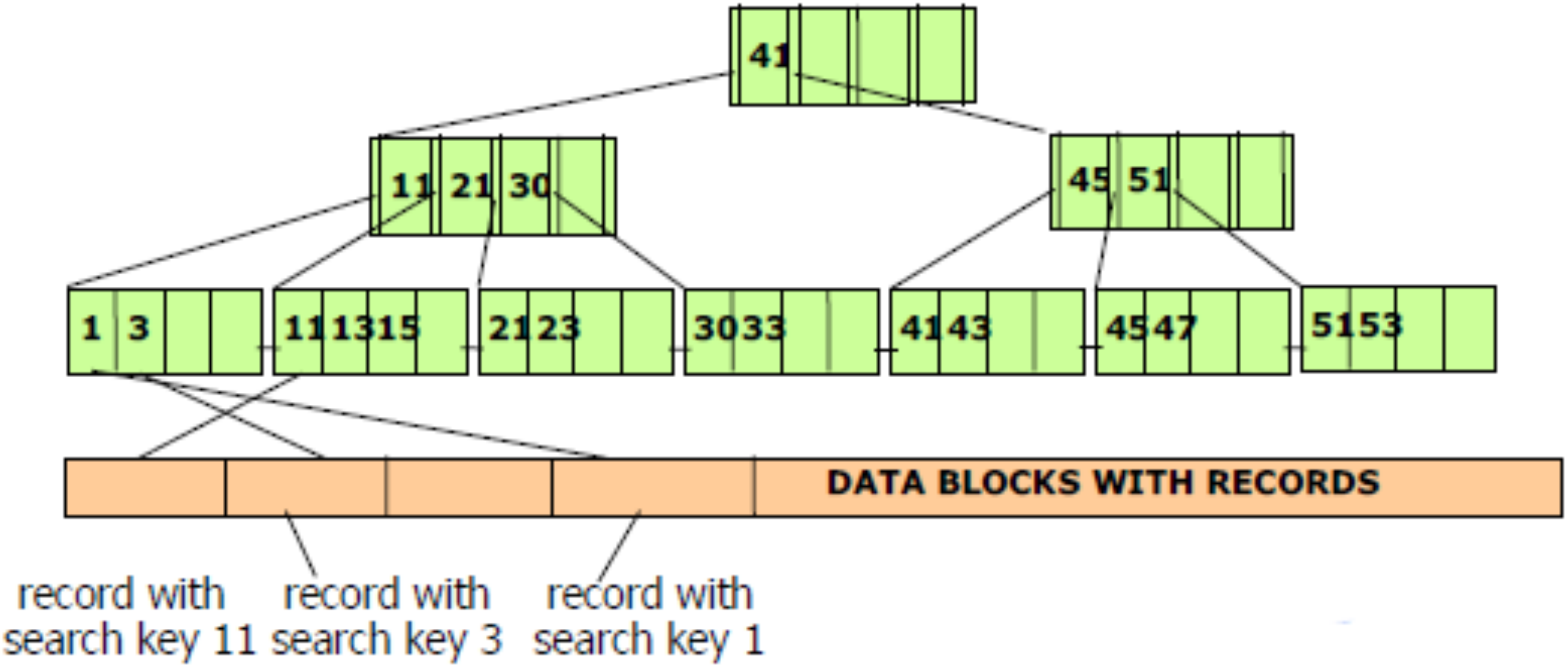
This is the height of the tree (root to appropriate leaf). Note that the leaf holds actual records.

Use the pointers to additional pages

Find leaf, insert or delete, write out

	Clustered Tree
Scan all records	1.5 B
Equality Search	$\log_F (1.5\ B)$
Range Search	$\log_F (1.5\ B)$ + #matching pages
Insert	$\log_F (1.5\ B) + 1$
Delete	$\log_F (1.5\ B) + 1$

Heap File with Unclustered B+ Tree Example



I/O Cost: Heap File with Unclustered B+ Tree

B: Number of data pages (packed)
R: Number of records per page
F: Fanout of B-Tree

Index size of 10% data size. Scan all index leaf pages 0.1(1.5B)= 0.15B. For each record, one I/O.

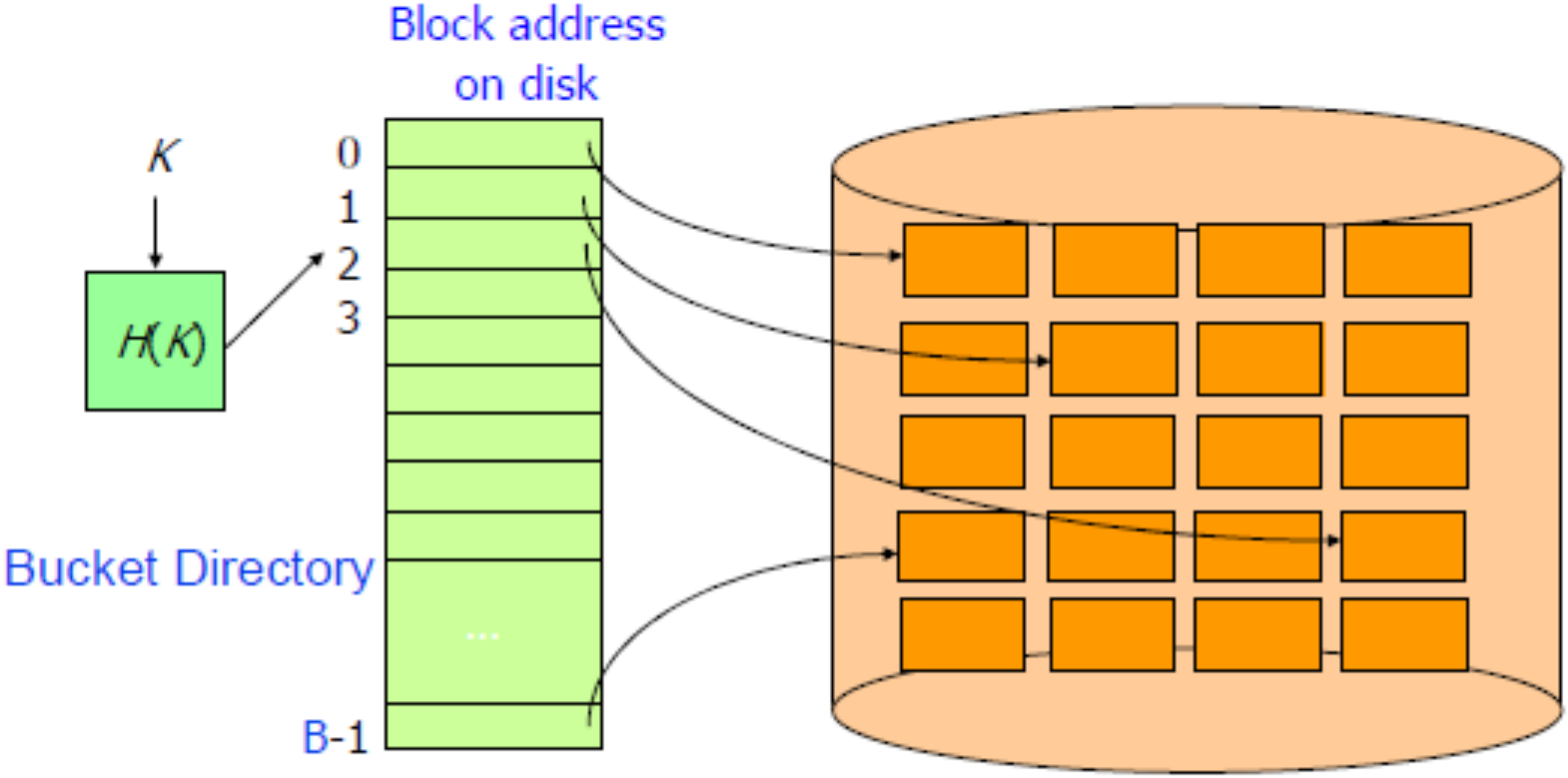
This is the height of the tree (root to appropriate leaf). Then load page on which record resides.

Note one I/O for each matching record since unclustered index.

Find corresponding index data entry, read corresponding page, insert or delete, write out page and corresponding index block.

	Unclustered Tree
Scan all records	$BR+0.15B$
Equality Search	$\log_F (0.15B) + 1$
Range Search	$\log_F (0.15B) +$ #matching records
Insert	$\log_F (0.15B) + 3$
Delete	$\log_F (0.15B) + 3$

Heap File with Hash Index Example



I/O Cost: Heap File with Unclustered Hash Index

B: Number of data pages (packed)
R: Number of records per page

Assume 80% occupancy, index size of 10% data size, so 0.10(1.25B) index pages. For each record, one I/O.

Retrieve hash bucket, then appropriate page.

Hash index doesn't help. Scan all records (without index this would be 1.25B)

Read and write hash bucket, read and write appropriate page.

	Hash Index
Scan all records	BR+0.125B
Equality Search	2
Range Search	BR+0.125B (or 1.25B)
Insert	4
Delete	4

I/O Cost of Operations: Summary

B: The number of data pages, **R:** Number of records per page,
F: Fanout of B-Tree

	Heap File	Sorted File	Clustered Tree	Unclustered Tree	Unclustered Hash Index
Scan all records	B	B	1.5 B	B(R+0.15)	B(R+0.125)
Equality Search	0.5 B	$\log_2 B$	$\log_F(1.5\ B)$	$\log_F(0.15B) + 1$	2
Range Search	B	$\log_2 B$ + #matching pages	$\log_F(1.5\ B)$ + #matching pages	$\log_F(0.15B)$ + #matching records	B(R+0.125)
Insert	2	$\log_2 B + B$	$\log_F(1.5\ B) + 1$	$\log_F(0.15B) + 3$	4
Delete	2	$\log_2 B + B$	$\log_F(1.5\ B) + 1$	$\log_F(0.15B) + 3$	4

Heap Files Example

- Consider a problem of data storage/retrieval:
  - The application has 10,000,000 records of 100 bytes each
  - The key value is a single attribute employee_number

- Available on the machine for an application:
  - 64MB main memory
  - 1MB main memory allocation for buffer (area of main memory in which data from disk is placed)
  - 64GB disk

Questions and Answers 1-4

1. Does the data fit on the disk? Yes, 1GB (1,000,000,000) is the requirement and 64GB is available
2. Does all the data fit in main memory at once? No, main memory even if all available holds only 64MB (64,000.000)
3. Assume that 2,000 records fit per disk page
   a) How many pages are required to hold the data? So R=2,000 (records/page) and there are 10,000,000 records so number of pages needed (B) is 10,000,000/2,000 = 5,000
   b) Given an average disk access time D=0.015 secs and the average time to process a record C=10^{-7} secs/record, how long does it take for an exhaustive search of the file stored as a heap? B(D+RC) = 5000(0.015 + 2000*10^{-7}) = 76 secs
4. Repeat 3) with a)100 records/page and b)10,000 records/page
   a) If R=100 then B=100,000 so B(D+RC) = 100,000(0.015+ 100*10^{-7}) = 1501 secs
   b) If R=10,000 then B= 1000 so B(D+RC) = 1000(0.015+10000*10^{-7}) = 16 secs

   Note that simply packing more records/page dramatically reduces the process time. So increase R to as high a figure as the amount of main memory available

Questions and Answers 5-7

5. Can the number of records/page be increased further? No as buffer size of 1MB (1,000,000) in main memory can only fit 10,000 records at 100 bytes each
6. What is the average cost of finding a particular record in this file with 2000 records/page using:
   a) heap access method? $B(D+RC)/2=2500(0.015+2000*10^{-7})=38$ secs
   b) sequential (sorted) access method? $\log_2 B * D + C * \log_2 R = 13* 0.015 + 10^{-7} * 11 \approx 0.195$ secs

   So sequential is much faster than heap in finding a particular record.

   Note: here log to base 2 of X is the power of 2 needed to reach or exceed X. These logs are whole numbers for our purposes. E.g. $\log_2 8 = 3$; $\log_2 256=8$; $\log_2 300=9$
7. What is the average cost of inserting a particular record into this file with 2,000 records/page using:
   a) a heap access method? $2D + C = (2*0.015)+10^{-7} \approx 0.03$ secs
   b) a sequential (sorted) access method? $\log_2 B * D + C * \log_2 R + B*(D + RC)= 13*0.015 + 10^{-7} * 11+5000 * 0.015 + 38 * 2 \approx 151.195$ secs

   So heap is far faster for insertions

Hashed Files Example

- Consider a problem of data storage/retrieval:
  - The application has 5,000,000 records of 100 bytes each
  - The key value is a single attribute employee_number of the form:
    - Annnnnnn (7 n’s)
    - where A is a capital letter and n is a number

- Available on the machine for an application:
  - 64MB main memory
  - 1MB main memory allocation for buffer (area of main memory in which data from disk is placed)
  - 64GB disk

Questions and Answers 1-5

1. What is the total number of possible employee numbers? Annnnnnn – nnnnnnn gives the range 0000000-9999999 (10,000,000 combinations), A gives 26 possibilities (A-Z) so the key can be represented in 260,000,000 ways, ample for 5,000,000 actual records
2. Can you find a hash function which maps the data into 1000 pages? Yes: remove initial letter, divide by 1000 and take the remainder as the page number giving the range of page numbers 0 .. 999
3. How many records will this produce per page? Number of records in file/number of pages in the file, that is 5,000,000/1,000 = 5,000
4. Is this the number of records per page to be used in the implementation? No, aim for about 80% packing so have perhaps 4,000 records/page
5. What assumptions does this hash function make about distribution of key values? That they are random with respect to the hash function. If disproportionately high numbers of key values ended in 000, page 0 would become full very quickly leading to excessive collisions

Questions and Answers 6-7

6. How long will it take to find the record with key E0011223? Trace through the steps taken.

Apply hash function to key value:

Remove initial letter giving 00112233

Find remainder after dividing 00112233 by 1000: giving page number of 233

Retrieve page 233 from disk (transfer to main memory, cost = D)

Search page 233 in main memory (assume found on average by searching half page, cost = 0.5RC)

Total Cost = D + 0.5RC = 0.015 + ($0.5*5000*10^{-7}$) = 0.01530 secs (very fast)

7. How long will it take to insert a record with key J9910657? Trace through the steps taken.

Apply hash function to key value:

Remove initial letter giving 9910657

Find remainder after dividing 9910657 by 1000: giving page number of 657

Retrieve page 657 from disk (transfer to main memory, cost = D)

Insert new record into page 657 in main memory (assume anywhere as in heap, cost = C)

Write page 657 back to disk (cost = D)

Cost = D + C + D= 0.015 + 10^{-7} + 0.015 = 0.0300001 secs (very fast)

Question 8 – Hash Search

8. How long will it take to find all employee numbers in the range K0011200..K0011299?

- How many pages will this be distributed over?
    - Not known. Really need idea of denseness - does every possible key exist?
- Maximum is over pages 200, 201, …., 299 so 100 pages; minimum is over 0 pages (does not exist)
- Because of uncertainty, hash method with any range (even on key) involves complete scan
- Cost = B*(D + RC) =1000*(0.015 + (5000*10^{-7})) = 15.5 secs (very long)

Question 9 – Sorted/Heap

9. How does the cost here compare with that in sorted and heap files?

- Heap cost: search all of file – cost = B(D+RC) = 833*(0.015 + 6,000*10^{-7}) = 13.0 secs (very long, B=833 as can pack 6,000 records per page at 100% full)
- Sorted cost: find initial page by binary chop then find all records in one, possibly two, accesses. Cost = $D\log_2 B + C\log_2 R \approx 0.15$ secs (by far the fastest)
- There is a minor extra cost for sorted. What is it?

Questions 10/11 – Cost K?

10. How long will it take to find all employee numbers beginning with K?
11. How does the cost here compare with that in sorted and heap files?

- Cannot say how many records there are
- Hashed – cost will be search of whole file =B*(D + RC) =1000*(0.015 + (5000*10^{-7})) = 15.5 secs (very long, longer than heap as <100% full)
- Heap – cost will be search of whole file = B(D+RC) = 833(0.015 + 6,000*10^{-7}) = 13.0 secs (very long)
- Sorted – cost will be D $\log_2$B+C$\log_2$R = (0.015*10)+ (10^{-7}*13) ≈ 0.15 secs
- So sorted is very much faster
- Note hashing gives fast single-record access but is slow for searching on ranges

Optimization of Disk-Block Access: Methods

- Disk-arm Scheduling: requests for several blocks may be speeded up by requesting them in the order they will pass under the head
  - If the blocks are on different cylinders, it is advantageous to ask for them in an order that minimizes disk-arm movement
  - Elevator algorithm -- move the disk arm in one direction until all requests from that direction are satisfied, then reverse and repeat
  - Sequential access is 1-2 orders of magnitude faster; random access is about 2 orders of magnitude slower
- Non-volatile write buffers
  - store written data in a RAM buffer rather than on disk
  - write the buffer whenever it becomes full or when no other disk requests are pending
  - buffer must be non-volatile to protect from power failure
    - called non-volatile random-access memory (NV-RAM)
    - typically implemented with battery-backed-up RAM
  - dramatic speedup on writes; with a reasonable-sized buffer write latency essentially disappears
  - why can’t we do the same for reads? (hints: ESP, clustering)

Optimization of Disk-Block Access: Methods

- File organization (Clustering): reduce access time by organizing blocks on disk in a way that corresponds closely to the way we expect them to be accessed
  - sequential files should be kept organized sequentially
  - hierarchical files should be organized with mothers next to daughters
  - for joining tables (relations) put the joining tuples next to each other
  - over time fragmentation can become an issue
    - restoration of disk structure (copy and rewrite, reordered) controls fragmentation
- Log-based file system
  - does not update in-place, rather writes updates to a log disk
    - essentially, a disk functioning as a non-volatile RAM write buffer
  - all access in the log disk is sequential, eliminating seek time
  - eventually updates must be propagated to the original blocks
    - as with NV-RAM write buffers, this can occur at a time when no disk requests are pending
    - the updates can be ordered to minimize arm movement
  - this can generate a high degree of fragmentation on files that require constant updates
    - fragmentation increases seek time for sequential reading of files

Storage Access

- Basic concepts (some already familiar):
  - a block is a contiguous sequence of sectors from a single track; blocks are units of both storage allocation and data transfer
  - a file is a sequence of records stored in fixed-size blocks (pages) on the disk
  - each block (page) has a unique address called BID
  - optimization is done by reducing I/O, seek time, etc.
  - database systems seek to minimize the number of block transfers between the disk and memory. We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.
  - buffer - portion of main memory used to store copies of disk blocks
  - buffer manager - subsystem responsible for allocating buffer space in main memory and handling block transfer between buffer and disk
- Disk-block access methods must take care of some information within each block, as well as information about each block:
  - allocate records (tuples) within blocks
  - support record addressing by address and by value
  - support auxiliary (secondary indexing) file structures for more efficient processing

Τέλος Ενότητας

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημειώματα

Σημείωμα αδειοδότησης

- Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση, Όχι Παράγωγο Έργο 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».

[1] http://creativecommons.org/licenses/by-nc-nd/4.0/

- Ως **Μη Εμπορική** ορίζεται η χρήση:
  - που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
  - που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
  - που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

- Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

.

Σημείωμα Αναφοράς

Copyright Πανεπιστήμιο Κρήτης, Δημήτρης Πλεξουσάκης. «**Συστήματα Διαχείρισης Βάσεων Δεδομένων. Φροντιστήριο 2: File Organization Examples**». Έκδοση: 1.0. Ηράκλειο/Ρέθυμνο 2015. Διαθέσιμο από τη δικτυακή διεύθυνση: http://www.csd.uoc.gr/~hy460/