

# CS 245: Database System Principles

## Notes 4: Indexing

Hector Garcia-Molina

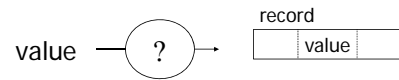
CS 245

Notes 4

1

### Chapter 4

#### Indexing & Hashing



CS 245

Notes 4

2

#### Topics

- Conventional indexes
- B-trees
- Hashing schemes

CS 245

Notes 4

3

#### Sequential File

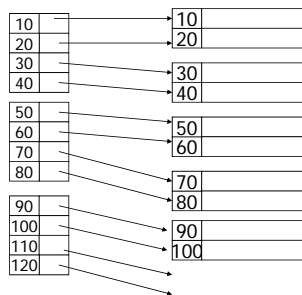
10	
20	
30	
40	
50	
60	
70	
80	
90	
100	

CS 245

Notes 4

4

#### Dense Index Sequential File

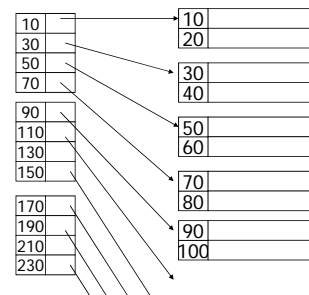


CS 245

Notes 4

5

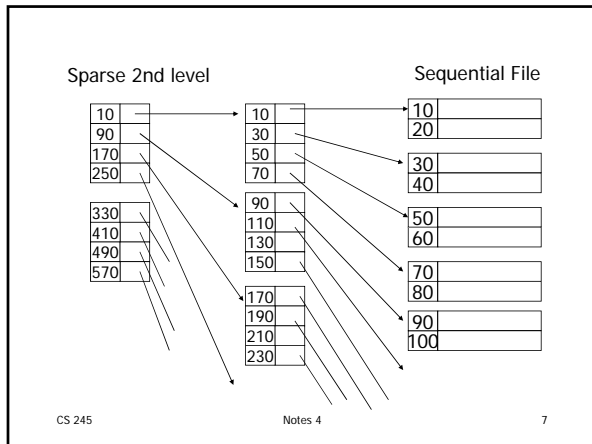
#### Sparse Index Sequential File



CS 245

Notes 4

6



• Comment:  
 {FILE, INDEX} may be contiguous  
 or not (blocks chained)

CS 245 Notes 4 8

Question:

- Can we build a dense, 2nd level index for a dense index?

CS 245 Notes 4 9

Notes on pointers:

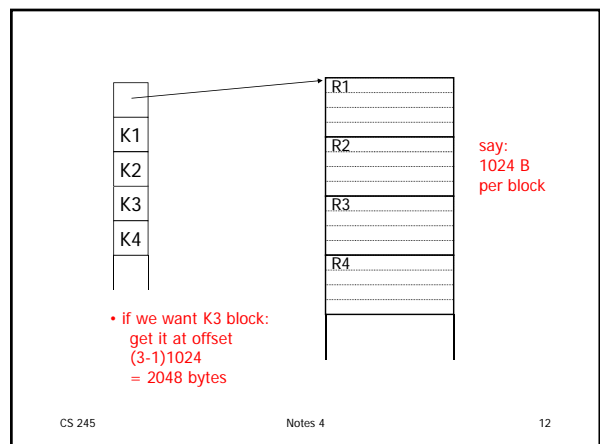
(1) Block pointer (sparse index) can be smaller than record pointer

CS 245 Notes 4 10

Notes on pointers:

(2) If file is contiguous, then we can omit pointers (i.e., compute them)

CS 245 Notes 4 11



## Sparse vs. Dense Tradeoff

- Sparse: Less index space per record can keep more of index in memory
- Dense: Can tell if any record exists without accessing file

(Later:

- sparse better for insertions
- dense needed for secondary indexes)

CS 245

Notes 4

13

## Terms

- Index sequential file
- Search key ( $\neq$  primary key)
- Primary index (on Sequencing field)
- Secondary index
- Dense index (all Search Key values in)
- Sparse index
- Multi-level index

CS 245

Notes 4

14

## Next:

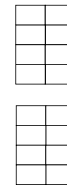
- Duplicate keys
- Deletion/Insertion
- Secondary indexes

CS 245

Notes 4

15

## Duplicate keys



10	
10	
10	
20	
20	
30	
30	
40	
45	

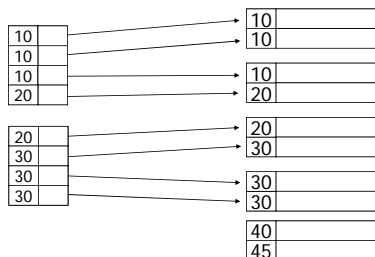
CS 245

Notes 4

16

## Duplicate keys

Dense index, one way to implement?



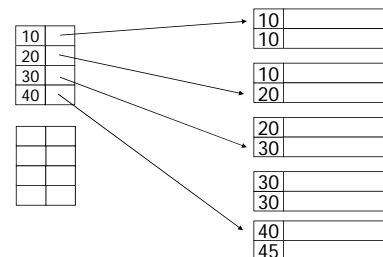
CS 245

Notes 4

17

## Duplicate keys

Dense index, better way?



CS 245

Notes 4

18

### Duplicate keys

#### Sparse index, one way?

careful if looking for 20 or 30!

CS 245      Notes 4      19

### Duplicate keys

#### Sparse index, another way?

– place first new key from block

should this be 40?

CS 245      Notes 4      20

### Summary

#### Duplicate values, primary index

- Index may point to first instance of each value only

CS 245      Notes 4      21

### Deletion from sparse index

CS 245      Notes 4      22

### Deletion from sparse index

– delete record 40

CS 245      Notes 4      23

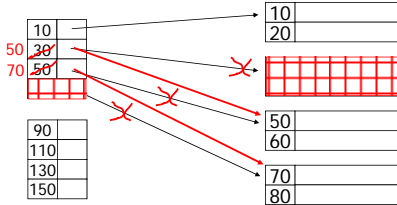
### Deletion from sparse index

– delete record 30

CS 245      Notes 4      24

### Deletion from sparse index

- delete records 30 & 40

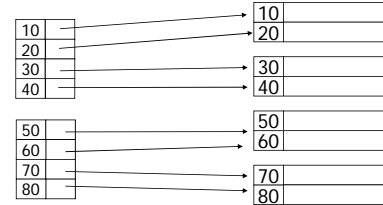


CS 245

Notes 4

25

### Deletion from dense index



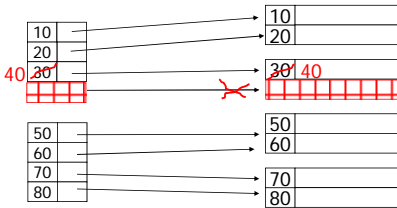
CS 245

Notes 4

26

### Deletion from dense index

- delete record 30

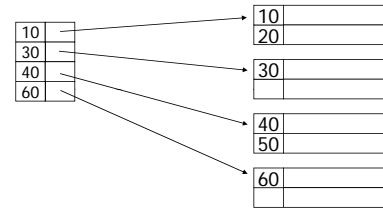


CS 245

Notes 4

27

### Insertion, sparse index case



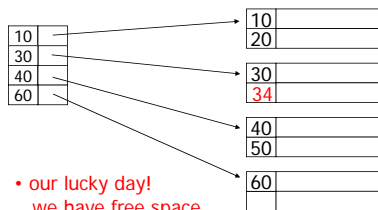
CS 245

Notes 4

28

### Insertion, sparse index case

- insert record 34



• our lucky day!  
we have free space  
where we need it!

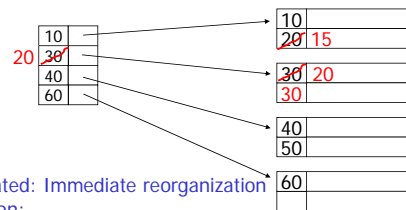
CS 245

Notes 4

29

### Insertion, sparse index case

- insert record 15



- Illustrated: Immediate reorganization
- Variation:
  - insert new block (chained file)
  - update index

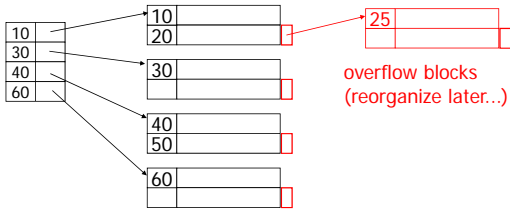
CS 245

Notes 4

30

### Insertion, sparse index case

- insert record 25



CS 245

Notes 4

31

### Insertion, dense index case

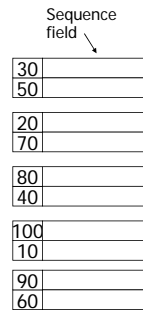
- Similar
- Often more expensive . . .

CS 245

Notes 4

32

### Secondary indexes



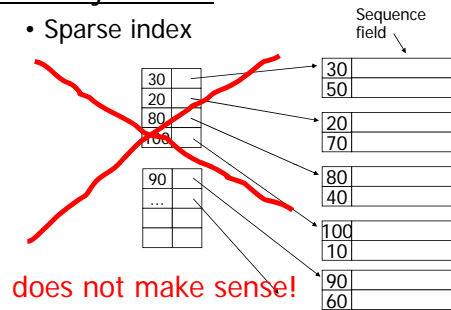
CS 245

Notes 4

33

### Secondary indexes

- Sparse index



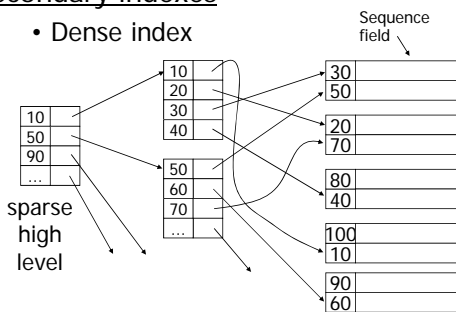
CS 245

Notes 4

34

### Secondary indexes

- Dense index



CS 245

Notes 4

35

### With secondary indexes:

- Lowest level is dense
- Other levels are sparse

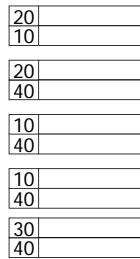
Also: Pointers are record pointers  
(not block pointers; not computed)

CS 245

Notes 4

36

### Duplicate values & secondary indexes



CS 245

Notes 4

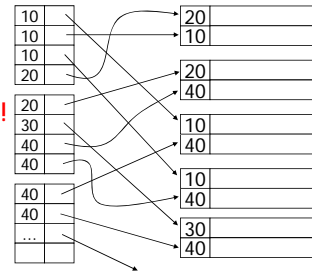
37

### Duplicate values & secondary indexes

one option...

**Problem:**  
excess overhead!

- disk space
- search time



CS 245

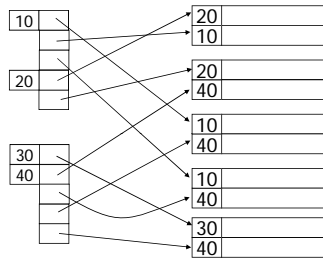
Notes 4

38

### Duplicate values & secondary indexes

another option...

**Problem:**  
variable size records in index!

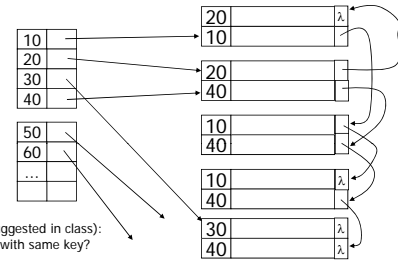


CS 245

Notes 4

39

### Duplicate values & secondary indexes



Another idea (suggested in class):  
Chain records with same key?

**Problems:**

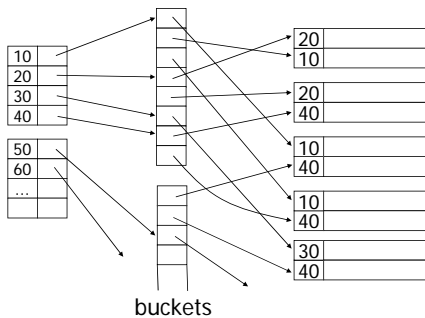
- Need to add fields to records
- Need to follow chain to know records

CS 245

Notes 4

40

### Duplicate values & secondary indexes



buckets

CS 245

Notes 4

41

### Why "bucket" idea is useful

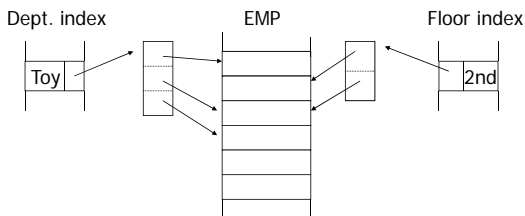
Indexes	Records
Name: primary	EMP (name,dept,floor,...)
Dept: secondary	
Floor: secondary	

CS 245

Notes 4

42

Query: Get employees in  
(Toy Dept)  $\wedge$  (2nd floor)



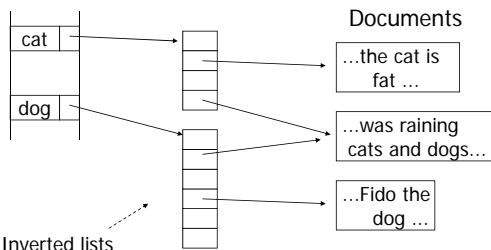
→ Intersect toy bucket and 2nd Floor bucket to get set of matching EMP's

CS 245

Notes 4

43

This idea used in  
text information retrieval



Inverted lists

CS 245

Notes 4

44

## IR QUERIES

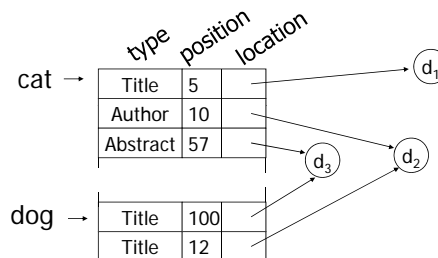
- Find articles with "cat" and "dog"
- Find articles with "cat" or "dog"
- Find articles with "cat" and not "dog"
- Find articles with "cat" in title
- Find articles with "cat" and "dog" within 5 words

CS 245

Notes 4

45

Common technique:  
more info in inverted list



CS 245

Notes 4

46

Posting: an entry in inverted list.  
Represents occurrence of term in article

Size of a list: 1 (in postings) Rare words or miss-spellings  
↓  
10<sup>6</sup> Common words

Size of a posting: 10-15 bits (compressed)

CS 245

Notes 4

47

## IR DISCUSSION

- Stop words
- Truncation
- Thesaurus
- Full text vs. Abstracts
- Vector model

CS 245

Notes 4

48

### Vector space model

w1 w2 w3 w4 w5 w6 w7 ...  
DOC = <1 0 0 1 1 0 0 ...>  
Query= <0 0 1 1 0 0 0 ...>  
PRODUCT =                   1 + ..... = score

CS 245

Notes 4

49

- Tricks to weigh scores + normalize

e.g.: Match on common word not as useful as match on rare words...

CS 245

Notes 4

50

- How to process V.S. Queries?

w1 w2 w3 w4 w5 w6 ...  
Q = < 0 0 0 1 1 0 ... >

CS 245

Notes 4

51

- Try Stanford Libraries
- Try Google, Yahoo, ...

CS 245

Notes 4

52

### Summary so far

- Conventional index
  - Basic Ideas: sparse, dense, multi-level...
  - Duplicate Keys
  - Deletion/Insertion
  - Secondary indexes
    - Buckets of Postings List

CS 245

Notes 4

53

### Conventional indexes

#### Advantage:

- Simple
- Index is sequential file good for scans

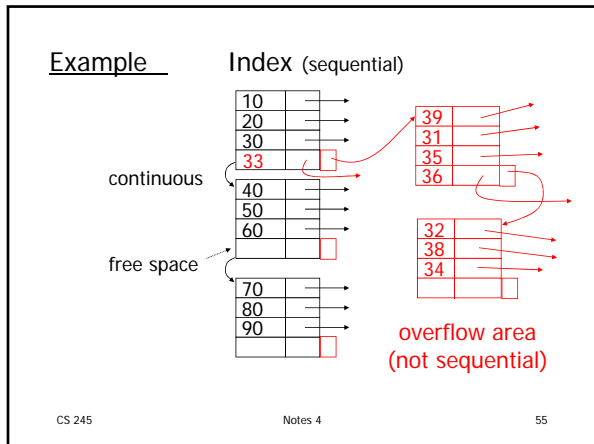
#### Disadvantage:

- Inserts expensive, and/or
- Lose sequentiality & balance

CS 245

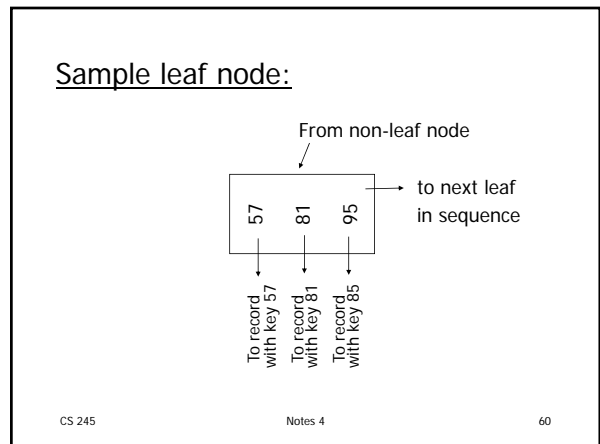
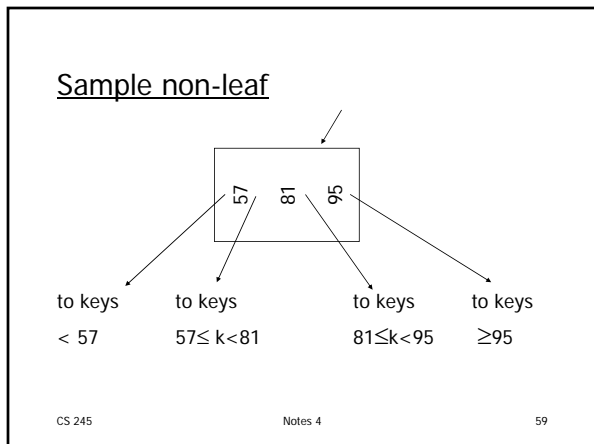
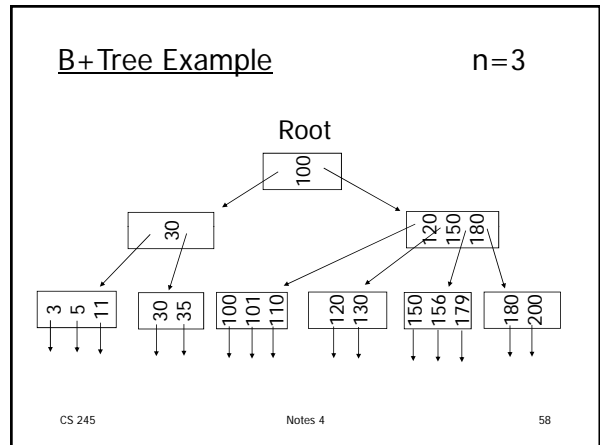
Notes 4

54



- Outline:
- Conventional indexes
  - B-Trees ⇒ NEXT
  - Hashing schemes
- CS 245 Notes 4 56

- NEXT: Another type of index
    - Give up on sequentiality of index
    - Try to get "balance"
- CS 245 Notes 4 57



In textbook's notation  $n=3$

Leaf:

Non-leaf:

CS 245 Notes 4 61

Size of nodes:  $\begin{cases} n+1 \text{ pointers} \\ n \text{ keys} \end{cases}$  (fixed)

CS 245 Notes 4 62

Don't want nodes to be too empty

- Use at least

Non-leaf:  $\lceil (n+1)/2 \rceil$  pointers

Leaf:  $\lfloor (n+1)/2 \rfloor$  pointers to data

CS 245 Notes 4 63

$n=3$

Full node  $\begin{cases} \text{Non-leaf} \\ \text{Leaf} \end{cases}$

min. node  $\begin{cases} \text{Non-leaf} \\ \text{Leaf} \end{cases}$

CS 245 Notes 4 64

B+tree rules tree of order  $n$

- All leaves at same lowest level (balanced tree)
- Pointers in leaves point to records except for "sequence pointer"

CS 245 Notes 4 65

(3) Number of pointers/keys for B+tree

	Max ptrs	Max keys	Min ptrs → data	Min keys
Non-leaf (non-root)	$n+1$	$n$	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$
Leaf (non-root)	$n+1$	$n$	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
Root	$n+1$	$n$	1	1

CS 245 Notes 4 66

## Insert into B+tree

- (a) simple case
  - space available in leaf
- (b) leaf overflow
- (c) non-leaf overflow
- (d) new root

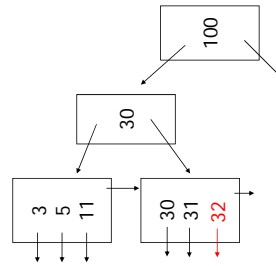
CS 245

Notes 4

67

(a) Insert key = 32

n=3



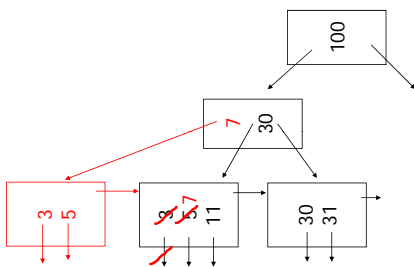
CS 245

Notes 4

68

(a) Insert key = 7

n=3



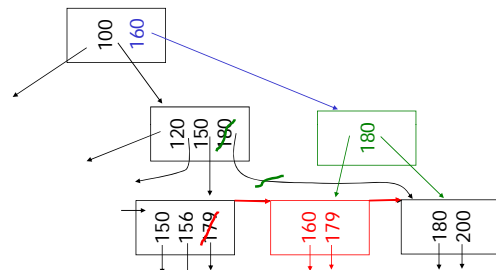
CS 245

Notes 4

69

(c) Insert key = 160

n=3



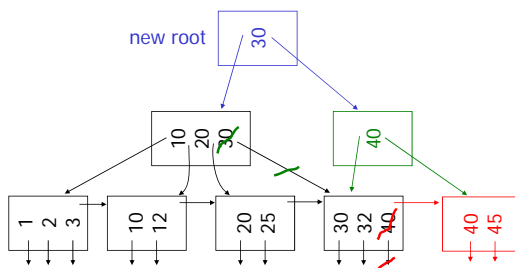
CS 245

Notes 4

70

(d) New root, insert 45

n=3



CS 245

Notes 4

71

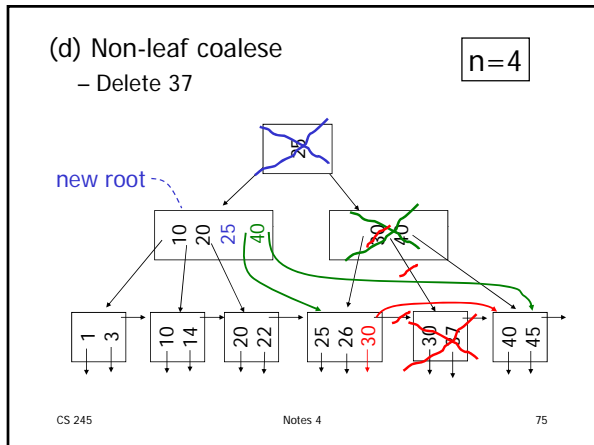
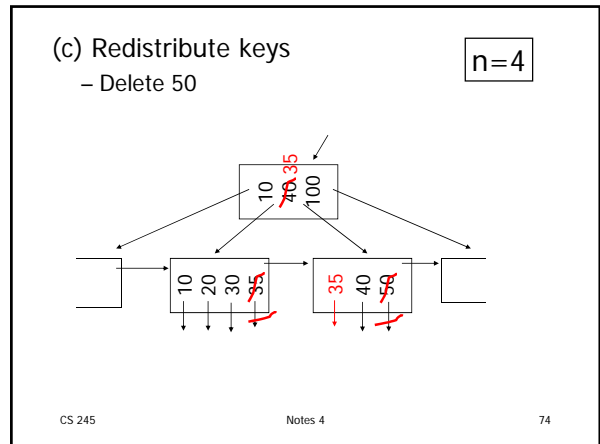
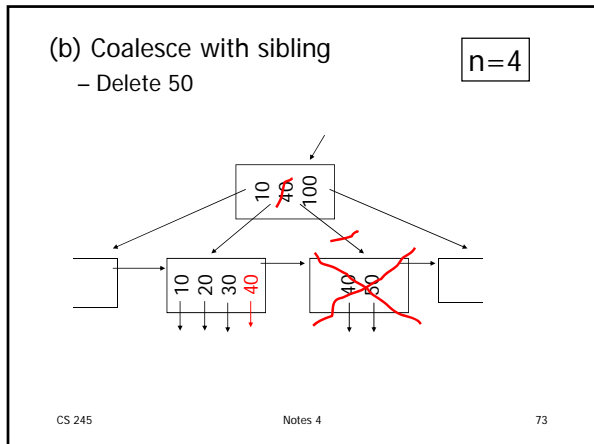
## Deletion from B+tree

- (a) Simple case - no example
- (b) Coalesce with neighbor (sibling)
- (c) Re-distribute keys
- (d) Cases (b) or (c) at non-leaf

CS 245

Notes 4

72



B+tree deletions in practice

- Often, coalescing is not implemented
  - Too hard and not worth it!

CS 245      Notes 4      76

Comparison: B-trees vs. static indexed sequential file

Ref #1: Held & Stonebraker  
"B-Trees Re-examined"  
CACM, Feb. 1978

CS 245      Notes 4      77

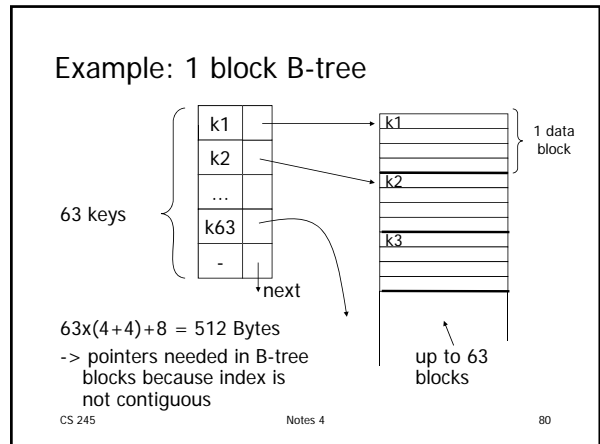
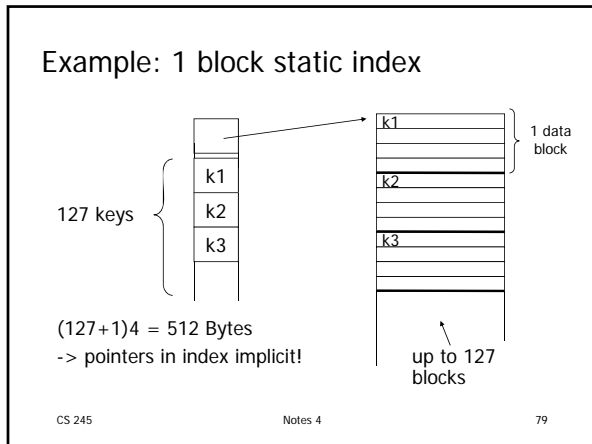
Ref # 1 claims:

- Concurrency control harder in B-Trees
- B-tree consumes more space

For their comparison:

- block = 512 bytes
- key = pointer = 4 bytes
- 4 data records per block

CS 245      Notes 4      78



### Size comparison Ref. #1

Static Index		B-tree	
# data blocks	height	# data blocks	height
2 -> 127	2	2 -> 63	2
128 -> 16,129	3	64 -> 3968	3
16,130 -> 2,048,383	4	3969 -> 250,047	4
		250,048 -> 15,752,961	5

CS 245 Notes 4 81

### Ref. #1 analysis claims

- For an 8,000 block file,
  - after 32,000 inserts
  - after 16,000 lookups
 => Static index saves enough accesses to allow for reorganization

Ref. #1 conclusion → Static index better!!

CS 245 Notes 4 82

### Ref #2: M. Stonebraker, "Retrospective on a database system," TODS, June 1980


Ref. #2 conclusion → B-trees better!!

CS 245 Notes 4 83

### Ref. #2 conclusion → B-trees better!!

- DBA does not know when to reorganize
- DBA does not know how full to load pages of new index

CS 245 Notes 4 84

Ref. #2 conclusion  B-trees better!!

- Buffering
  - B-tree: has fixed buffer requirements
  - Static index: must read several overflow blocks to be efficient (large & variable size buffers needed for this)

CS 245

Notes 4

85

- Speaking of buffering...  
Is LRU a good policy for B+tree buffers?

→ Of course not!  
→ Should try to keep root in memory at all times

(and perhaps some nodes from second level)

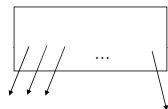
CS 245

Notes 4

86

### Interesting problem:

For B+tree, how large should  $n$  be?



$n$  is number of keys / node

CS 245

Notes 4

87

### Sample assumptions:

- (1) Time to read node from disk is  $(S+Tn)$  msec.
- (2) Once block in memory, use binary search to locate key:  
 $(a + b \text{ LOG}_2 n)$  msec.  
For some constants  $a, b$ : Assume  $a \ll S$
- (3) Assume B+tree is full, i.e.,  
# nodes to examine is  $\text{LOG}_n N$   
where  $N = \#$  records

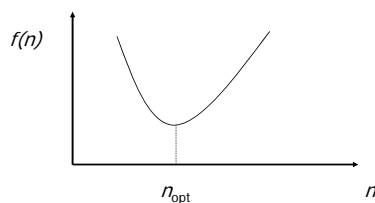
CS 245

Notes 4

88

Can get:

$f(n) =$  time to find a record



CS 245

Notes 4

89

Find  $n_{\text{opt}}$  by  $f'(n) = 0$

Answer is  $n_{\text{opt}} =$  "few hundred"  
(see homework for details)

What happens to  $n_{\text{opt}}$  as

- Disk gets faster?
- CPU get faster?

CS 245

Notes 4

90

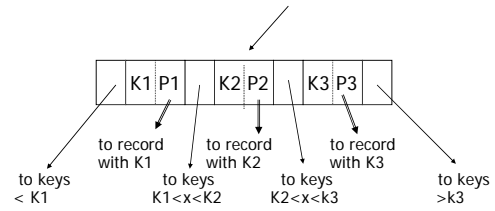
## Variation on B+tree: B-tree (no +)

- Idea:
  - Avoid duplicate keys
  - Have record pointers in non-leaf nodes

CS 245

Notes 4

91



CS 245

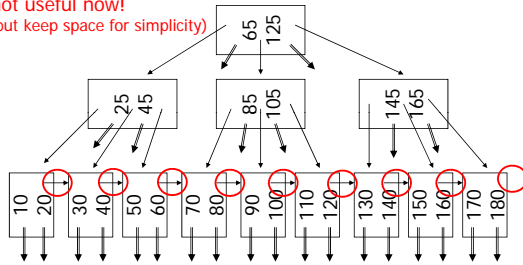
Notes 4

92

## B-tree example

$n=2$

- sequence pointers  
not useful now!  
(but keep space for simplicity)



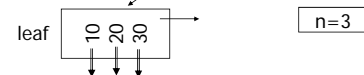
CS 245

Notes 4

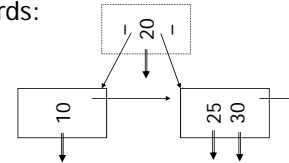
93

## Note on inserts

- Say we insert record with key = 25



- Afterwards:



CS 245

Notes 4

94

## So, for B-trees:

	MAX			MIN		
	Tree Ptrs	Rec Ptrs	Keys	Tree Ptrs	Rec Ptrs	Keys
Non-leaf non-root	$n+1$	$n$	$n$	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$	$\lceil (n+1)/2 \rceil - 1$
Leaf non-root	1	$n$	$n$	1	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
Root non-leaf	$n+1$	$n$	$n$	2	1	1
Root Leaf	1	$n$	$n$	1	1	1

CS 245

Notes 4

95

## Tradeoffs:

- ⊗ B-trees have faster lookup than B+ trees
- ⊗ in B-tree, non-leaf & leaf different sizes
- ⊗ in B-tree, deletion more complicated

➔ B+ trees preferred!

CS 245

Notes 4

96

But note:

- If blocks are fixed size  
(due to disk and buffering restrictions)

Then lookup for B+tree is actually better!!

Example:

- Pointers 4 bytes
- Keys 4 bytes
- Blocks 100 bytes (just example)
- Look at full 2 level tree

B-tree:

Root has 8 keys + 8 record pointers  
+ 9 son pointers  
=  $8 \times 4 + 8 \times 4 + 9 \times 4 = 100$  bytes

Each of 9 sons: 12 rec. pointers (+12 keys)  
=  $12 \times (4+4) + 4 = 100$  bytes

2-level B-tree, Max # records =  
 $12 \times 9 + 8 = 116$

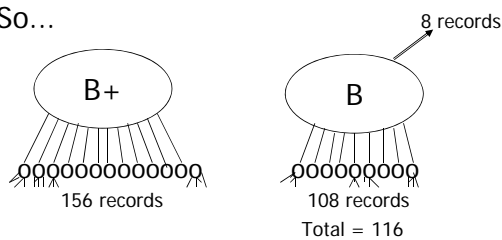
B+tree:

Root has 12 keys + 13 son pointers  
=  $12 \times 4 + 13 \times 4 = 100$  bytes

Each of 13 sons: 12 rec. ptrs (+12 keys)  
=  $12 \times (4 + 4) + 4 = 100$  bytes

2-level B+tree, Max # records  
=  $13 \times 12 = 156$

So...



• Conclusion:

- For fixed block size,
- B+ tree is better because it is bushier

**An Interesting Problem...**

- What is a good index structure when:
  - records tend to be inserted with keys that are larger than existing values? (e.g., banking records with growing data/time)
  - we want to remove older data

## One Solution: Multiple Indexes

- Example: I1, I2

day	days indexed	
	I1	I2
10	1,2,3,4,5	6,7,8,9,10
11	11,2,3,4,5	6,7,8,9,10
12	11,12,3,4,5	6,7,8,9,10
13	11,12,13,4,5	6,7,8,9,10

- advantage: deletions/insertions from smaller index
- disadvantage: query multiple indexes

CS 245

Notes 4

103

## Another Solution (Wave Indexes)

day	I1	I2	I3	I4
10	1,2,3	4,5,6	7,8,9	10
11	1,2,3	4,5,6	7,8,9	10,11
12	1,2,3	4,5,6	7,8,9	10,11, 12
13	13	4,5,6	7,8,9	10,11, 12
14	13,14	4,5,6	7,8,9	10,11, 12
15	13,14,15	4,5,6	7,8,9	10,11, 12
16	13,14,15	16	7,8,9	10,11, 12

- advantage: no deletions
- disadvantage: approximate windows

CS 245

Notes 4

104

## Outline/summary

- Conventional Indexes
  - Sparse vs. dense
  - Primary vs. secondary
- B trees
  - B+trees vs. B-trees
  - B+trees vs. indexed sequential
- Hashing schemes --> Next

CS 245

Notes 4

105