## CS422 Principles of Database Systems
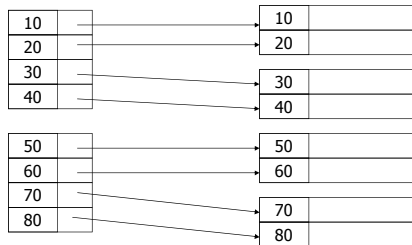Indexes

Chengyu Sun
California State University, Los Angeles

## Indexes

◈ Auxiliary structures that speed up operations that are not supported efficiently by the basic file organization

## A Simple Index Example

| | |
|---|---|
| 10 | |
| 20 | |
| 30 | |
| 40 | |

| | |
|---|---|
| 50 | |
| 60 | |
| 70 | |
| 80 | |

| | |
|---|---|
| 10 | |
| 20 | |

| | |
|---|---|
| 30 | |
| 40 | |

| | |
|---|---|
| 50 | |
| 60 | |

| | |
|---|---|
| 70 | |
| 80 | |

Index blocks                    Data blocks

## About Indexes

◈ Index entry
- <key, rid>
- <key, list of rid>
- Data record

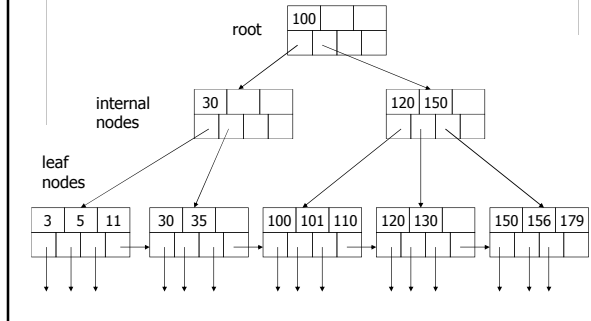◈ The majority of database indexes are designed to reduce *disk access*

## Organization of Index Entries

◈ Tree-structured
- B-tree, R-tree, Quad-tree, kd-tree, …

◈ Hash-based
- Static, dynamic

◈ Other
- Bitmap, VA-file, …

## From BST to BBST to B

◈ Binary Search Tree
- Worst case??

◈ Balance Binary Search Tree
- E.g. AVL, Red-Black

◈ B-tree
- Why not use BBST in databases??

## B-tree (B+-tree) Example



root 100

internal nodes: 30 | 120 150

leaf nodes:
3 5 11 | 30 35 | 100 101 110 | 120 130 | 150 156 179

## B-tree Properties

- ◈ Each node occupies one block
- ◈ Order n
  - ▪ n keys, n+1 pointers
- ◈ Nodes (except root) must be at least half full
  - ▪ Internal node: $\lceil (n+1)/2 \rceil$ pointers
  - ▪ Leaf node: $\lfloor (n+1)/2 \rfloor$ pointers
- ◈ All leaf nodes are on the same level

## B-tree Operations

- ◈ Search
- ◈ Insert
- ◈ Delete

## B-tree Insert

- ◈ Find the appropriate leaf
- ◈ Insert into the leaf
  - ▪ there's room → we're done
  - ▪ no room
    - ◆ split leaf node into two
    - ◆ insert a new <key,pointer> pair into leaf's parent node
- ◈ *Recursively apply previous step if necessary*
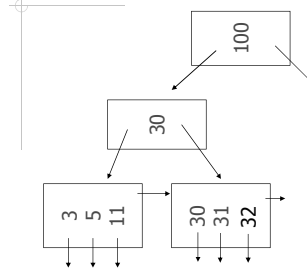  - ▪ A split of current ROOT leads to a new ROOT

## B-tree Insert Examples

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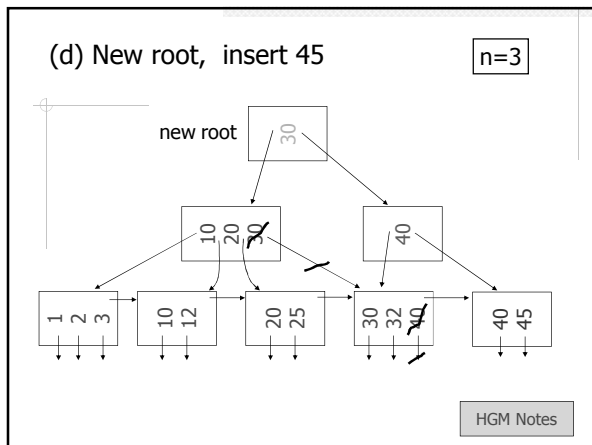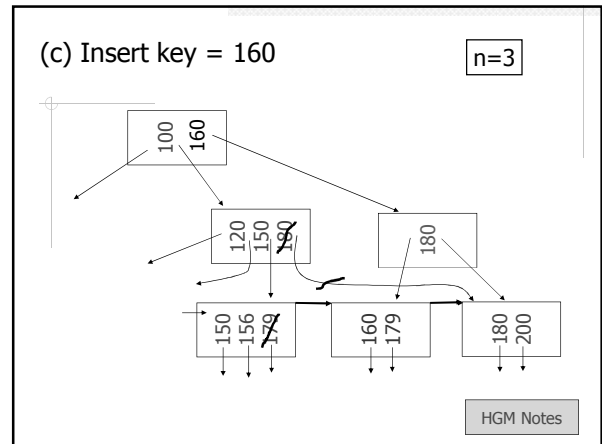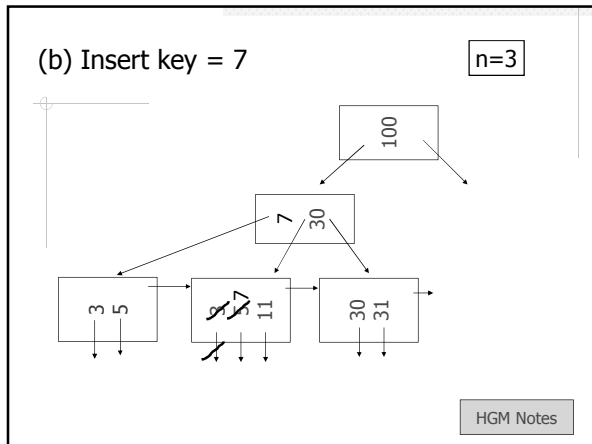- ◈ (b) leaf overflow
- ◈ (c) non-leaf overflow
- ◈ (d) new root

## (a) Insert key = 32

n=3



100

30

3 5 11 | 30 31 32

## (b) Insert key = 7  n=3

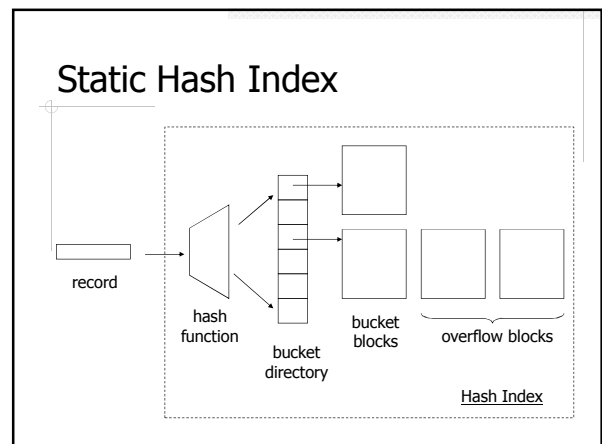## (c) Insert key = 160  n=3

## (d) New root, insert 45  n=3



new root

# B-tree Delete

- ◈ Find the appropriate leaf
- ◈ Delete from the leaf
  - still at least half full → we're done
  - below half full – coalescing
    - ◆ borrow a <key,pointer> from one sibling node, *or*
    - ◆ merge with a sibling node, and delete from a parent node
- ◈ Recursively apply previous step if necessary

# B-tree Delete in Practice

- ◈ Coalescing is usually not implemented because it's too hard and not worth it

# Static Hash Index



record

hash function

bucket directory

bucket blocks

overflow blocks

Hash Index

3

## Hash Function

◈ A commonly used hash function: `K%B`
- `K` is the key value
- `B` is the number of buckets

## Static Hash Index Example …

◈ `4` buckets
◈ Hash function: `key%4`
◈ `2` index entries per bucket block

## … Static Hash Index Example
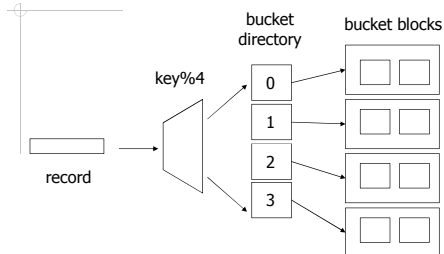


◈ Insert the records with the following keys: `4, 3, 7, 17, 22, 10, 25, 33`

## Dynamic Hashing

◈ Problem of static hashing??
◈ Dynamic hashing
- Extendable Hash Index

## Extendable Hash Index …

◈ Maximum $2^M$ buckets
- `M` is maximum depth of index
◈ Multiple buckets can share the same block
◈ Inserting a new entry to a block that is already full would cause the block to split

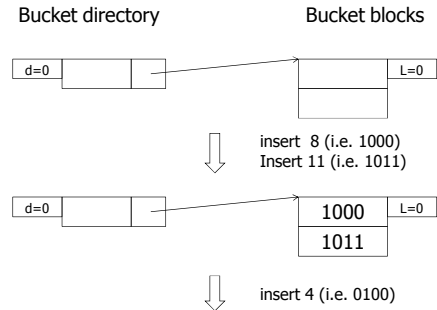## … Extendable Hash Index

◈ Each block has a local depth `L`, which means that the hash values of the records in the block has the same rightmost `L` bit
◈ The bucket directory keeps a global depth `d`, which is the highest local depth
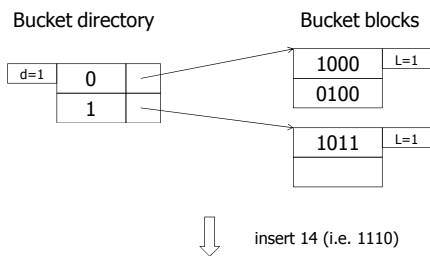
## Extendable Hash Index Example

- M=4 (i.e. could have at most 16 buckets)
- Hash function: key%2$^4$
- 2 index entries per block
- Insert 8, 11, 4, 14

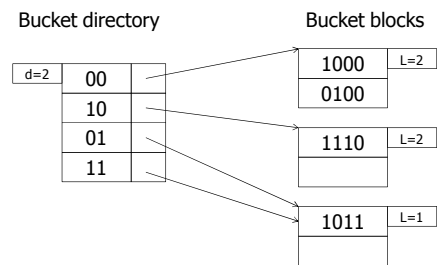## Extendable Hashing (I)

Bucket directory          Bucket blocks

d=0                                        L=0

insert 8 (i.e. 1000)
Insert 11 (i.e. 1011)

d=0                          1000    L=0
                             1011

insert 4 (i.e. 0100)

## Extendable Hashing (II)

Bucket directory          Bucket blocks

d=1    0                     1000    L=1
       1                     0100

                             1011    L=1

insert 14 (i.e. 1110)

## Extendable Hashing (III)

Bucket directory          Bucket blocks

d=2    00                    1000    L=2
       10                    0100
       01                    1110    L=2
       11
                             1011    L=1

## Readings

- Textbook Chapter 21.1 – 21.4