CS411
Database Systems

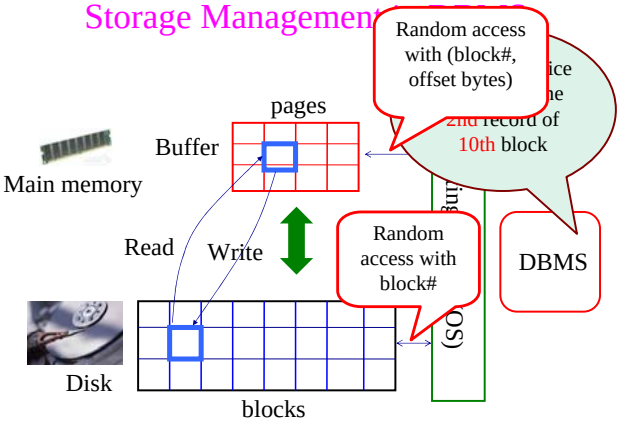
10: Indexing-1

Kazuhiro Minami

Storage Representation: Basic questions

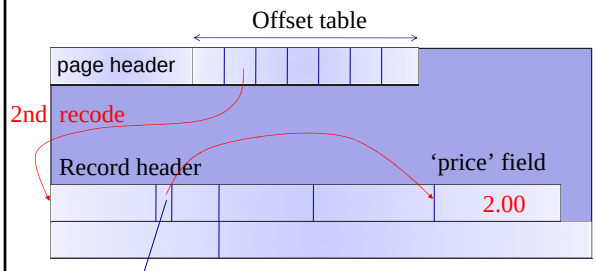
- What is a “block”?
- What’s the metrics for evaluating algorithms in DBMS?
- What’s the purpose of main memory buffer?
- Why do we need a *record* header? What kind of information is included?
- Why do we need a *block* header? What kind of information is included?
- What’s the major difference between a block storing fixed-length records and that storing variable-length records?
- What is a “pointer”?

Storage Management

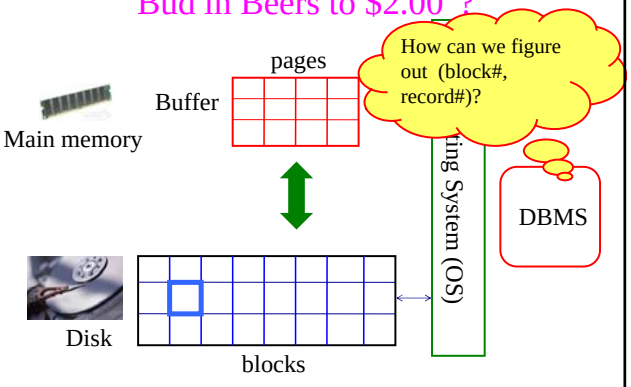


Accessing a Field of a Record Within a Block

Address (block#, record#) = (10, 2)

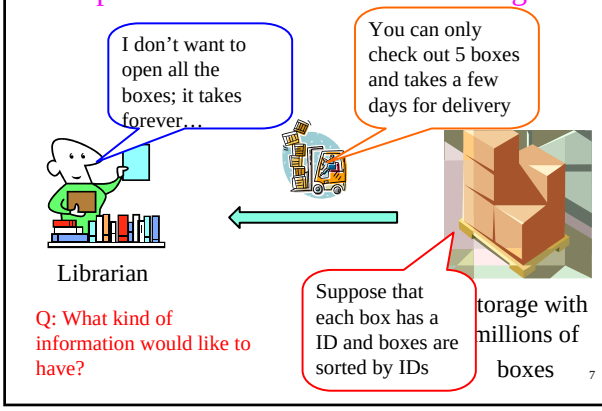


What if a user say "Update the price of Bud in Beers to $2.00"?



Indexing

How to find boxes containing "History of Japan" Volume 1 – 100 from Storage?



Q: What kind of information would like to have?

Probably, what you want is a table (i.e., index)

Book title	Box ID
History of Japan vol. 1	925
History of Japan vol. 2	925
History of Japan vol. 3	926
History of Japan vol. 4	928
History of Japan vol. 5	928
History of Japan vol. 6	928
History of Japan vol. 7	1001
History of Japan vol. 8	1002
History of Japan vol. 9	1002
History of Japan vol. 10	1003

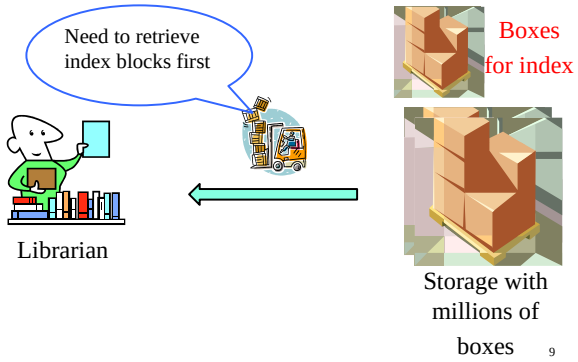
Q: do you think that we solved the problem?

Q: how do you find the entry for "History of Japan" in this table?

Q: What if this table is so big and is stored in boxes in the storage?

This is the exact problem we need to address in DBMS

How to first find boxes containing Index for "History of Japan" from Storage?

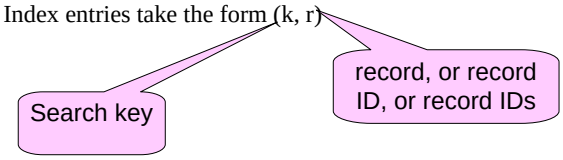


Indexes are used to speed up selections on particular attributes

Search key field(s) :

- The attribute(s) that you want to look up tuples by
- any subset of the fields of a relation
- *Search key* is not the same as *key* (minimal set of fields that uniquely identify a record in a relation).

Index entries take the form (k, r)

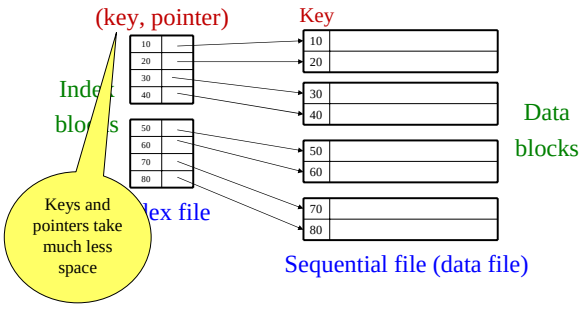


There are several different kinds of indexes used in DBMSs

- Clustered/unclustered
  - Clustered = records sorted in the (search) key order
  - Unclustered = no
- Dense/sparse
  - Dense = each record has an entry in the index
  - Sparse = only some records have
- Primary/secondary
  - Primary = on the primary key
  - Secondary = on any key

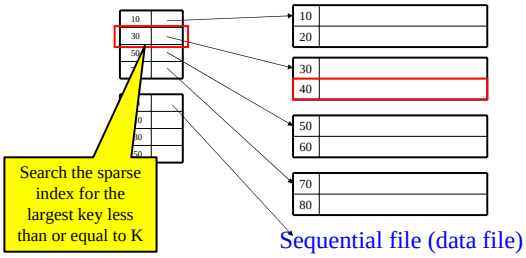
Dense Indexes on a Sequential Data File

- (key, pointer) pair for every record
- File is sorted by the primary key



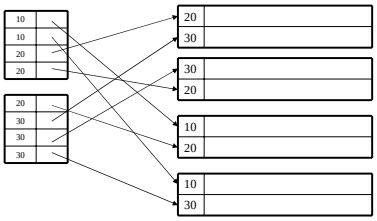
Sparse Indexes on a Sequential Data File

- *Sparse* index: one key per data block
- Use less space, but takes more time for search
- Only work with sequential files

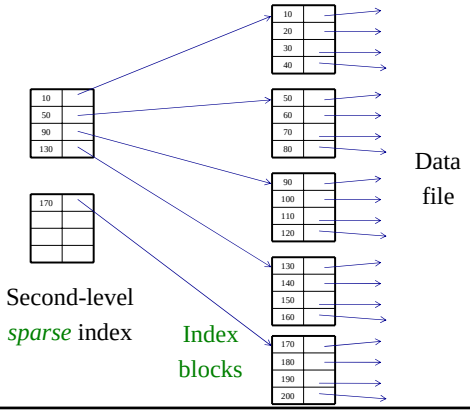


Unclustered Indexes

- To index other attributes than primary key
- Always dense (why ?)



How to find an index entry efficiently?



B-Trees

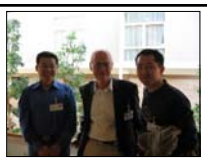
B-Trees

- Automatically maintain as many level of index as is appropriate for the size of the file being indexed
- Organize its blocks into a tree
  - Balanced: all paths from the root to a leaf have the same length
- Manage the space on the blocks they use so that every block is between half used and completely full.
  - No overflow blocks are needed

B-Trees: Balanced Trees

- Intuition:
  - Give up on sequentiality of index
  - Try to get "balance" by dynamic reorganization
- B+trees:
  - Textbook refers to B+trees (a popular variant) as B-trees (as most people do)
  - Distinction will be clear later (ok to confuse now)

UIUC (Alumni) Contribution!



Prof. Rudolf Bayer

Rudolf Bayer studied Mathematics in Munich and at the University of Illinois, where he received his Ph.D. in 1966. After working at Boeing Research Labs he became an Associate Professor at Purdue University. He is a Professor of Informatics at the Technische Universität München since 1972 and … …

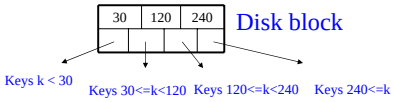
The 2001 SIGMOD Innovations Award goes to Prof. Rudolf Bayer of the Technical University of Munich, for his invention of the B-Tree (with Edward M. McCreight), of B-Tree prefix compression, and of lock coupling (a.k.a. crabbing) for concurrent access to B-Trees (with Mario Schkolnick). All of these techniques are widely used in commercial database products. …..…

The Original Publication

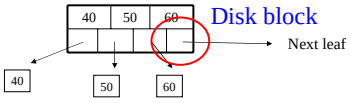
Rudolf Bayer, Edward M. McCreight: Organization and Maintenance of Large Ordered Indices. Acta Informatica 1: 173-189(1972)

B-Trees Basics

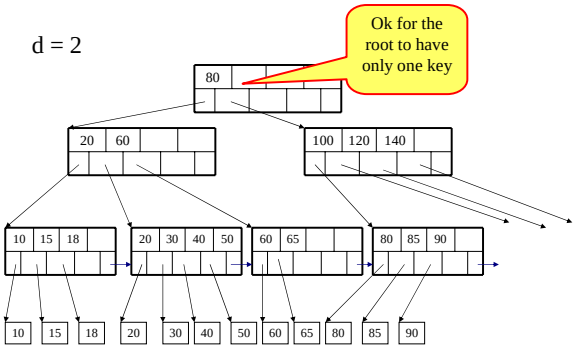
- Parameter d = the *degree* (In the textbook, d/2 is the parameter *n*)
- Each node has k keys and k+1 pointers where d <= k <= 2d keys (except root)



- Each leaf has k keys where d <= k <= 2d:



B-Tree Example



B-Tree Design

- How large d ?
- Example:
  - Key size = 4 bytes
  - Pointer size = 8 bytes
  - Block size = 4096 byes
- 2d * 4 + (2d+1) * 8 <= 4096
- d = 170

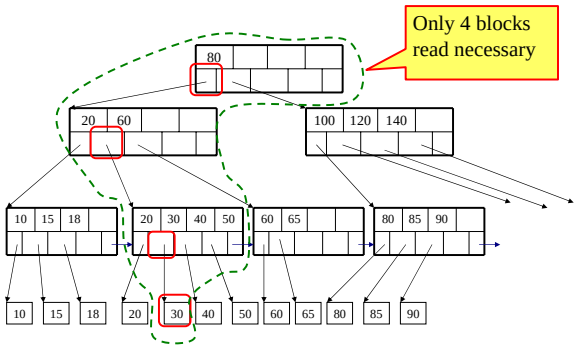
Searching a B-Tree

- Exact key values:
  - Start at the root
  - Proceed down, to the leaf
- Range queries:
  - As above
  - Then sequential traversal of the leaf nodes

Select name
From people
Where age = 25

Select name
From people
Where 20 <= age
and age <= 30

Example: Find a record with key 30



B-Trees in Practice

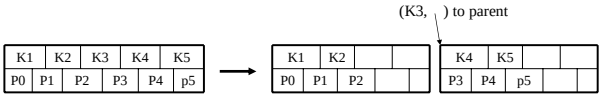
- Typical order: 100. Typical fill-factor: 67%.
  - average fanout = 133
- Typical capacities:
  - Height 4: 133^4 = 312,900,700 records
  - Height 3: 133^3 = 2,352,637 records
- Can often hold top levels in buffer pool:
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBytes

Big enough for most applications

Insertion in a B-Tree

Insert (K, P)

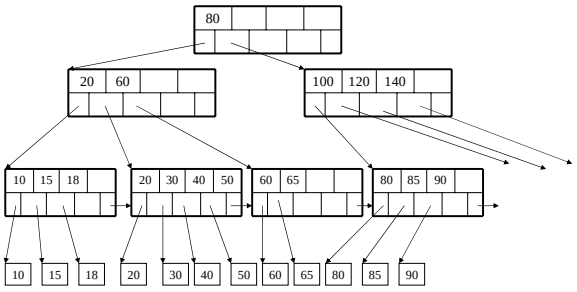
- Find leaf where K belongs, insert
- If no overflow (2d keys or less), halt
- If overflow (2d+1 keys), split node, insert in parent:



- If leaf, keep K3 too in right node

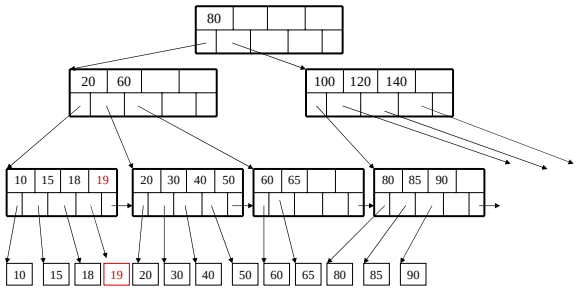
Insertion in a B-Tree

Insert K=19



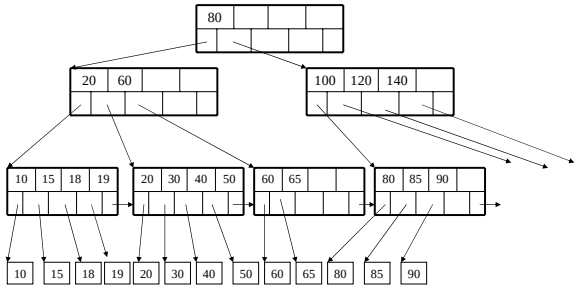
Insertion in a B-Tree

After insertion



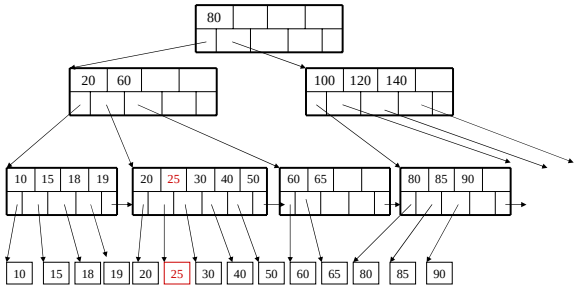
Insertion in a B-Tree

Now insert 25



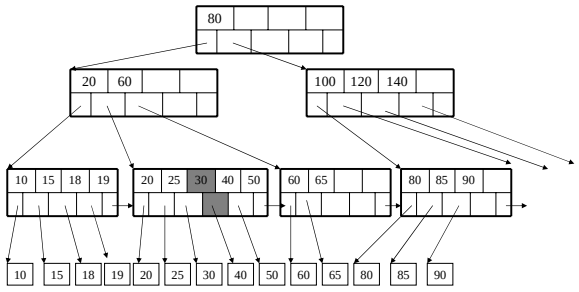
Insertion in a B-Tree

After insertion



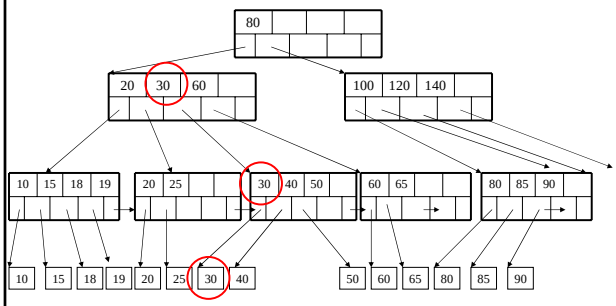
Insertion in a B-Tree

But now have to split !



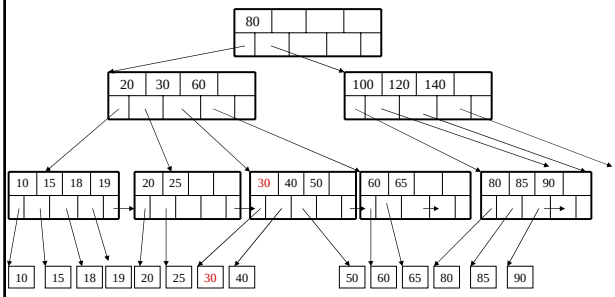
Insertion in a B-Tree

After the split



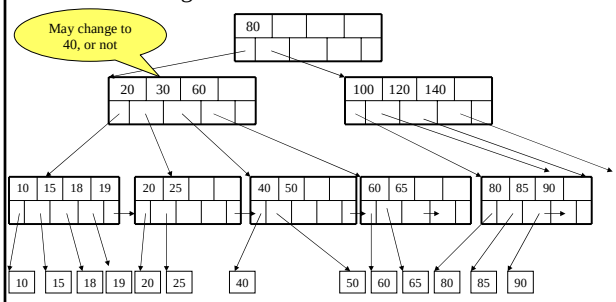
Deletion from a B-Tree

Delete 30



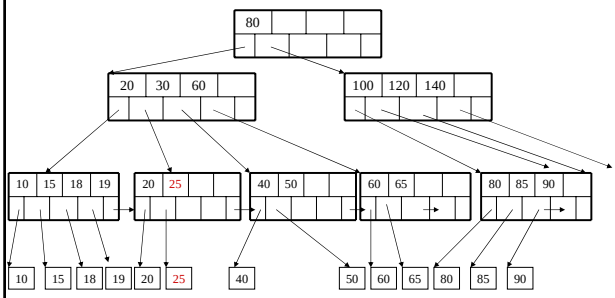
Deletion from a B-Tree

After deleting 30



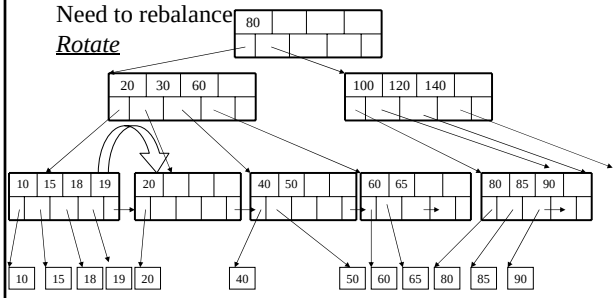
Deletion from a B-Tree

Now delete 25



Deletion from a B-Tree

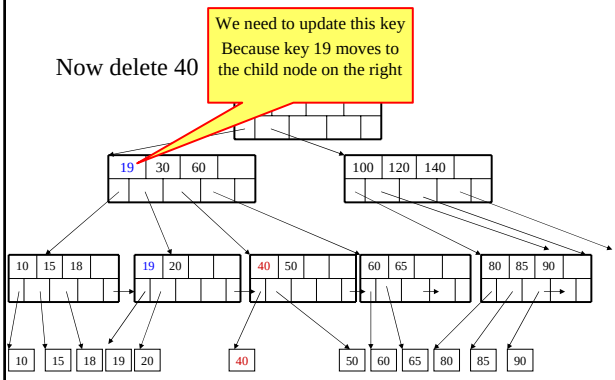
After deleting 25
Need to rebalance
Rotate



Deletion from a B-Tree

Now delete 40

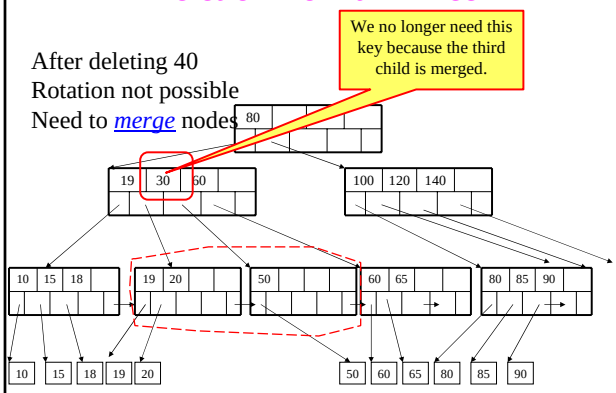
We need to update this key
Because key 19 moves to
the child node on the right



Deletion from a B-Tree

After deleting 40
Rotation not possible
Need to *merge* nodes

We no longer need this
key because the third
child is merged.



Deletion from a B-Tree

Final tree