CSE 306/506 Operating Systems
File Management

YoungMin Kwon

Filesystem

- Filesystems allow users to create data collections, called files

- Files have following properties:
  - Long-term existence
  - Sharable between processes
  - Files can have an internal structure and be organized into a hierarchical structure

File Structure

- Field
  - A basic element of data
  - E.g., employee’s name, a date, a sensor reading…
- Record
  - A collection of related fields that can be treated as a unit
  - E.g., an employee record with name, id, hiring date, …

File Structure

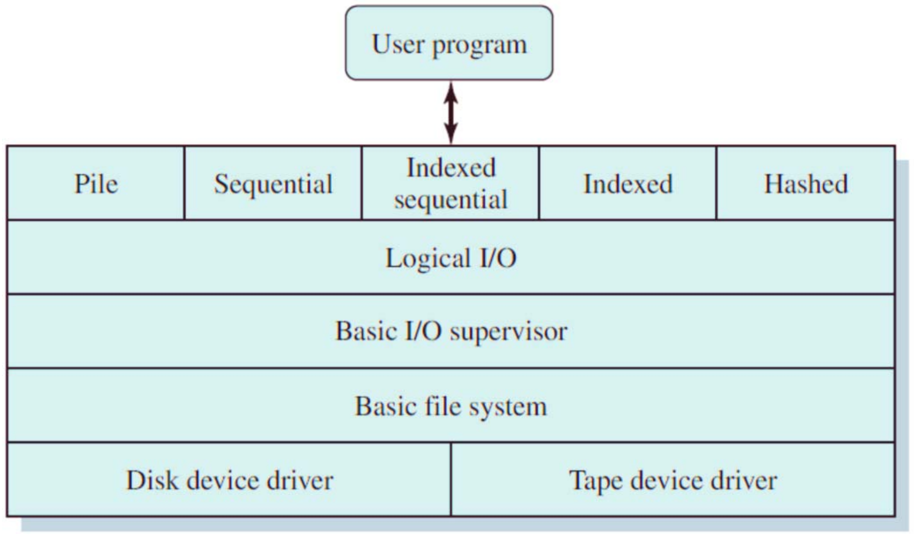
- File
  - A collection of similar records
  - Access control usually apply at the file level
- Database
  - A collection of related data
  - Designed for use by a number of different applications
  - May consists of one or more types of files

File Management System

- File management system
  - A set of software that provides services to users and applications in the use of files

- Requirements
  - Each user is able to create, delete, read, write, and modify files
  - Each user may have controlled access to other user’s files
  - Each user is able to move data between files
  - Each user can backup and recover the user’s files
  - Each user can access files by name rather than by a numeric id

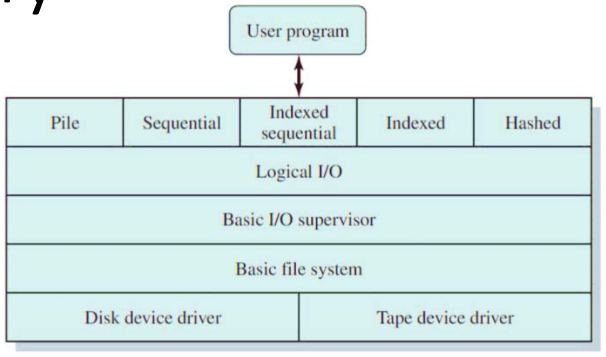
File System Architecture



- Device drivers
  - Communicate directly with devices
  - Responsible for staring I/O operations and handling the completion of an I/O request

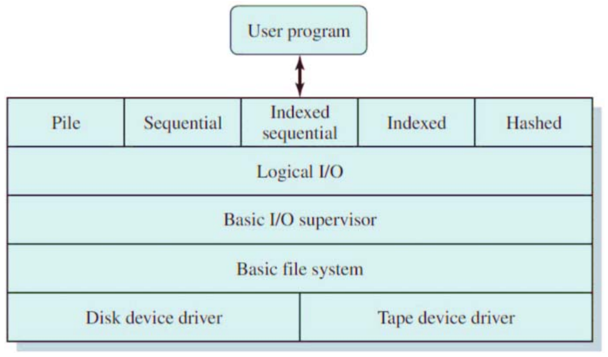
File System Architecture

- Basic file system
  - Primary interface with the environment outside of the computer system
  - Placement of blocks on the secondary storage device, buffering of the blocks in main memory
  - Does not understand the contents or the structure of data



- Basic I/O supervisor
  - Responsible for all file I/O initiation and termination
  - Selecting device, scheduling access, allocating buffer

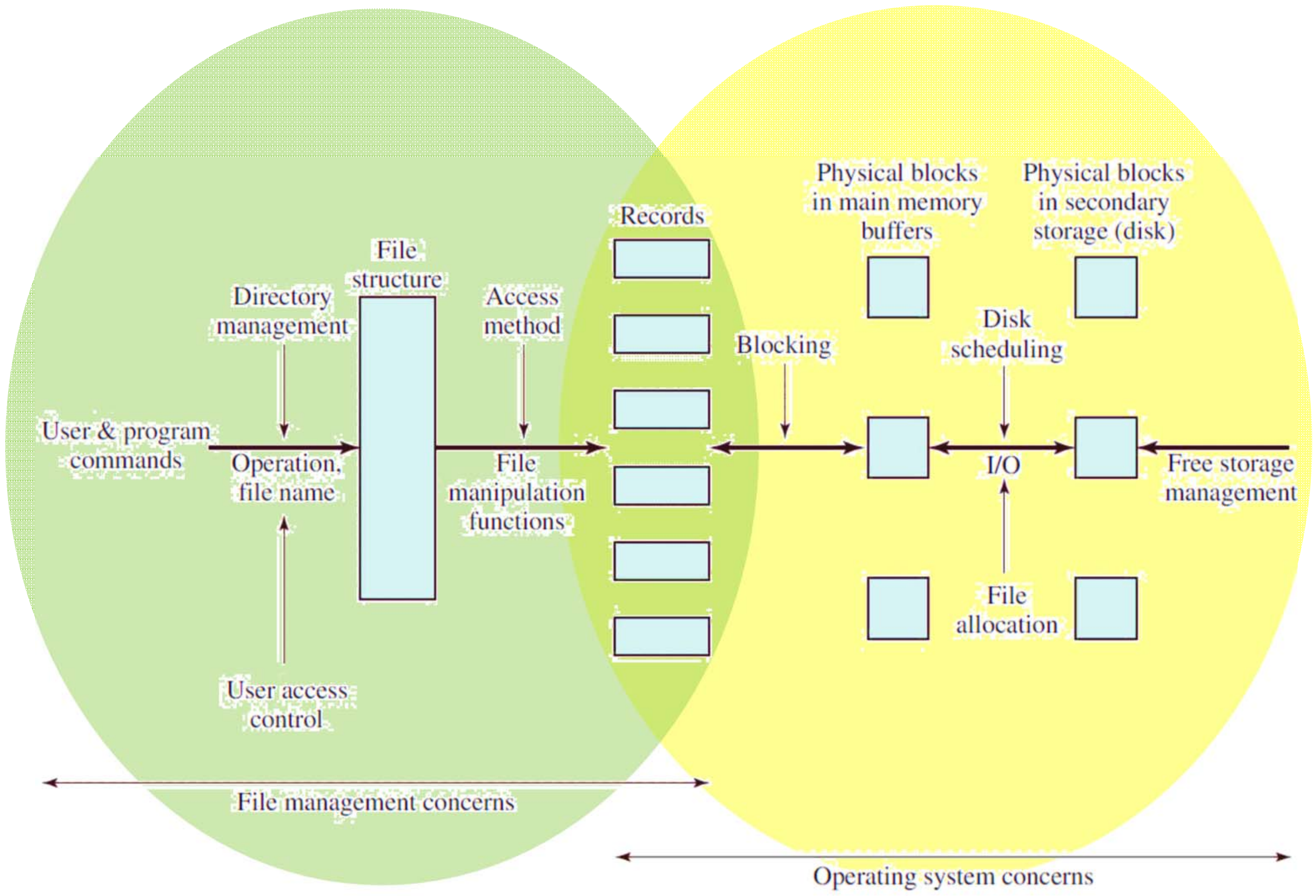
File System Architecture



- Logical I/O
  - Enable users and applications to access records
  - Whereas the basic file system deals with blocks, logical I/O module deals with records

- Access Method
  - Provides standard interfaces between applications and file systems

Elements of File Management



File Organization and Access

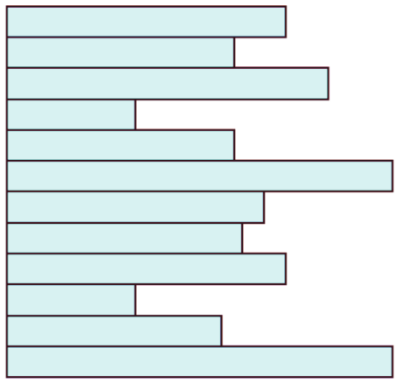
- File organization
  - Logical structuring of the records

- Criteria to consider
  - Short access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability

File Organization and Access

- Five fundamental organizations
  - Pile
  - Sequential file
  - Indexed sequential file
  - Indexed file
  - Direct or hashed file

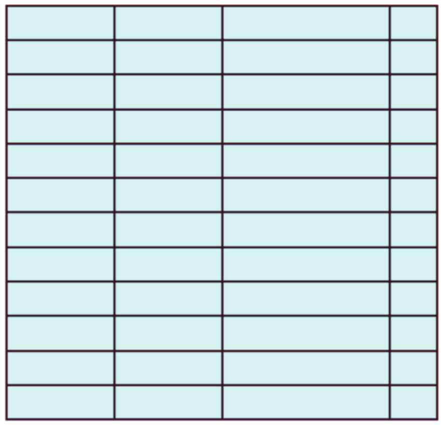
The Pile



- Record organization
  - Collected in the order they arrive
  - Records may have different fields in different orders
  - Variable length records

- Record access
  - Exhaustive search

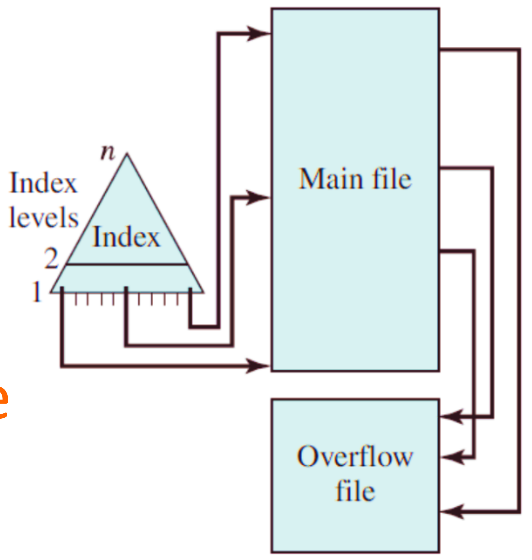
The Sequential File



- Record organization
  - All records are of the same length, same number of fixed length fields, in particular order
  - There is a key field that identifies the record
    - Records are sorted by the key
- Record access
  - Sequential scan
  - For a new record, append it to a pile type log file and merge it to the master file later

Indexed Sequential File

- Record organization
  - Records are a sorted sequence based on the key
  - Index file has (key, file location) pairs
  - New records are added to overflow file with a link from the main file



- Record access
  - From the index file, find a nearby record location and do a sequential search from the record
  - Multilevel index file
    - Lower level indexes are like the sequential file
    - Higher level indexes are indexes into lower level

Indexed Sequential File

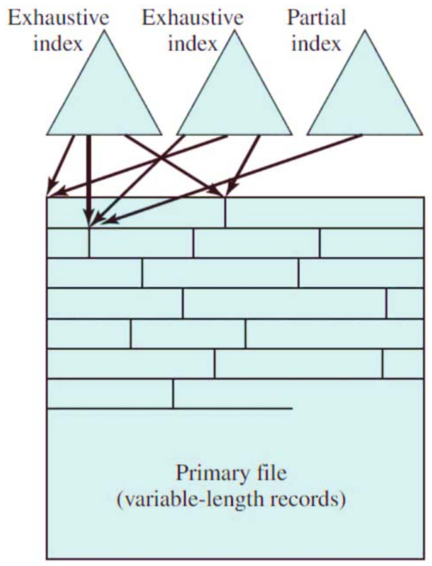
- Example
  - A sequential file with 1 million records
    - On average 500,000 records are searched
  - An indexed sequential file with 1000 indexes and 1 million records
    - On average 500 indexes and 500 records are searched (total 1,000 searches)
  - A 2-level indexed sequential file with 100 high-level indexes, 10,000 low-level indexes, and 1 million records
    - On average 50 high-level indexes, 50 low-level indexes, and 50 records are searched (total 150 searches)

Direct (or Hashed) File

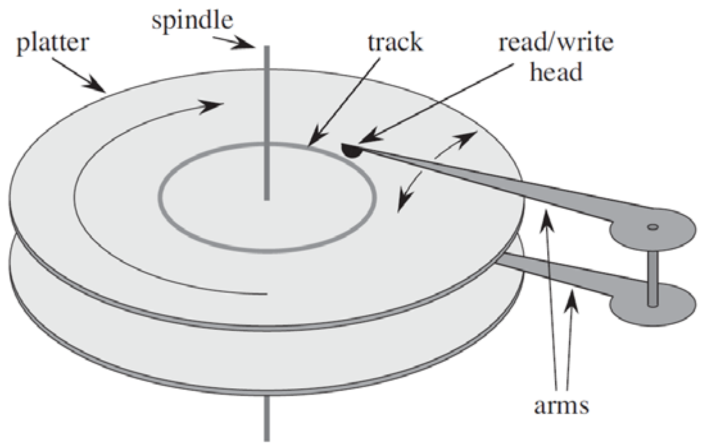
- Hashing on the key value
  - Store data at the hash value location
  - Retrieve data from the hash value location

Indexed File

- Indexed sequential file
  - Inefficient when multiple keys are used
- Record organization
  - Variable-length records are appended to the primary file
  - Multiple index files have its own (key, file location) pairs
- Record access
  - Find a file location from the index file for a certain key



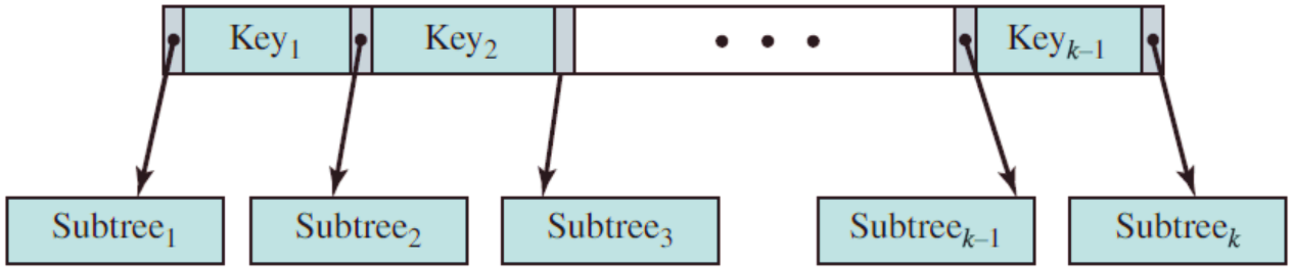
B-Tree



- For a large file or database
  - For efficiency, a multi-level index file can be used
  - A balanced structure is preferred
    - To avoid uneven access times
  - Need to reduce the access to the secondary memory

- B-tree
  - Balanced tree with large branching factor (height is small)
  - Each node holds large amount of indexes
  - Has efficient searching, adding, and deleting algorithms

B-tree

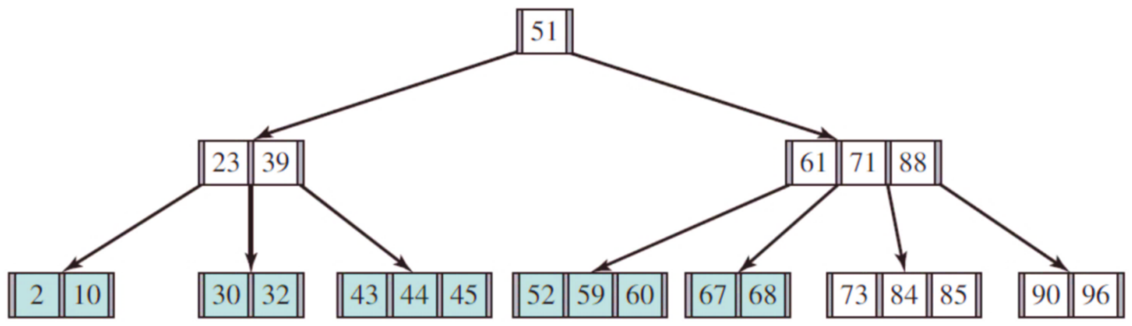


A B-tree Node with *k* Children

- Definition
  - The tree consists of nodes and leaves
  - Each node contains at least one key and more than one pointers to child nodes or leaves
  - Each node can have the same maximum number of keys
  - Keys in a node are stored in non-decreasing order

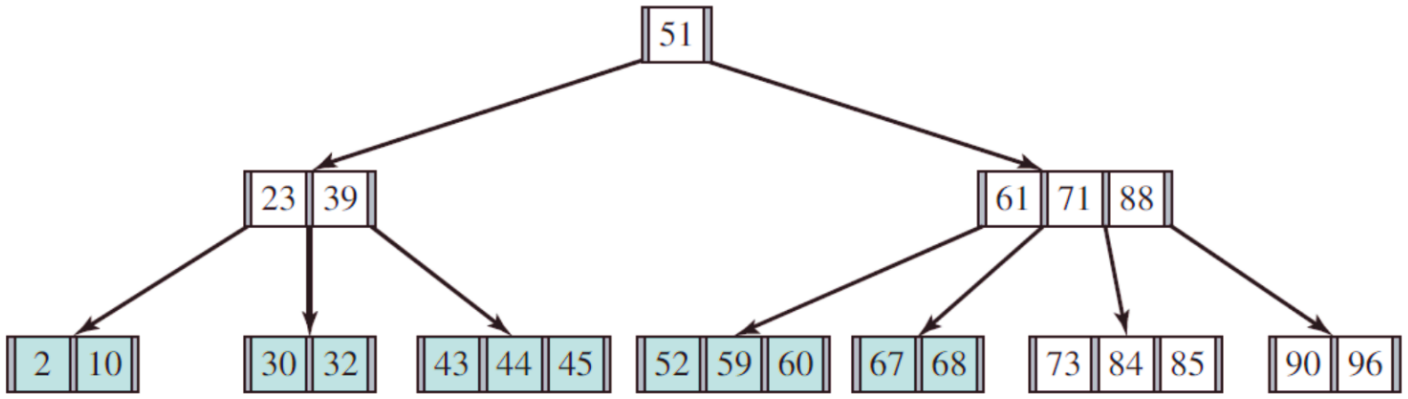
B-Tree

- B-tree with minimum degree d
  - Every node has at most 2d – 1 keys and 2d children
  - Each node (except for the root) has at least d – 1 keys and d pointers
  - The root node has at least 1 key and 2 children
  - All leaves appear on the same level
  - A non-leaf node with k pointers contain k – 1 keys



B-Tree

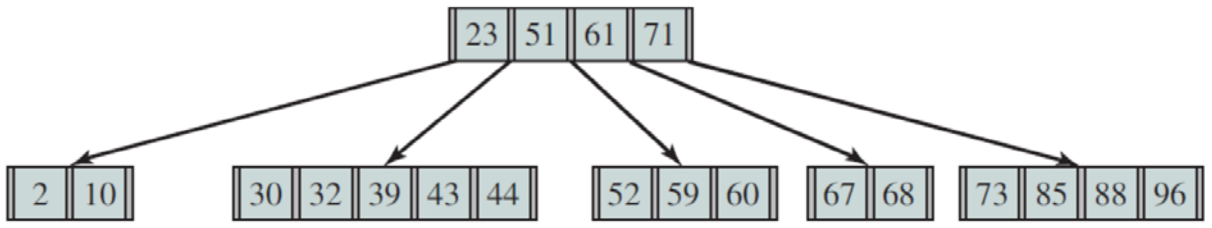
- Search for a key
  - If the key is found in the node, the search is done
  - If the key is smaller than the smallest key of the node, follow the leftmost pointer
  - If the key is larger than the largest key of the node, follow the rightmost pointer
  - If the key is in between two keys of the node, follow the pointer in between the keys



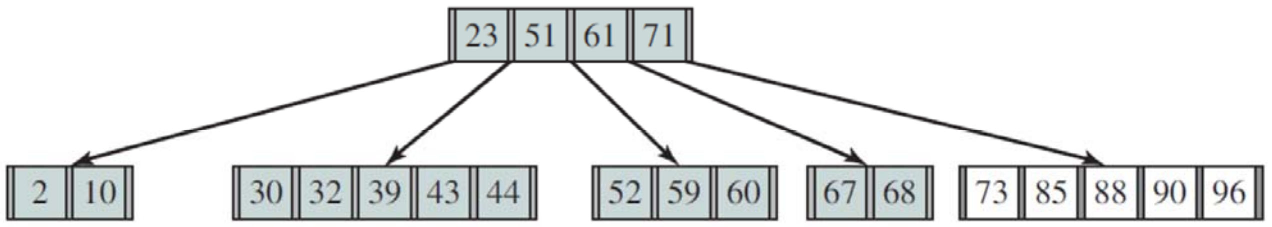
B-Trees

- Inserting a new key
  - Search the tree for the new key; if the key is not in the tree, you are at the lowest level
  - If the node has fewer than 2d – 1 keys, insert the key to the node at the proper position
  - Otherwise split the node
    - Move the median key to the parent node
    - Split the node into two: LHS node has keys less than the median key and the RHS node has the other keys
    - If the key is smaller than the median key, insert it to the left node; otherwise insert it to the right node
  - The median key moved to the parent node may split its parent node again
    - Keep splitting the parent node until a non-full node is encountered or the root node is split

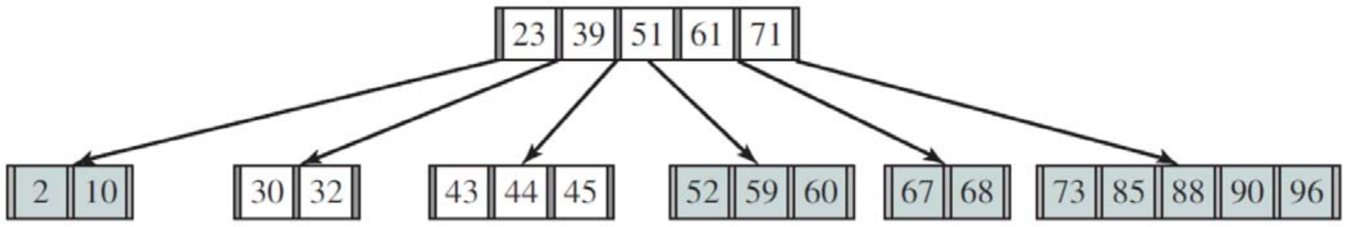
B-tree (d=3)



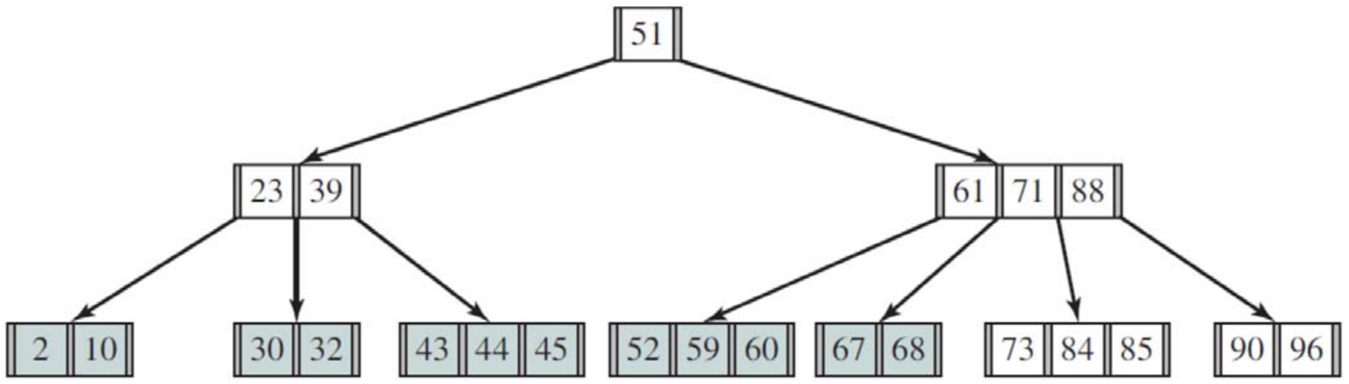
90 is added



45 is added

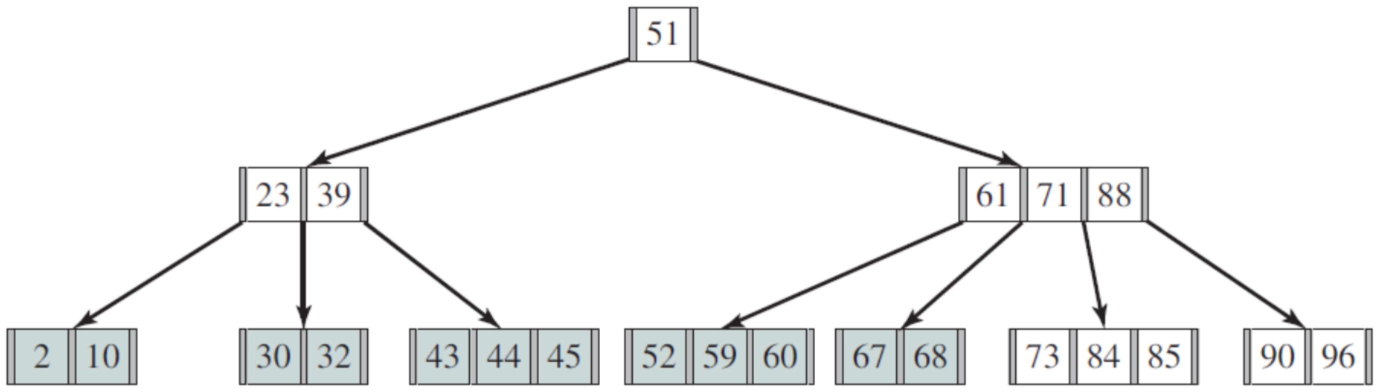


84 is added

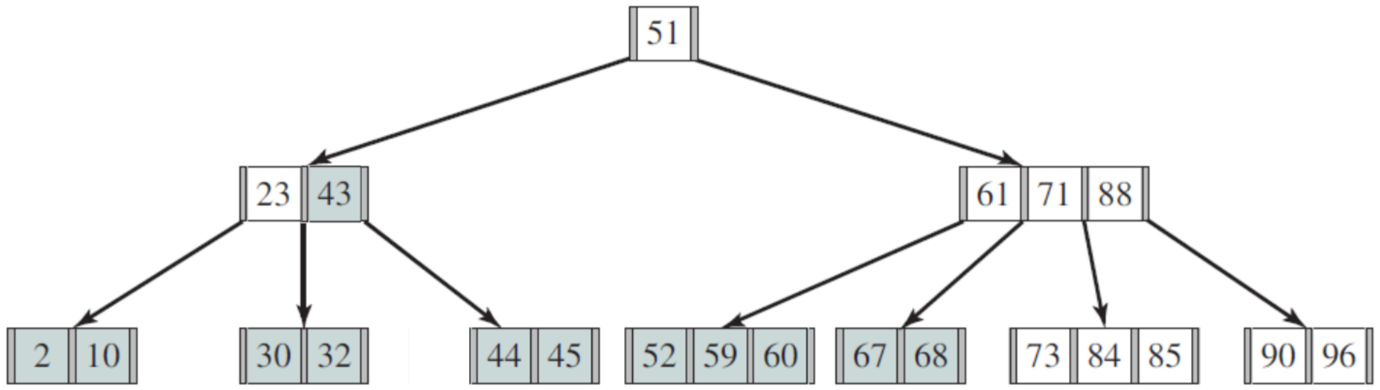


B-Trees

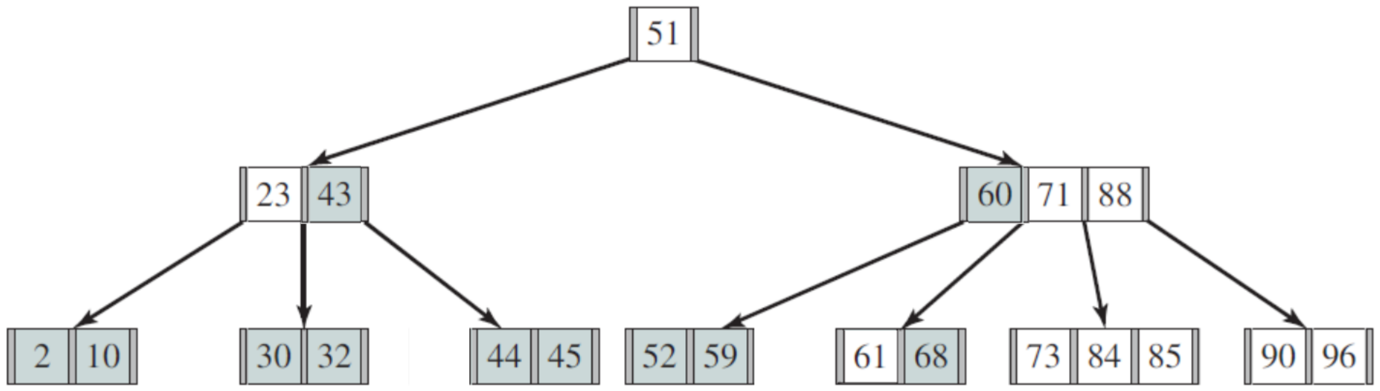
- Deleting a key k
  - Find the node that has k
    - If node is non-leaf: replace k with the successor (the smallest key larger than k) or the predecessor (the largest key smaller than k)
  - If the node has at least d – 1 keys, we are done
  - Otherwise
    - Borrow a key from a neighbor if the node has d or more keys (key rotation)
    - Merge with a neighbor that has d – 1 keys and with the medium key from the parent
  - Recursively perform the second step if a key was removed from the parent



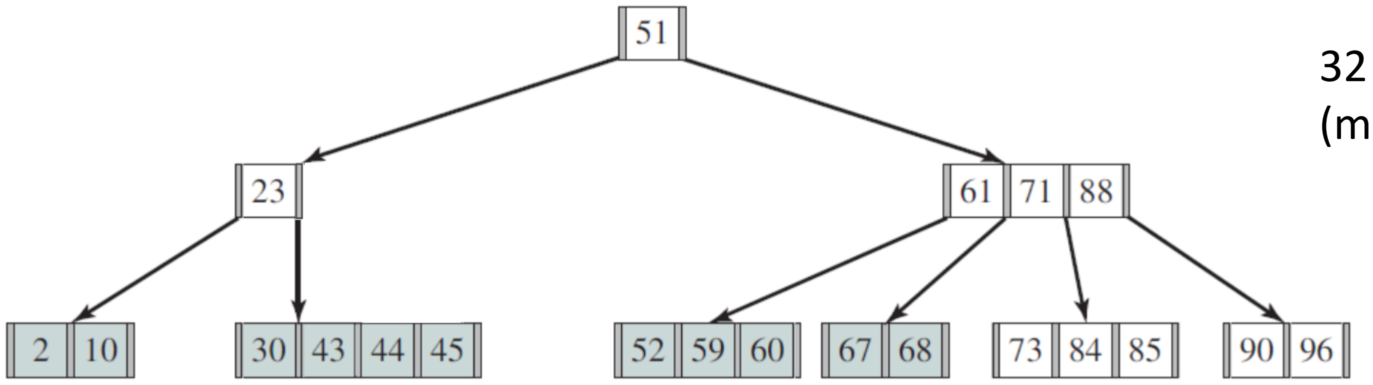
B-tree (d=3)



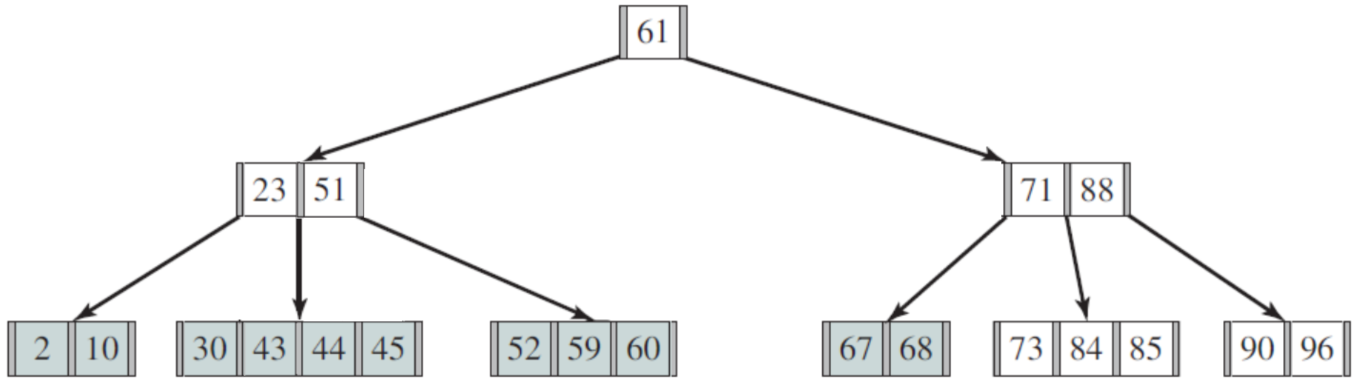
39 is deleted



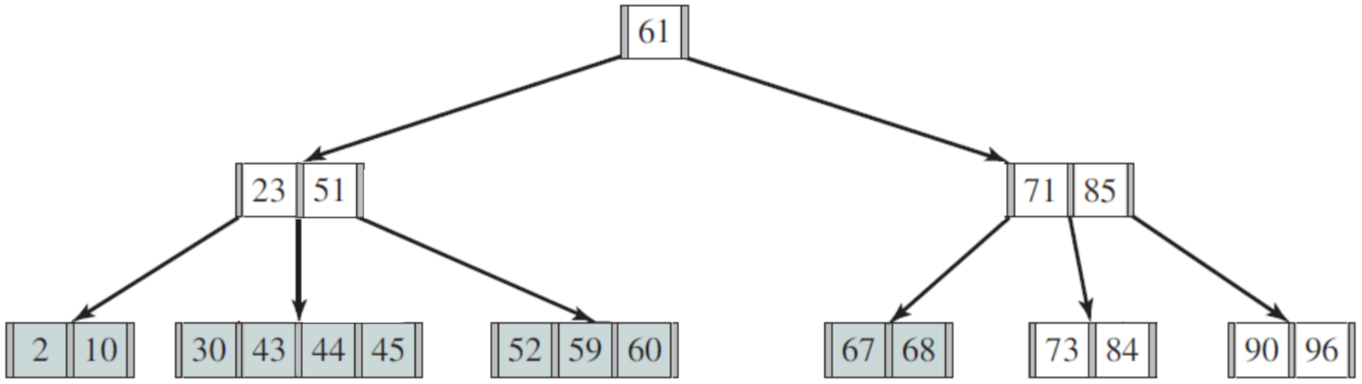
67 is deleted
(key rotation)



32 is deleted
(merge)

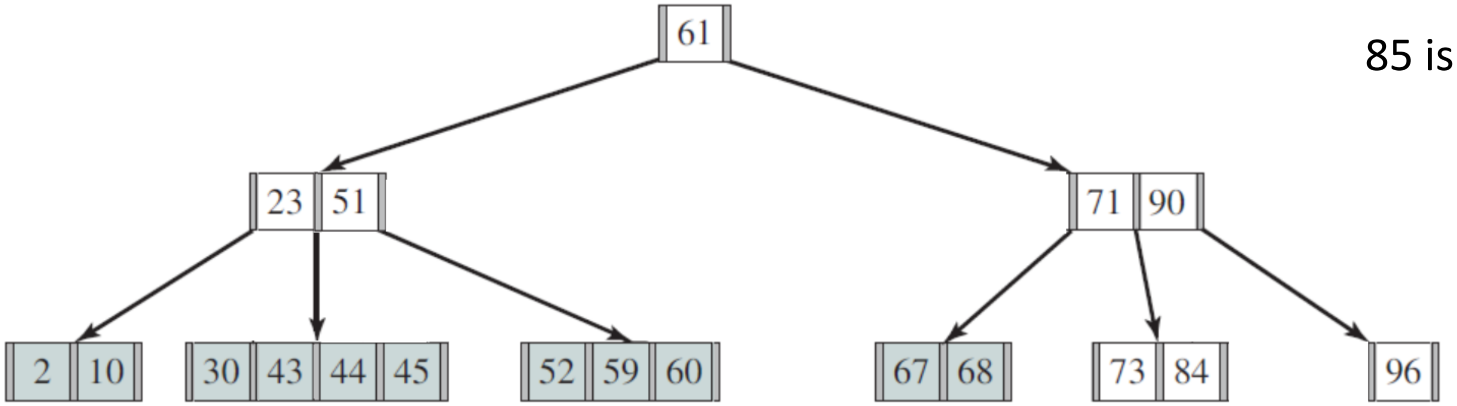


(key rotation)

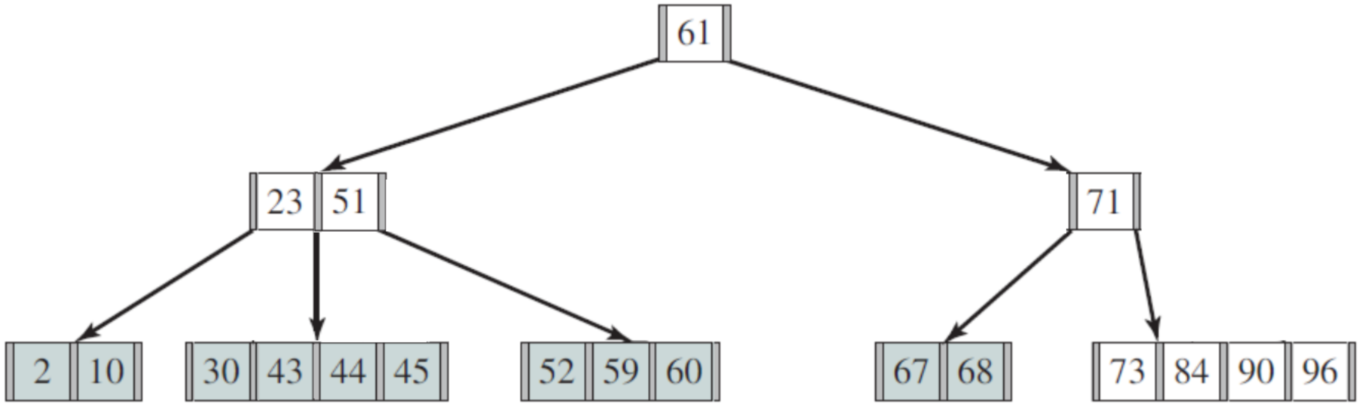


88 is deleted

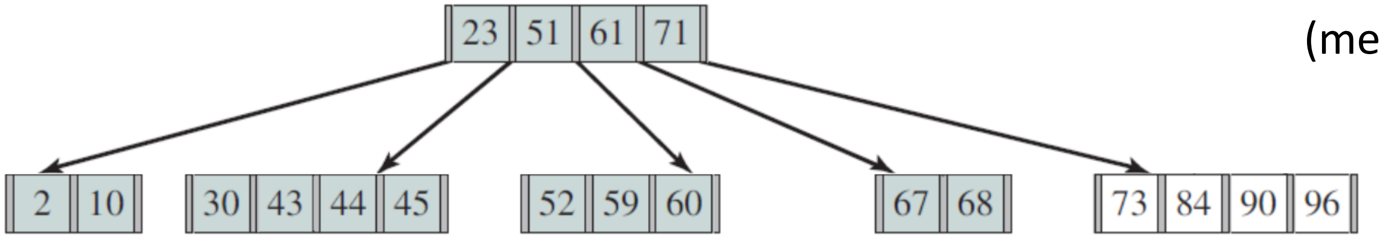
85 is deleted



(merge)



(merge)


23 51 61 71
2 10
30 43 44 45
52 59 60
67 68
73 84 90 96