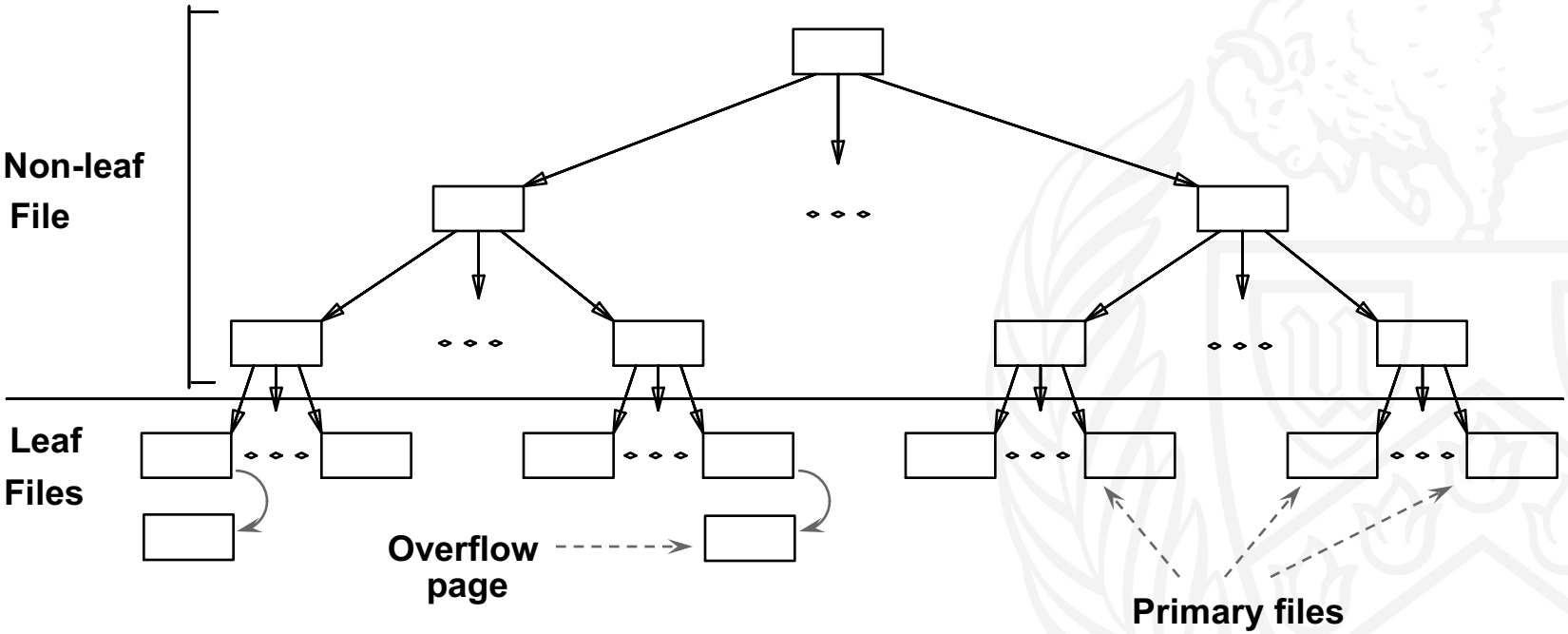
INDEXING

Tree-Structured Indices

- Tree-structured indexing techniques support both *range searches* and *equality searches*.
- *ISAM*: static structure; *B+ tree*: dynamic, adjusts gracefully under inserts and deletes.

ISAM

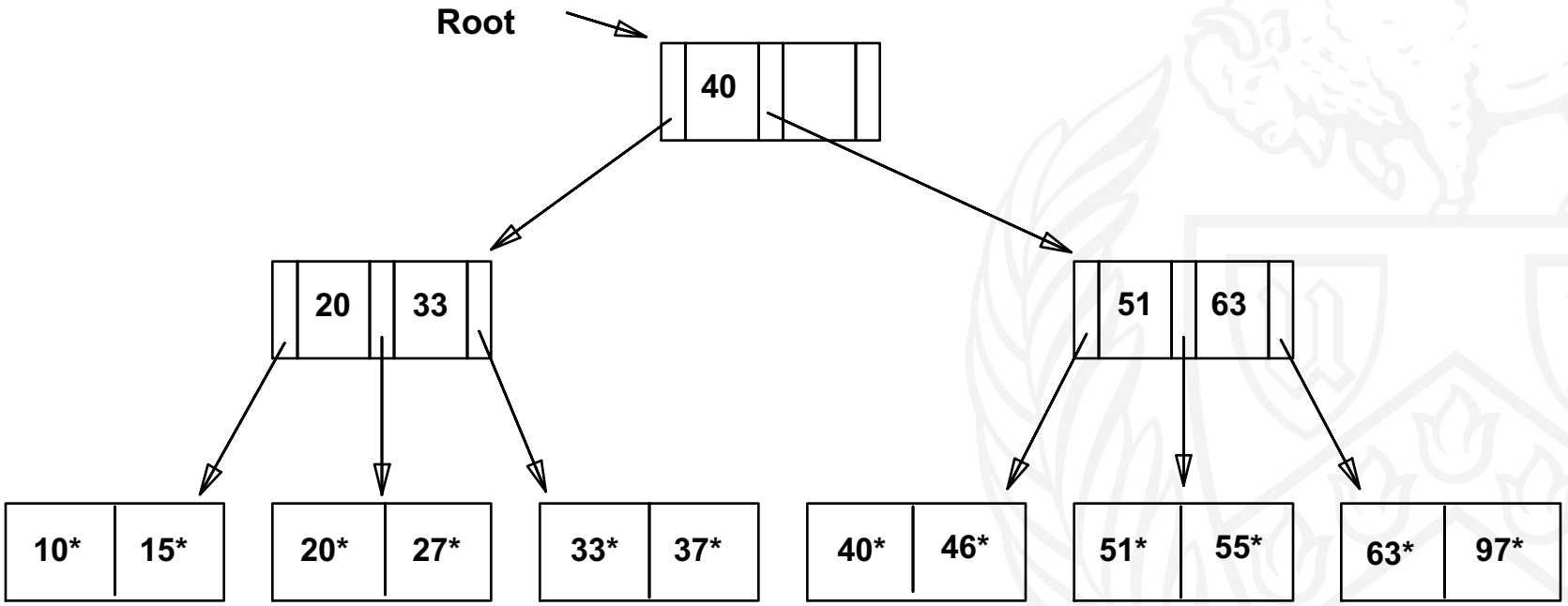
- Repeat sequential indexing until sequential index fits on one page.



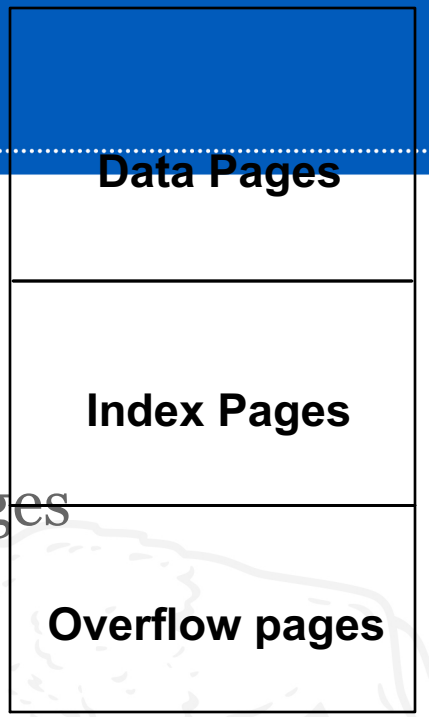
☛ *Leaf files contain data entries.*

Example ISAM Tree

- Each node can hold 2 entries; no need for `next-leaf-page' pointers. (Why?)



Comments on ISAM

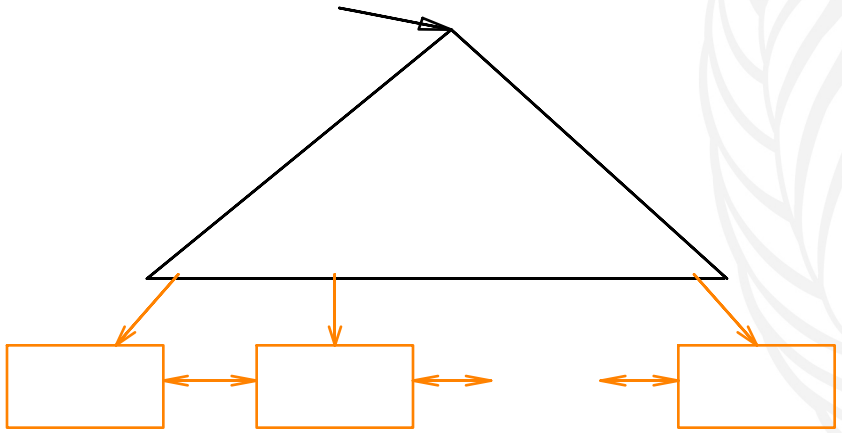


- *File creation*: Leaf (data) pages allocated sequentially, sorted by search key; then index pages allocated, then space for overflow pages.
- *Index entries*: <search key value, page id>; they `direct' search for *data entries*, which are in leaf pages.
- *Search*: Start at root; use key comparisons to go to leaf.
- *Insert*: Find leaf data entry belongs to, and put it there.
- *Delete*: Find and remove from leaf; if empty overflow page, de-allocate.

☛ **Static tree structure**: *inserts/deletes affect only leaf pages.*

B+ Tree: The Most Widely-Used Index

- Insert/delete at $\log_F N$ cost; keep tree *height-balanced.* (F (fanout) = # of entries/index pages, N = # leaf pages)
- Minimum 50% occupancy (except for root). Each node contains **d** <= *m* <= 2**d** entries. The parameter **d** is called the *order* of the tree.
- Supports equality and range-searches efficiently.

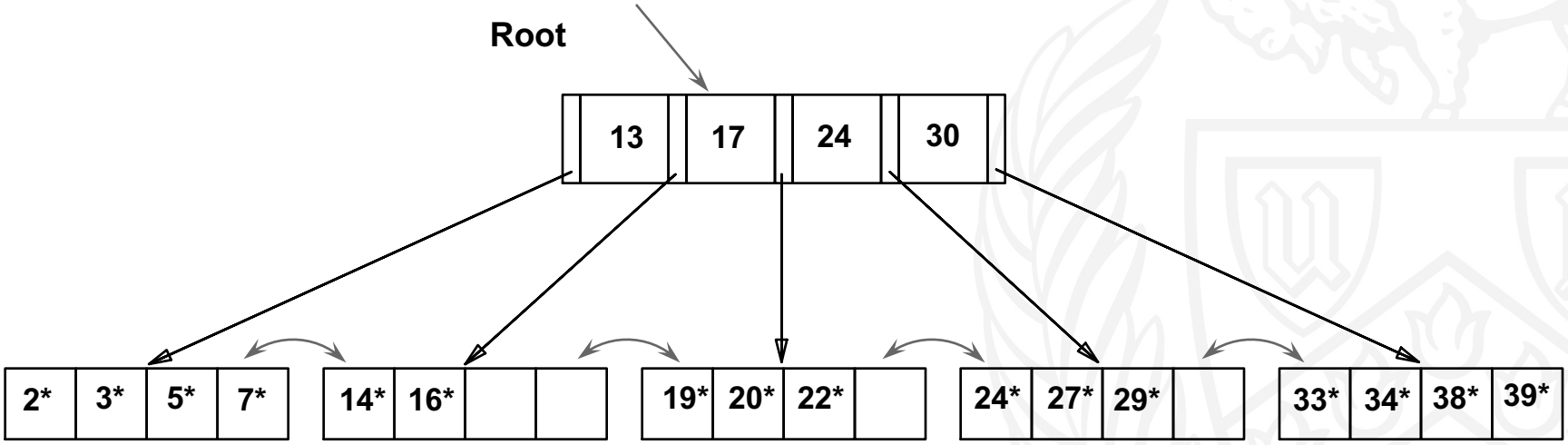


Index Entries
(Direct search)

Data Entries
("Sequence set")

Example B+ Tree

- Search begins at root, and key comparisons direct it to a leaf (as in ISAM).
- Search for 5*, 15*, all data entries >= 24* ...



☛ *Based on the search for 15*, we know it is not in the tree!*

Summary

- Tree-structured indexes are ideal for range-searches, also good for equality searches.
- ISAM is a static structure.
  - Performance can degrade over time – but OK for the project (No I/O)
- B+ tree is a dynamic structure.
  - Inserts/deletes leave tree height-balanced; $\log_F N$ cost.
  - High fanout (**F**) means depth rarely more than 3 or 4.
  - Almost always better than maintaining a sorted file.
  - Typically, 67% occupancy on average.
- Most widely used index in database management systems because of its versatility. One of the most optimized components of a DBMS.
- For projects, you can implement your own indexing mechanisms
  - Hash-based indexes
  - ISAM
  - Partitioning
  - Sorting
  - Binary Search, etc.