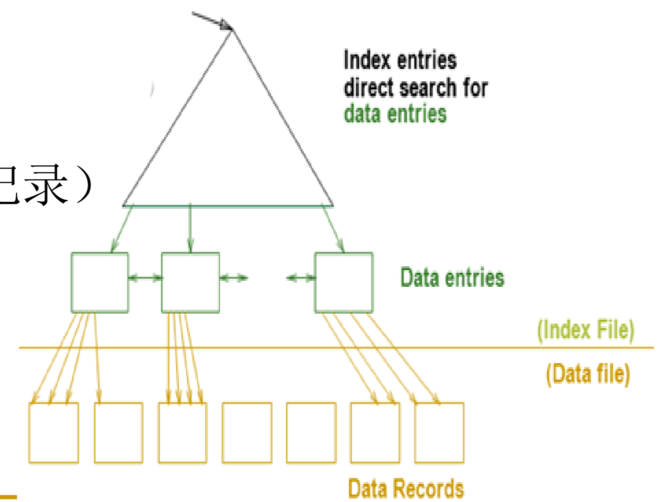
Tree-Structured Indexes

SUN YAT-SEN UNIVERSITY

Review

- Cost Model（代价模型）
  - 只关心磁盘块的IO次数
    - 文件由页组成，而每个页包含一组记录
    - *Record id* = <page id, slot #>
    - 从随机访问的角度来说，读写一条记录需要一次磁盘IO。
- 索引技术概述
  - 可以为关系建立索引，都是文件
  - 索引文件由两部份组成
    1. 数据项部分
       - Data Entry(数据项) ⟺ data record（数据记录）
    2. 引导部份
       - 树索引技术
       - Hash索引

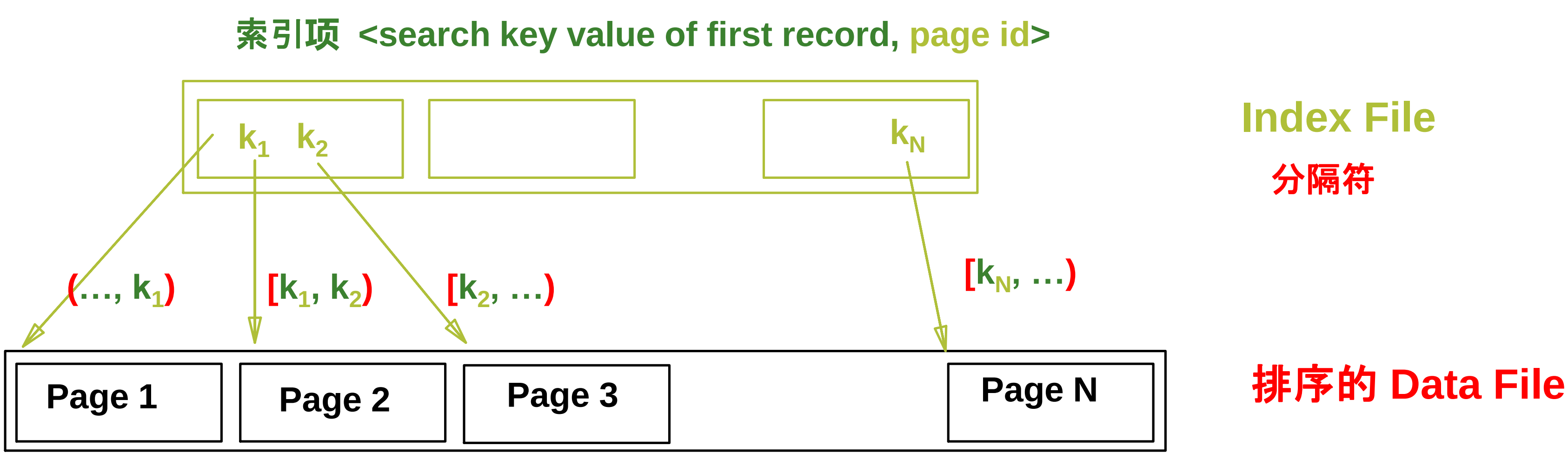


Tree-Structured Indexes: Introduction

- Tree-structured indexing techniques support both *equality selections* and *range selections*.
  - *ISAM(*索引顺序存取方法*)*: static structure; early index technology.
  - *B+ tree*: dynamic, adjusts gracefully under inserts and deletes.

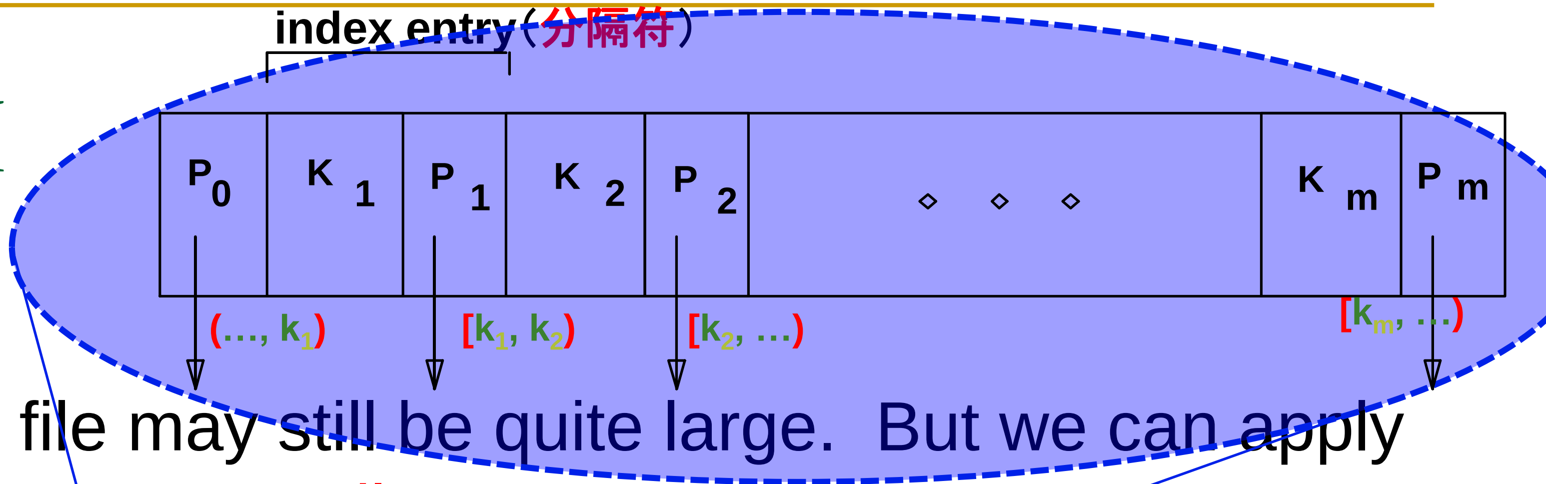
Range Searches

- ``*Find all students with gpa > 3.0*''
  - If data is in sorted file, do binary search to find first such student, then scan to find others.
  - Cost of binary search in a database can be quite high.
    - Why???
- Simple idea: Create an `index' file.

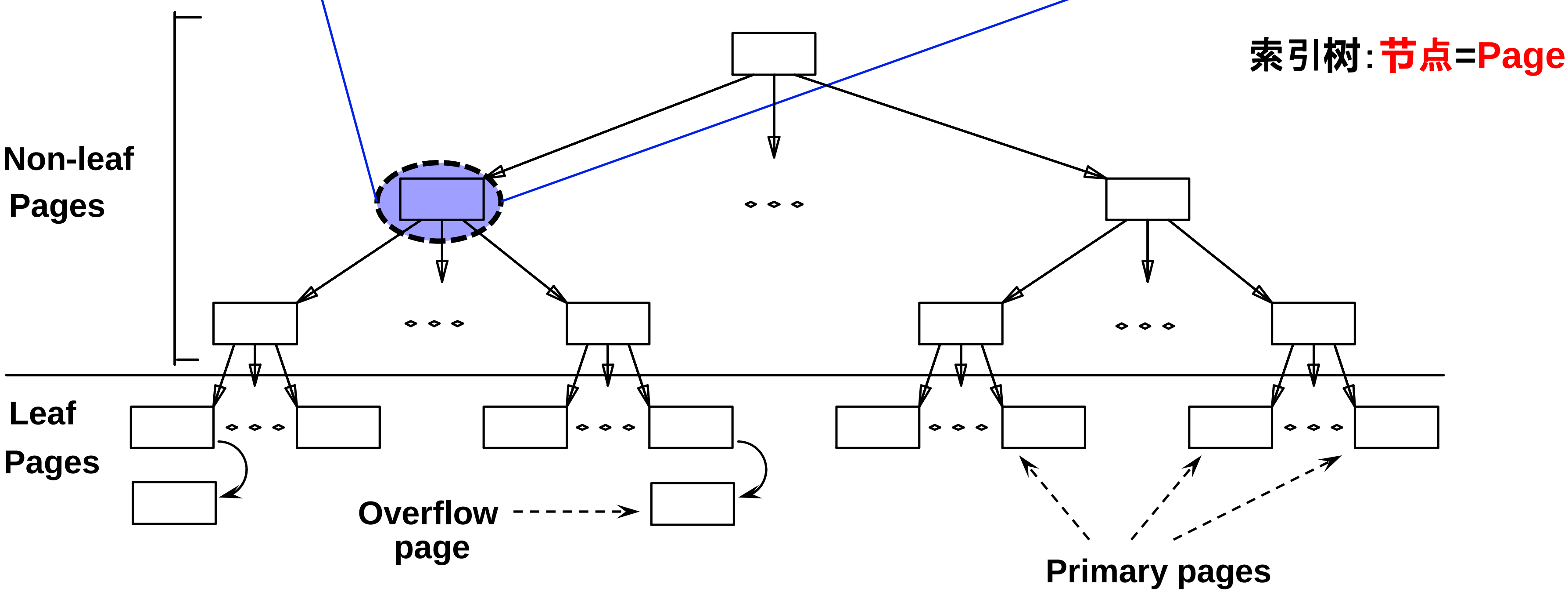


Can do binary search on (smaller) index file!

ISAM

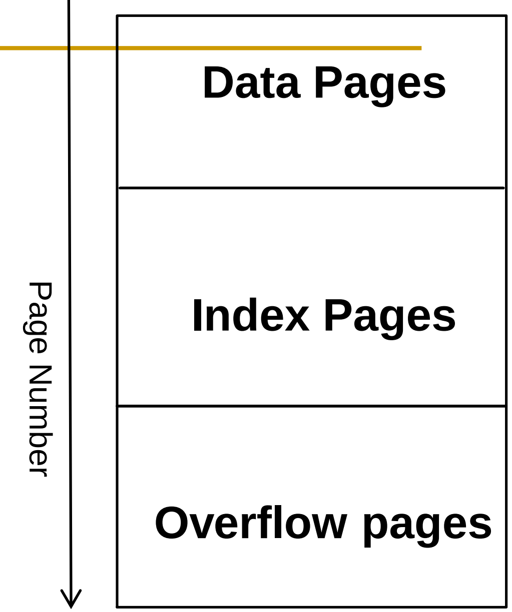


- Index file may still be quite large. But we can apply the idea repeatedly!

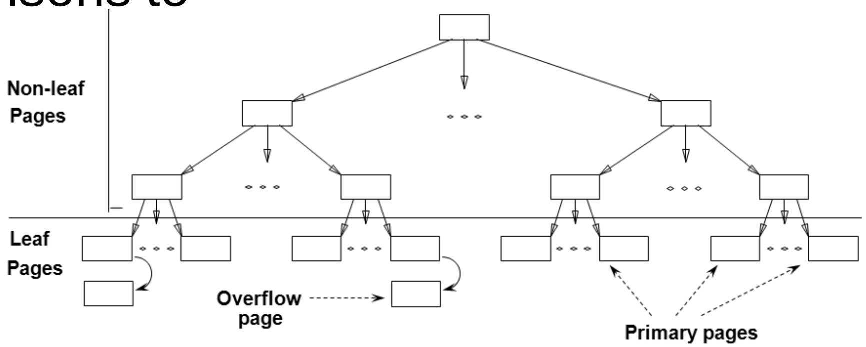


- *Leaf pages contain data entries.*

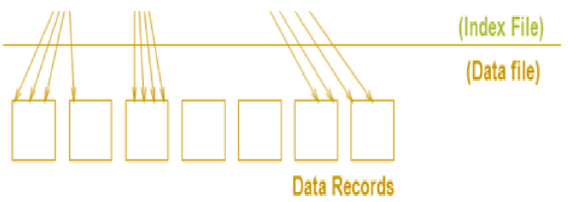
ISAM is a STATIC Structure



- *File creation*:
  - Leaf (data) pages allocated sequentially, sorted by search key
  - then index pages
  - then overflow pgs.
- *Search*: Start at root; use key comparisons to go to leaf.
- Cost = $\log_F N$
  - F = # entries/page (i.e., fanout 扇出)
  - N = # leaf pages
  - no need for `next-leaf-page' pointers. (Why?)



- *Insert*: Find leaf that data entry belongs to, and put it there. Overflow page(溢出页) if necessary.
- *Delete*: Seek and destroy! If deleting a tuple empties an overflow page, de-allocate it and remove from linked-list.

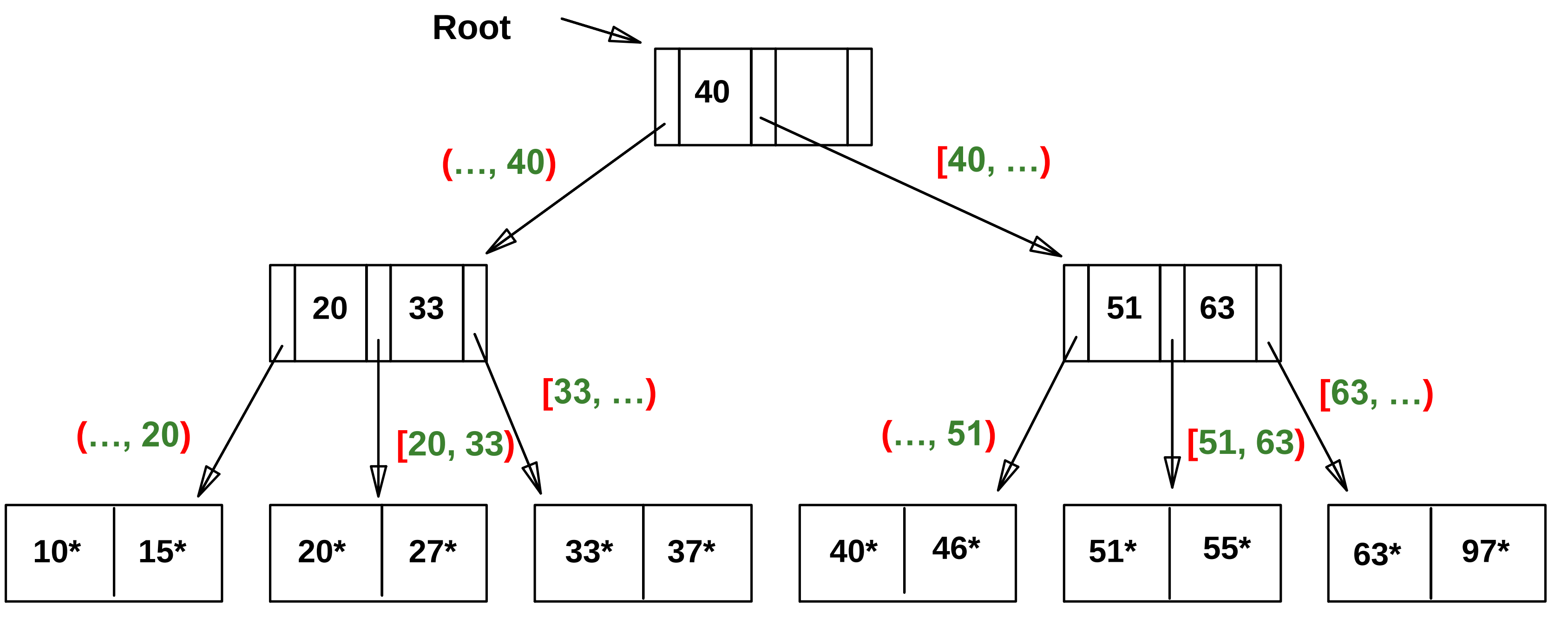


Static tree structure: *inserts/deletes affect only leaf pages*.

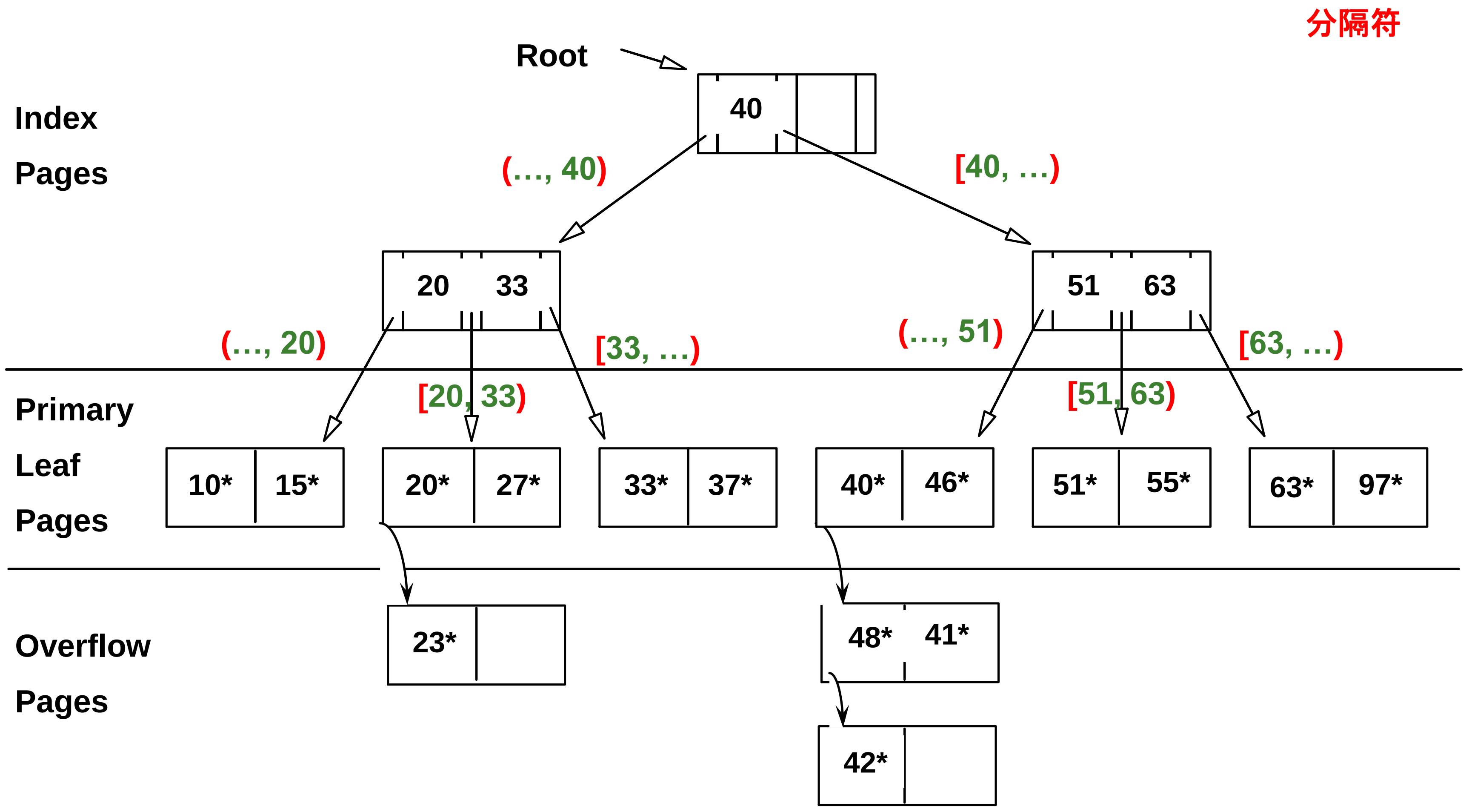
Example ISAM Tree

- **Example where each node can hold 2 entries;**
- ***Index entries*: <search key value, page id>, they direct search for data entries *in leaves.***

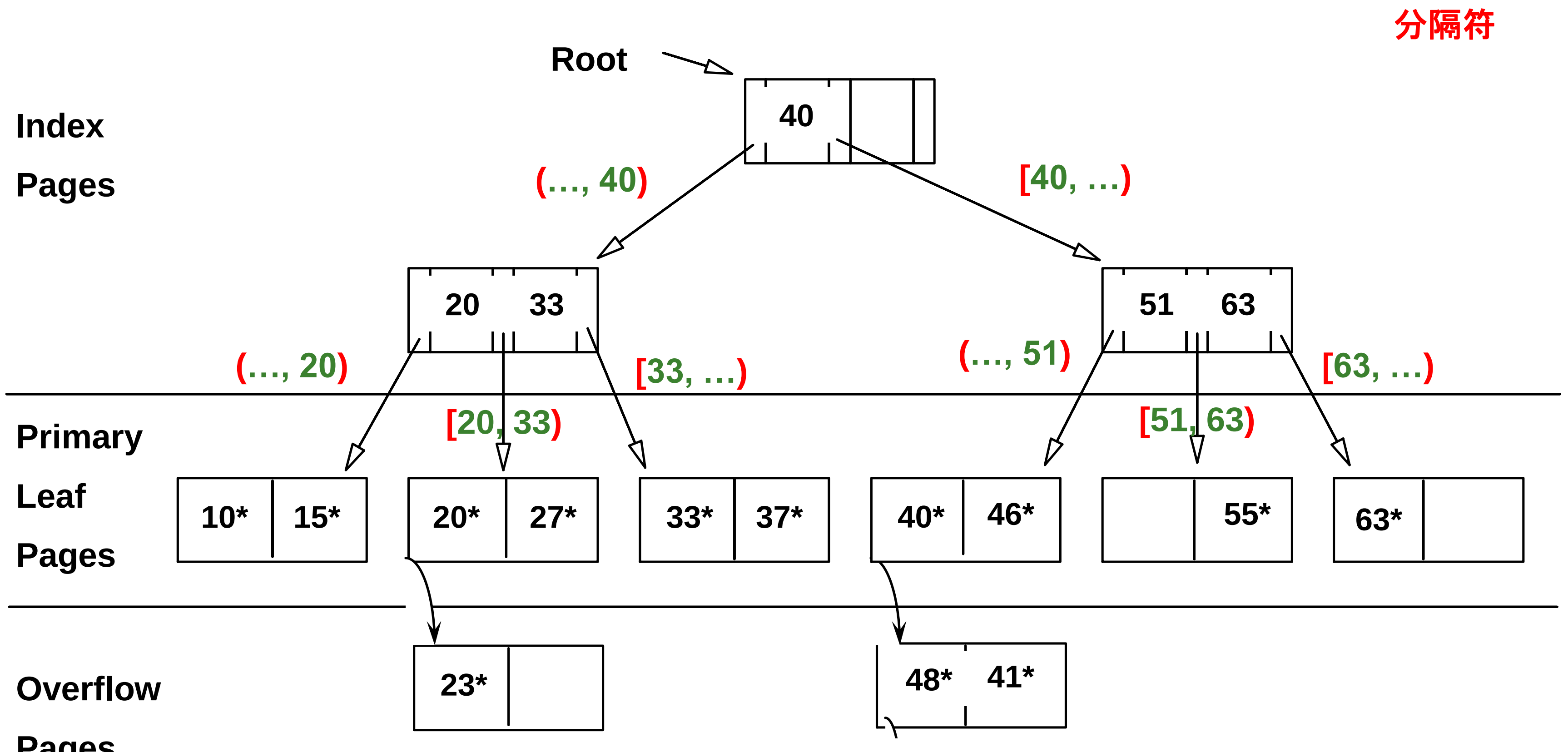
分隔符



Example: Insert 23*, 48*, 41*, 42*



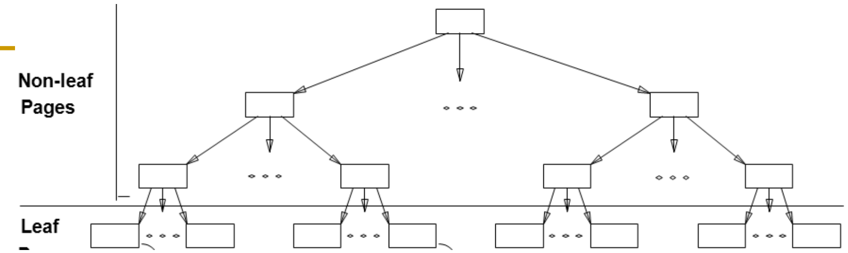
... then Deleting 42*, 51*, 97*



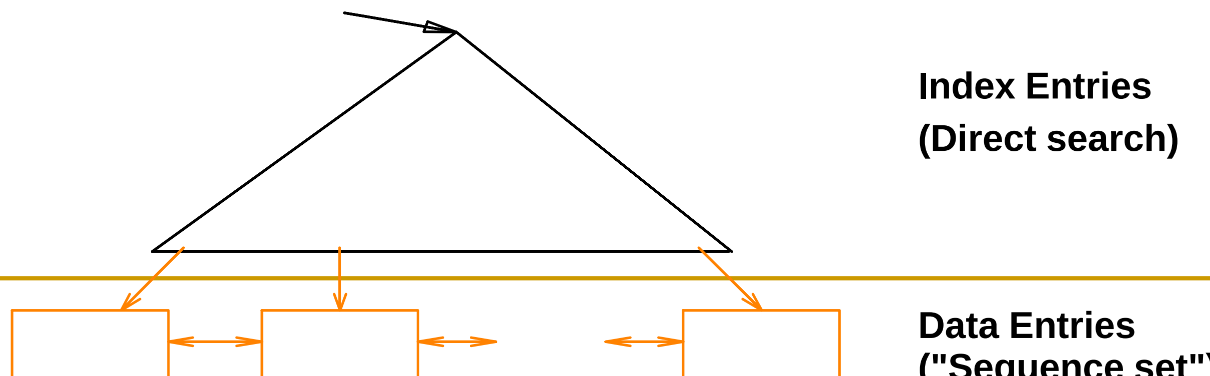
ISAM is a STATIC Structure

- *Note that 51 appears in index levels, but not in leaf!*

B+ Tree Structure

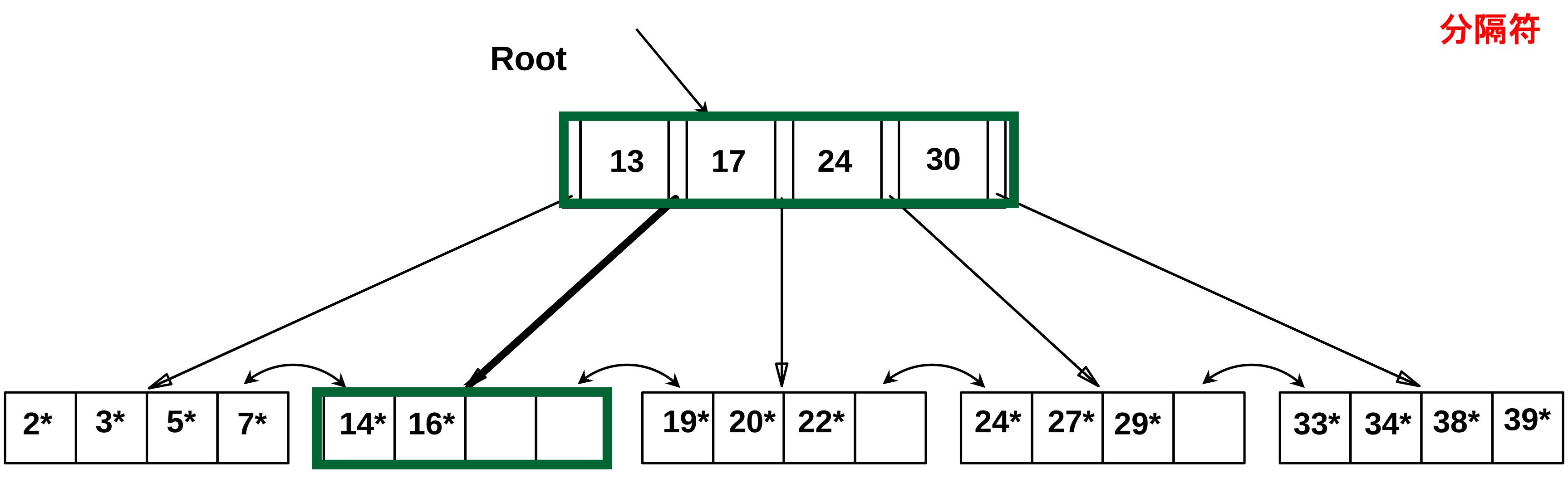


- Each node contains **d** <= <u>*m*</u> <= 2**d** entries (index or data)
  - The parameter **d** is called the *order*(秩) of the tree.
  - Each internal node contains *m* index entries: <key, page id>.
  - Each leaf node contains *m* data entries: <key, record or record id>
- The ROOT node contains between 1 and 2**d** index entries.
  - It is a leaf or has at least two children.
- Each path from the ROOT to any leaf has the same length. - 平衡树 -- 树的高度
- Supports equality and range-searches efficiently.



B+ Tree Equality Search

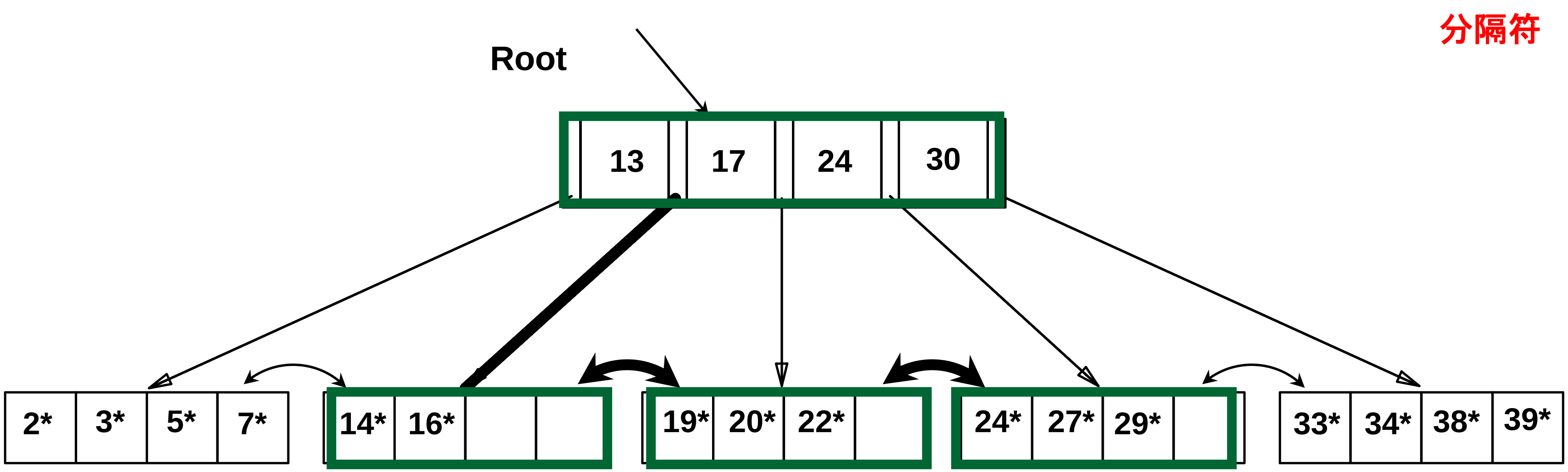
- Search begins at root, and key comparisons direct it to a leaf.
- Search for 15*…



Based on the search for 15, we <u>know</u> it is not in the tree!*

B+ Tree Range Search

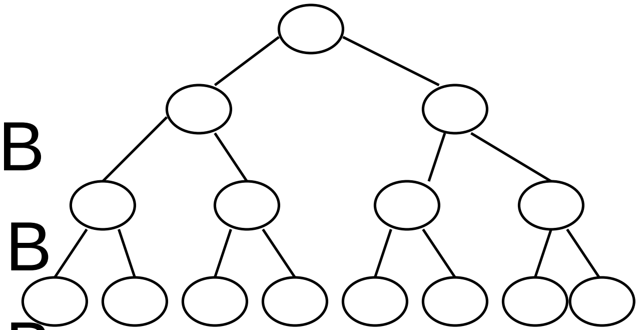
- Search all records whose ages are in [15,28].
  - Equality search 15*.
  - Follow sibling pointers.



B+ Trees in Practice

d <= *m* <= 2**d**

- Typical order (秩) : 100. Typical fill-factor: 67%.
  - average fanout(扇出) = 133 2d*2/3
  - Level 1 = 1 page = 8 KB
  - Level 2 = 133 pages = 1 MB
  - Level 3 = 17,689 pages = 145 MB
  - Level 4 = 2,352,637 pages = 19 GB



- Can often hold top levels in buffer pool:
  - With 1 MB buffer, can locate one record in 19 GB (or 0.3 billion records) in two I/Os!