Storage &
Indexing

B+ trees and RAID levels

(courtesy : The University of Sydney)

Index structures

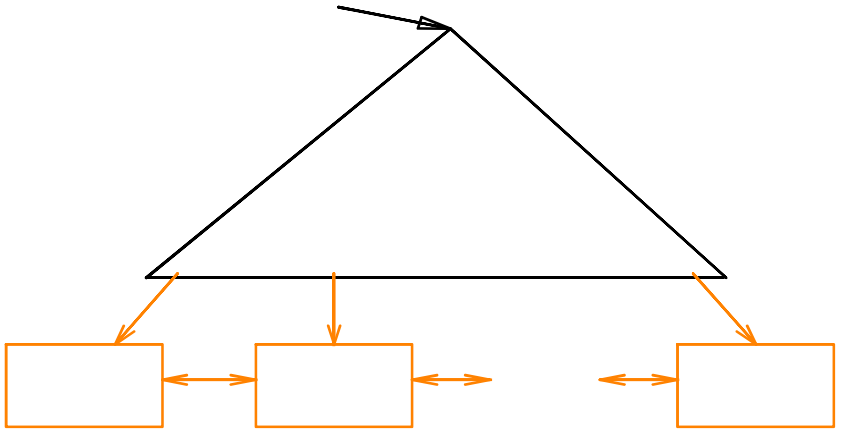
- There are many alternatives for how to arrange the data entries in the index
- And often there are higher levels of pointers that lead to the data entries
  - A tree structure for the index
- Different vendors offer different choices
- This can have some impact on performance
- But the biggest performance problems usually come from not having index at all!

Tree-Structured Indices

- Index Sequential Access Method (ISAM)
  - ordered sequential file with a (fixed) primary index.
  - Disadvantage of ISAM
    - performance degrades as file grows, since many overflow blocks get created.
    - Periodic reorganization of entire file is required.
- B+ Tree
  - dynamic multi-level index structure
  - most commonly used: B^+-Tree
  - reorganization of entire file is not required to maintain performance.
  - Disadvantages: extra insertion and deletion overhead and space overhead.

B+ Tree: Most Widely Used Index

- Insert/delete at log $_F$ N cost; keep tree *height-balanced*. (F = fanout, 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.



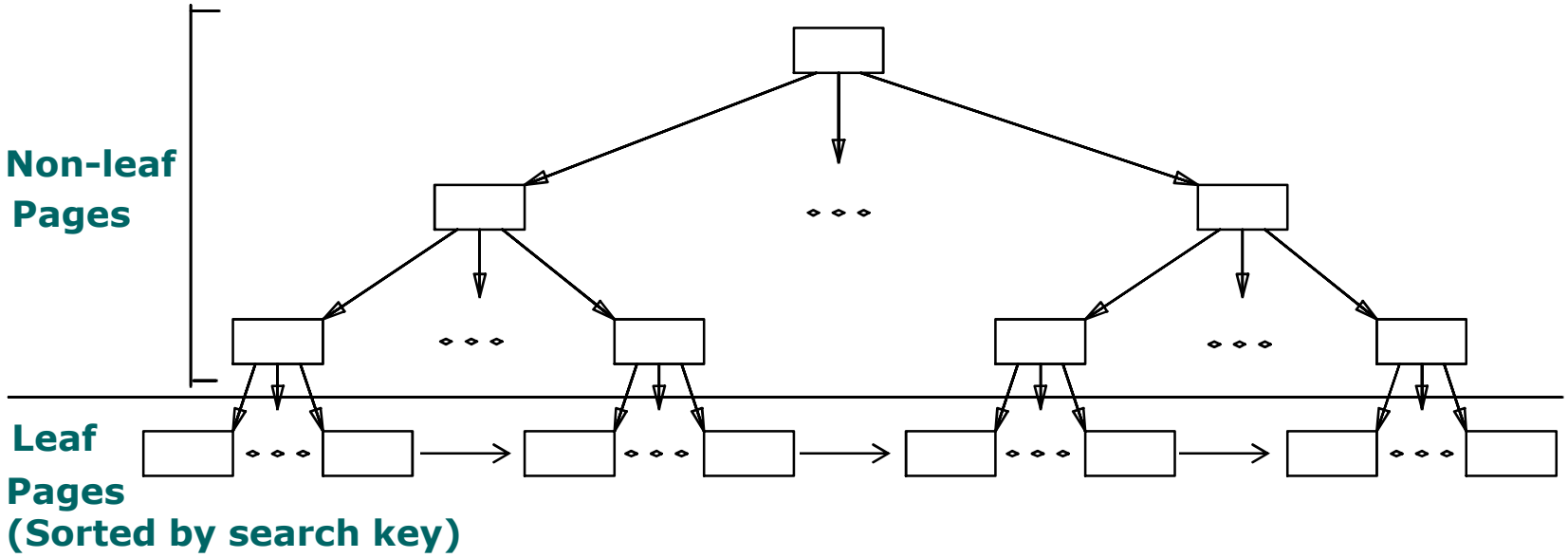
Index Entries

(Direct search)

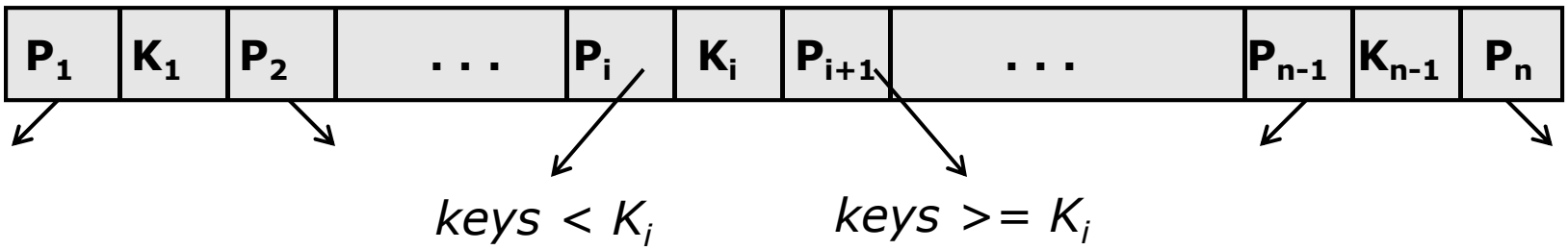
Data Entries

("Sequence set")

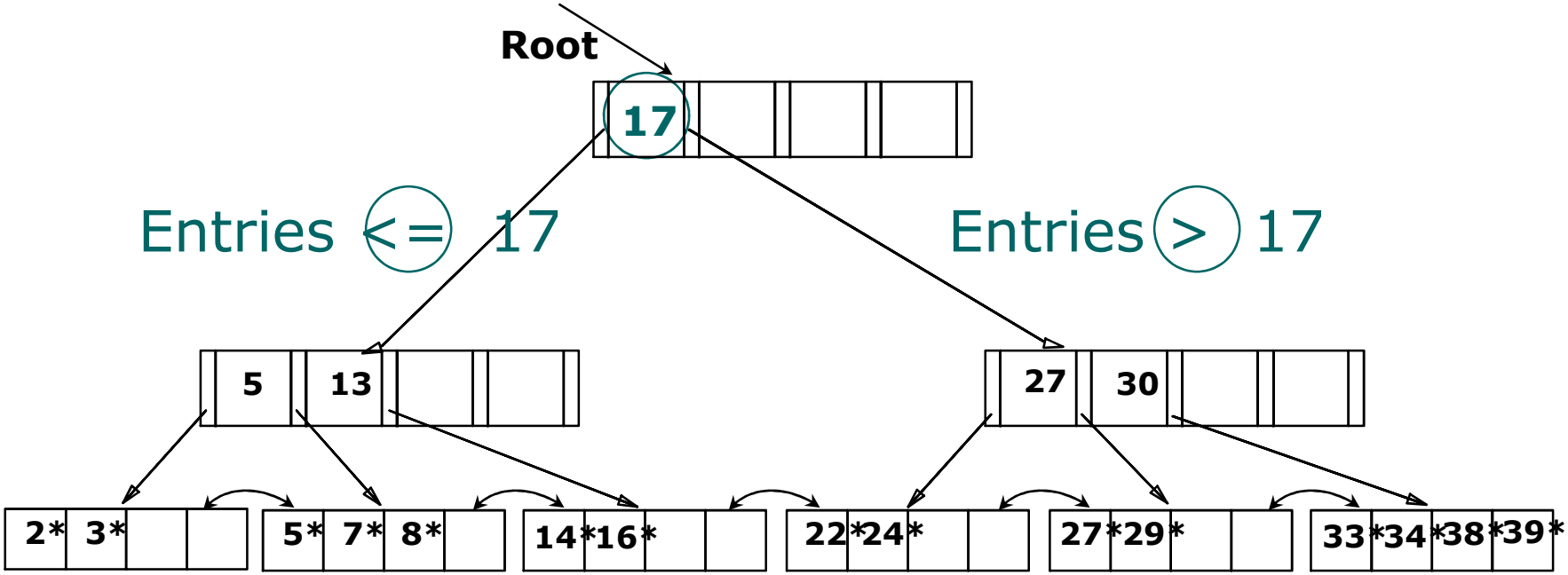
B+-Tree Index



- Leaf nodes contain *data entries*, and are chained
- Non-leaf nodes have *index entries*;only used to direct searches



Example of a B+-tree



- Note how data entries in the leaf level are sorted
- Find 28? 29? All > 15 and < 30?
- Insert/delete:
  - Find data entry in leaf, then change it.Need to adjust parent sometimes.

B+-Tree Index Structure

- A B^+-tree is a rooted tree satisfying the following properties:
  - All paths from root to leaf are of the same length
    - i.e., it is a *balanced tree*
  - Each node that is not a root or a leaf has between $[n/2]$ and n (pointers to) children.
    - The number of pointers in a node is also called *fanout*
    - The search keys within a node are sorted.
  - A leaf node has between $[(n–1)/2]$ and $n–1$ values
  - Special cases:
    - If the root is not a leaf, it has at least 2 children.
    - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and $(n–1)$ values.

Queries on B+-Trees

- Find all records with a search-key value of *k.*

1. Start with the root node
   1. Examine the node for the smallest search-key >= *k.*
   2. If such a value exists, assume it is K_i. Then follow P_i to the child node
   3. Otherwise $k \geq K_{n-1}$, where there are *n* pointers in the node. Then follow P_n to the child node.
2. Repeat the above procedure until a leaf node is reached.
3. Eventually reach a leaf node. If for some *i*, key K_i = k follow pointer P_i to the desired record or bucket. Else no record with search-key value *k* exists.

Updates on B+-Trees

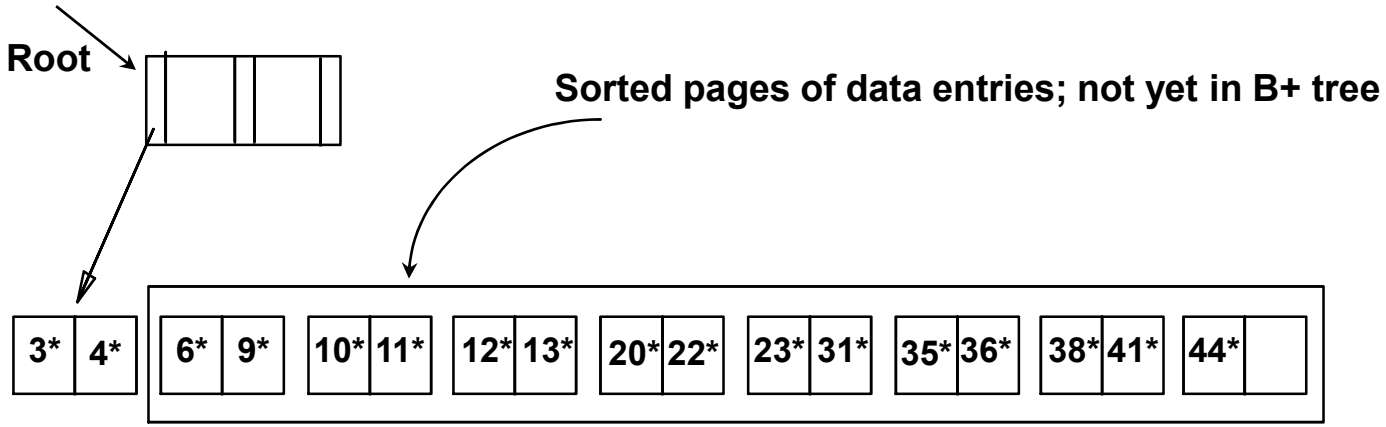
- To insert a data entry, when a data record is inserted (or when the search key is updated)
  1. Find where the new entry should be
  2. If there is room in that page, insert it
  3. If not, split the page into two, and redistribute the entries; then insert the new entry
  4. This may lead to further splits higher in the tree

The algorithm and code is complicated!

Deletion can be done similarly, but in practice the entry is simply removed, which may leave the page underfull

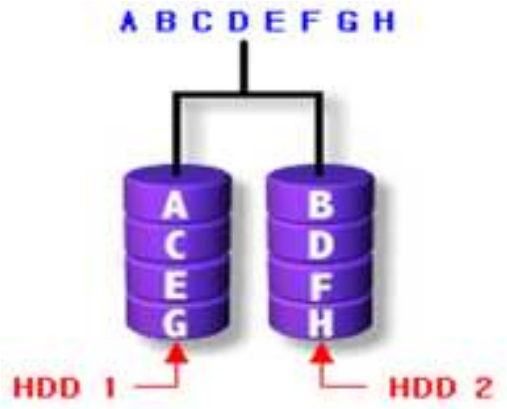
Bulk Loading of a B+ Tree

- If we have a large collection of records, and we want to create a B+ tree on some field, doing so by repeatedly inserting records is very slow.
- *Bulk Loading* can be done much more efficiently.
- *Initialization*: Sort all data entries, insert pointer to first (leaf) page in a new (root) page.

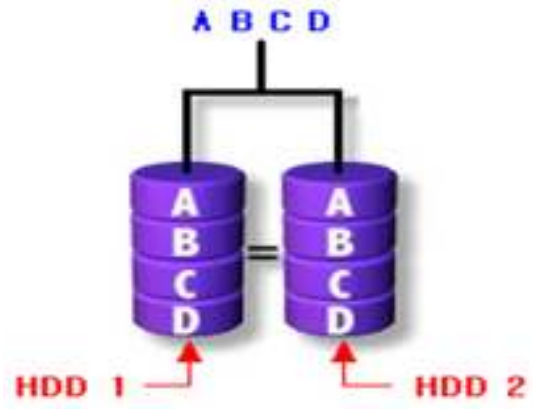


RAID Levels

- Schemes to provide redundancy at lower cost by using disk striping combined with parity bits
  - Different RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics
- RAID Level 0: Block striping; non-redundant.
  - Used in high-performance applications where data lost is not critical.
- RAID Level 1: Mirrored disks with block striping
  - Offers best write performance.
  - Popular for applications such as storing log files in a database system.



RAID 0: nonredundant striping

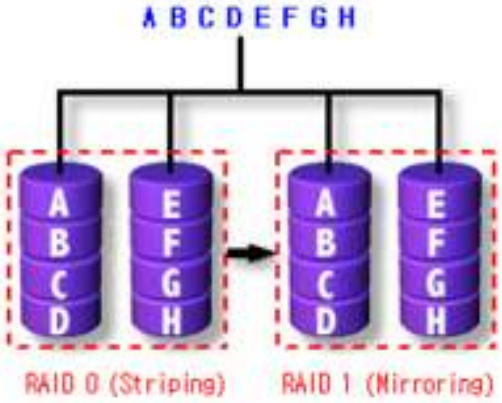


RAID 1: mirrored disks

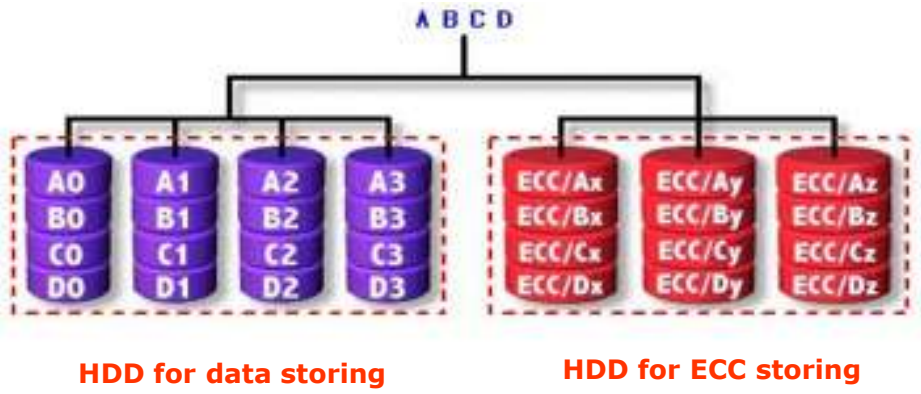
RAID Levels (Cont.)

- **RAID Level 0+1: Striping and Mirroring**
  - Parallel reads, a write involves two disks.

- **RAID Level 2: Memory-Style Error-Correcting-Codes (ECC) with bit striping.**
  - Striping unit is single bit
  - Store code for error correcting



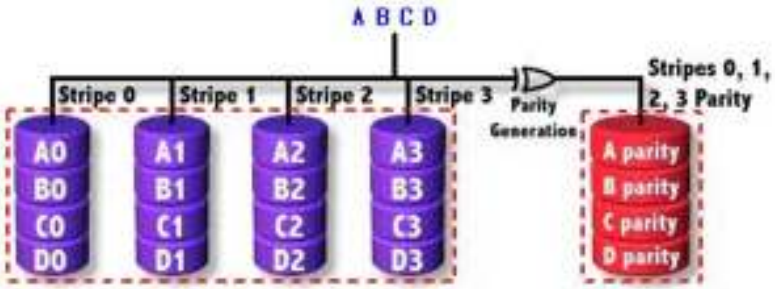
RAID 0+1: striping and mirroring



RAID 2: error correcting codes

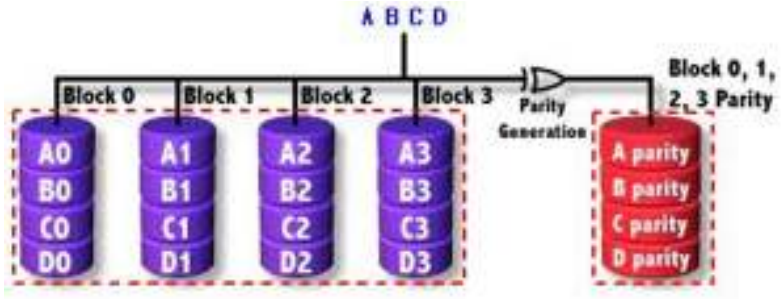
RAID Levels (Cont.)

- RAID Level 3: Bit-Interleaved Parity
  - a single parity bit is enough for error correction, since we know which disk has failed
    - When writing data, corresponding parity bits must also be computed and written to a parity bit disk
- RAID Level 4: Block-Interleaved Parity;
  - uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from N other disks.



HDD for data storing HDD for parity storing

RAID 3: bit-interleaved parity

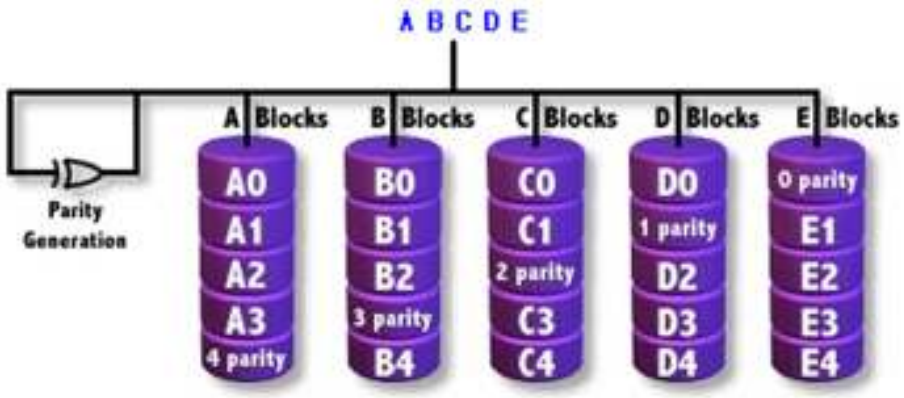


HDD for data storing HDD for parity storing

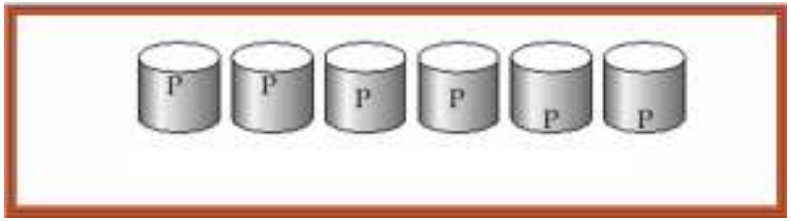
RAID 4: block-interleaved parity

RAID Levels (Cont.)

- RAID Level 5: Block-Interleaved Distributed Parity;
  - partitions data and parity among all N + 1 disks, rather than storing data in N disks and parity in 1 disk.
    - E.g., with 5 disks, parity block for nth set of blocks is stored on disk (n mod 5) + 1, with the data blocks stored on the other 4 disks.
- RAID Level 6: P+Q Redundancy scheme; similar to Level 5, but stores extra redundant information to guard against multiple disk failures.
  - Better reliability than Level 5 at a higher cost; not used as widely.



RAID 5: block-interleaved distribute parity



RAID 6: P+Q redundancy schem

Choice of RAID Level

- Factors in choosing RAID level
  - Monetary cost
  - Performance: # of I/Os per second and bandwidth during normal operation
  - Performance during failure
  - Performance during rebuild of failed disk / time to rebuild failed disk
- RAID 0 is used only when data safety is not important
  - e.g. data can be recovered quickly from other sources
- Level 2 and 4 never used since they are subsumed by 3 and 5
- Level 3 is not used anymore since bit-striping forces single block reads to access all disks, wasting disk arm movement, which block striping (level 5) avoids
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for almost all applications
- So competition is between 1 and 5 only
  - Level 5 is preferred for applications with low update rate, and large amounts of data
  - Level 1 is preferred for all other applications