ΗΥ360
Αρχεία και Βάσεις Δεδομένων

Διδάσκων: Δ. Πλεξουσάκης

Φυσική Σχεδίαση Βάσεων Δεδομένων

Υμεράλλη Ελισιάνα

Physical DB Design

- Data Structures for Primary Indices
  - Structures that determine the location of the records of a file.
  - Most common structures: heaps, hashed files, indexed files , B-Trees .
- The Heap Organization
  - Records are packed into blocks in no special order and with no special organization of the blocks.

Physical DB Design

Efficiency of Heaps

- n: number of records we need to store
- R: number of records that can fit in a block

The minimum number of blocks needed to store these records is [n/R].

If records are of variable length R is taken to be the average number of records that can fit in one block

Physical DB Design

Efficiency of Heaps:

- Lookup: must retrieve n/2R blocks on average. If there is no record with they key value, we must retrieve all n/R records.
- Insertion: must retrieve the last record on the heap. If the current block has no space, a new block must be used. In both cases, the block must be written to secondary storage after the insertion. Hence, insertion takes 2 block accesses.
- Deletion: requires n/2R block accesses to find the record and 1 more to delete it on average. If the record does not exist, n/R accesses are required.
- Modification: requires n/2R block accesses to find the record and 1 more to write the new values.

Physical DB Design

Hashed Files

- Records are divided into buckets according to the value of the key.
- A hash function h takes as argument a value for the key and produces an integer in the range 0 to B-1, where B is the number of buckets.
- Each bucket consists of a (usually small) number of blocks. The blocks in each bucket are organized as a heap.
- The bucket directory is an array of pointers indexed from 0 to B-1.
- The entry for i in the directory is a pointer to the first block of bucket i, called the bucket header. The blocks in a bucket form a linked list.

Physical DB Design

Hash Function

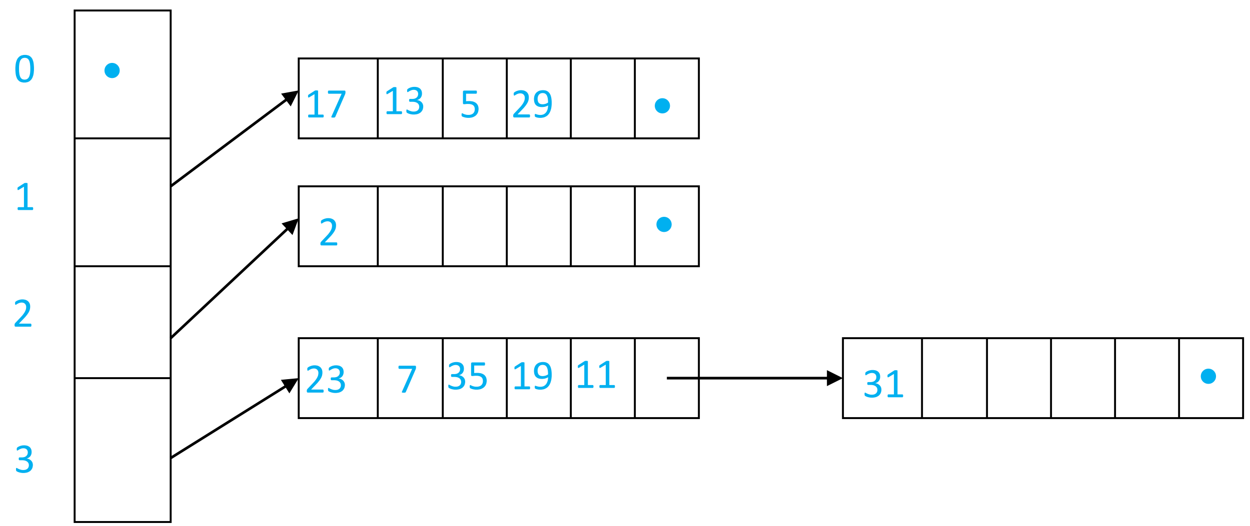
- A simple hash function :

  Convert each key value into an integer and then take the remainder of that integer modulo B(#buckets).
- If the key value v is an integer, then h(v) = v mod B

Physical DB Design

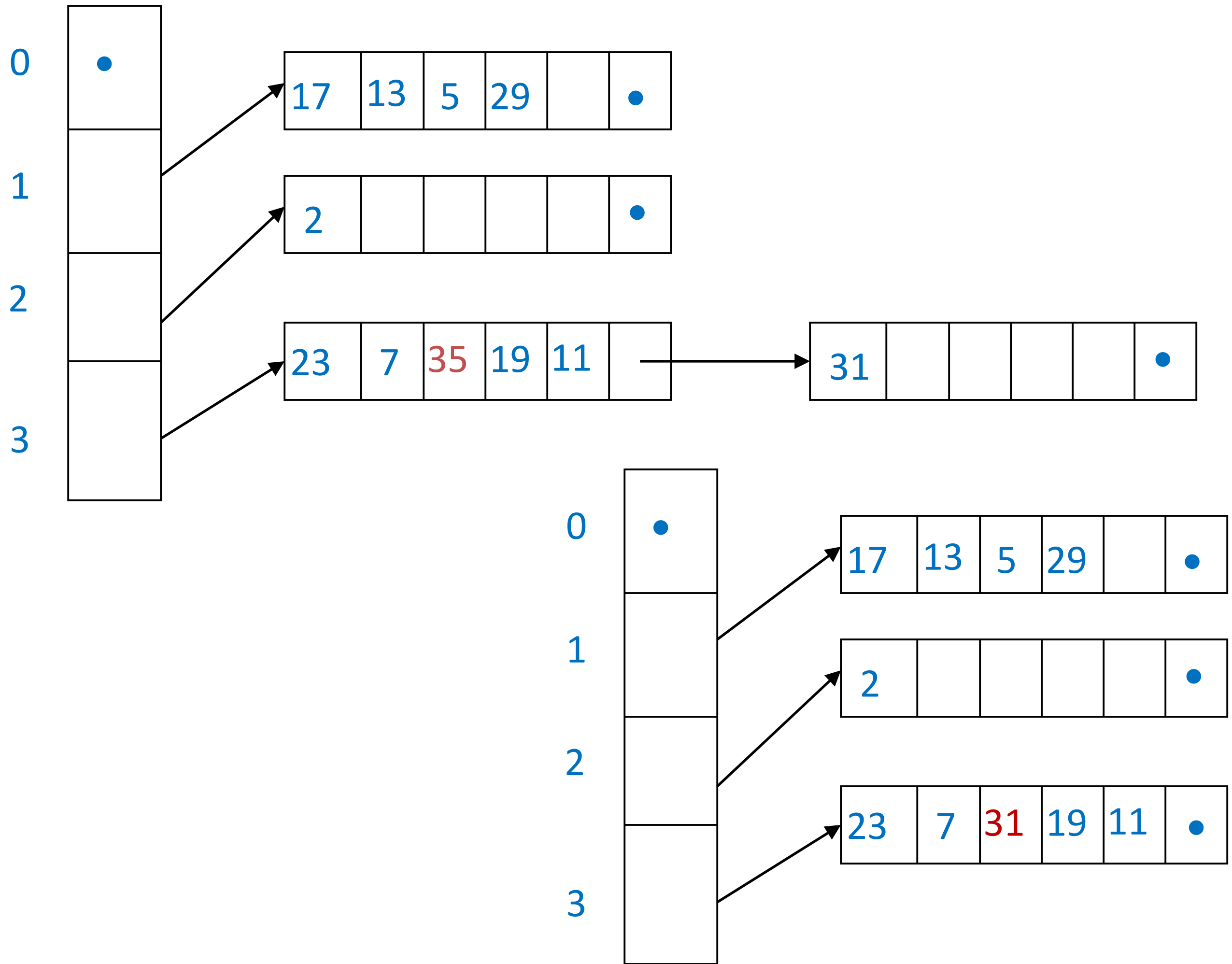
Example: h(v) = v mod B

A file of numbers organized in a hashed file with 4 buckets.



Physical DB Design

Example: Delete record with key value (35)



Physical DB Design

Efficiency of Hashing:

For a file of n records, of which R fit in a block, and for a hashed organization with B buckets (whose headers are kept in main memory) we require on average:

- $\lceil n/2BR \rceil$ for a successful lookup, deletion or modification of an existing record
- $\lceil n/BR \rceil$ for an unsuccessful lookup

Physical DB Design

Example:

- A file contains 1,000,000 records of 200 bytes each. Blocks are 2^{12}=4096 bytes long. R=20.
  - If B=1000, then the average bucket holds n/B=1000 records. These are distributed over n/BR=50 blocks.
  - If each block address requires 4 bytes, the bucket directory requires =1000 records * 4bytes =4000 bytes
  - An unsuccessful lookup takes 50 block accesses(n/BR=1,000,000/20000=50 blocks)
  - A successful lookup requires 26 block accesses on the average.

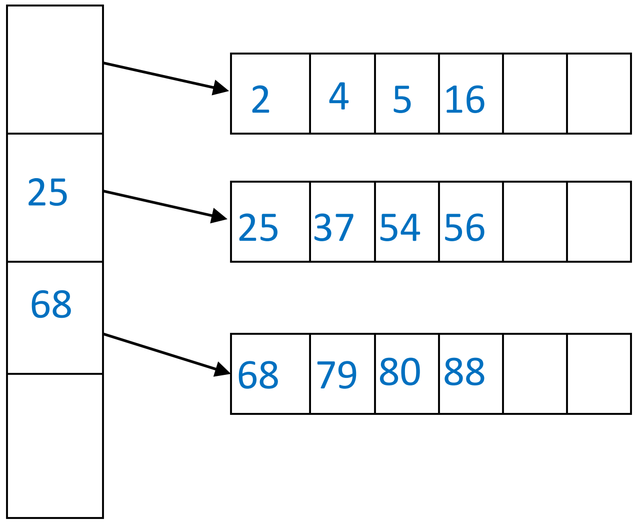
Physical DB Design

Sorted Files:

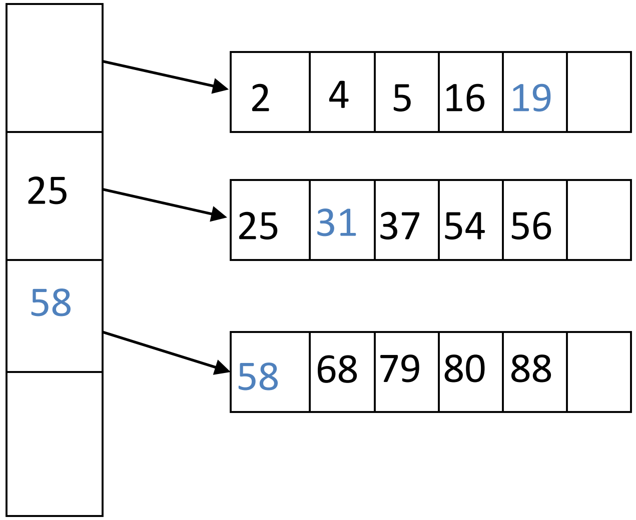
- A file called a sparse index is created for a sorted file. The index contains pairs of the form: (<key value>,<block address>).
- For every block b of the file, there is a record (v,b) in the index; v is a key value that is at least as low as any key value on b, but higher than any key value on any block preceding b.

Physical DB Design

Example: sorted list of numbers: 2,4,5,16,25,37,54,68,79,80,88



After insertion of numbers 19,58,31:



Physical DB Design

Searching Index Files:

- 1) Linear Search:

  Scan the index from the beginning, examining each record until the one that covers the one searched for is found

  Inefficient for large indices: half the index blocks will have to be examined on average in a successful lookup

  Index records are usually shorter than file records.

Physical DB Design

Searching Index Files:

2) Binary Search: : assume $B_1, B_2, \ldots, B_n$ are the blocks of the index file and $v_1, v_2, \ldots, v_n$ are the keys of the first records in the respective blocks. To locate record with key v:

- Retrieve index block $B_{\lceil n/2 \rceil}$ and let w be the value of its key: if v<w, repeat the search for the blocks $B_1, B_2, \ldots, B_{\lceil n/2 \rceil - 1}$; if v>=w, repeat the search for the blocks $B_{\lceil n/2 \rceil} \ldots B_n$; when only one block remains, use linear search to find the record.
- Roughly $\log_2 n$ block accesses are needed.

Physical DB Design

- Example: A file contains 1,000,000 records of 200 bytes each. Blocks are 2^{12}=4096 bytes long. The length of the key fields is 20 bytes.
  - R=20, hence the main file uses 50,000 blocks (n/R=1,000,000/20). The same number of records is needed for the index file.
  - An index record used 24 bytes: 20 bytes for the key, 4 for a pointer to a block. 170 index records can fit in one block(4096/24=170 records) if no additional bits are used. Then 50,000/170=294 blocks are needed for the index file.
  - Linear search would require about 147 block accesses on average for a successful lookup.
  - Binary search requires about $\log_2 294 = 9$ block accesses.

Physical DB Design

- Example(cont'd):
  - Hashed organization would only require 3 accesses: (1 to read the bucket directory, and 2 to read/write the block)
  - However, binary search is preferable to hashed organization for answering range queries, i.e., queries of the form “retrieve all records with keys in the range (a,b)”. A hashed organization would require examining practically all buckets.

Physical DB Design

B-Tree (Balanced Tree):

- B-tree of order m is a tree with the following properties:
  - The root has at least 2 children, unless it is a leaf.
  - No node in the tree has more then m children.
  - Every node except for the root and the leaves have at least $\lceil m/2 \rceil$ children.
  - All leaves appear at the same level.
  - An internal node with k children contains exactly k-1 keys.

Physical DB Design

Operations on B-trees:

- Lookup: Search for a record with key value v, find a path from the root of the B-tree to some leaf node where the desired record will be found if it exists
- Insertion: Insert a record with key value v
- Deletion: Delete a record with key value v

Physical DB Design

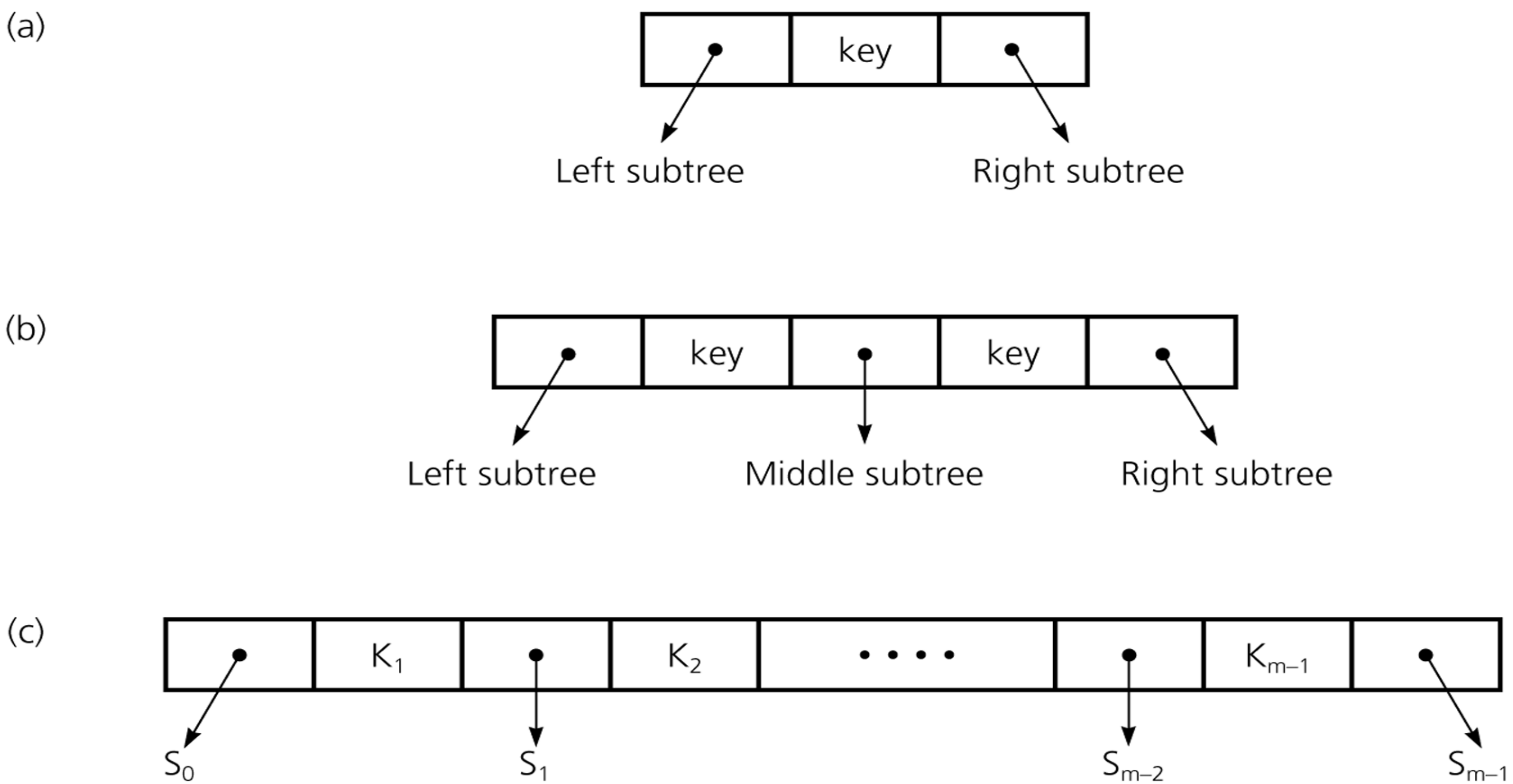
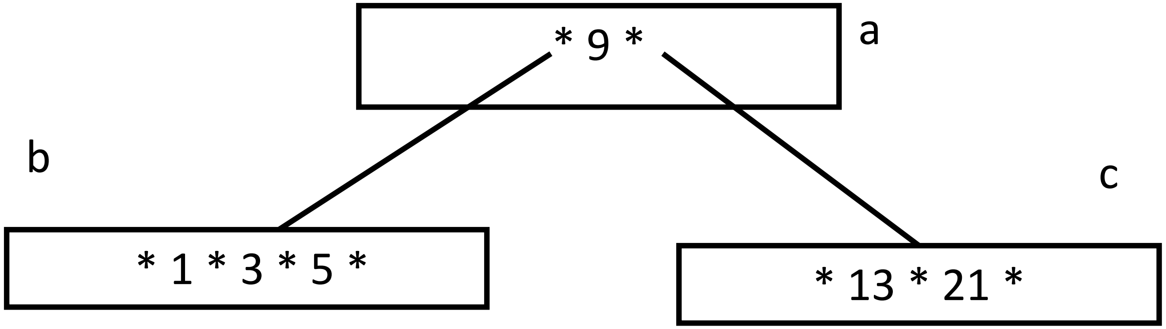


Figure 1

a) A node with two children; b) a node with three children; c) a node with *m* children

Physical DB Design

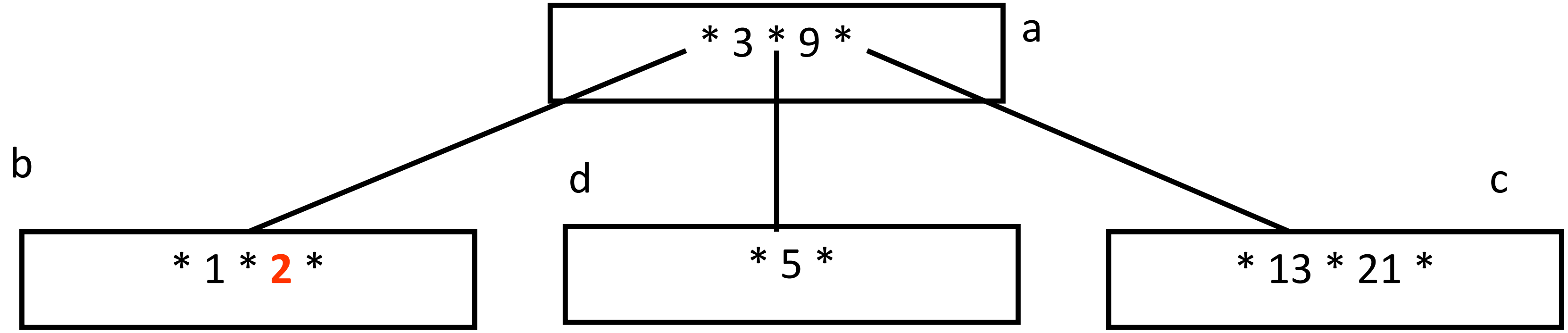
- B-Tree:



Nodes b and c have room to insert more elements

Physical DB Design

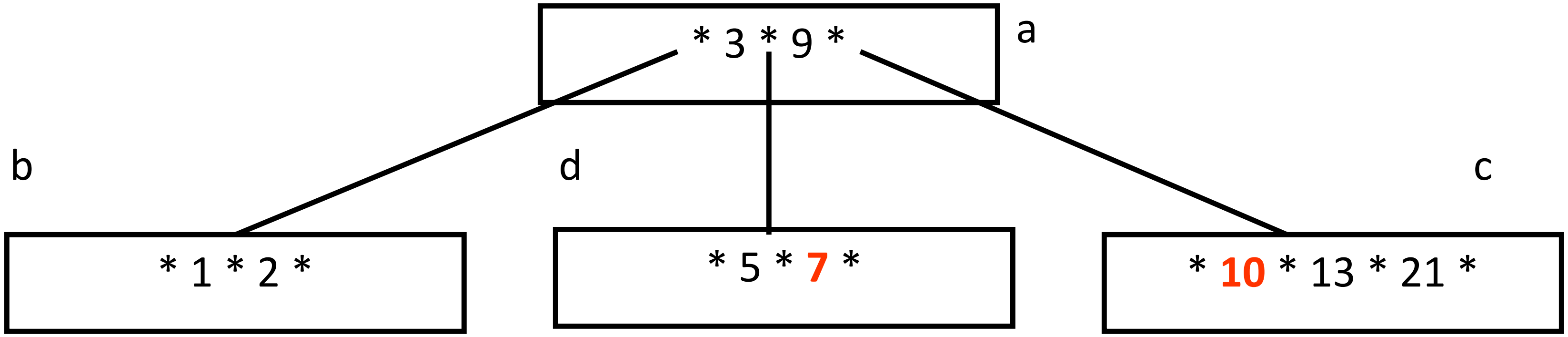
- Insert 2:



Node b has no more room, so it splits creating node d.

Physical DB Design

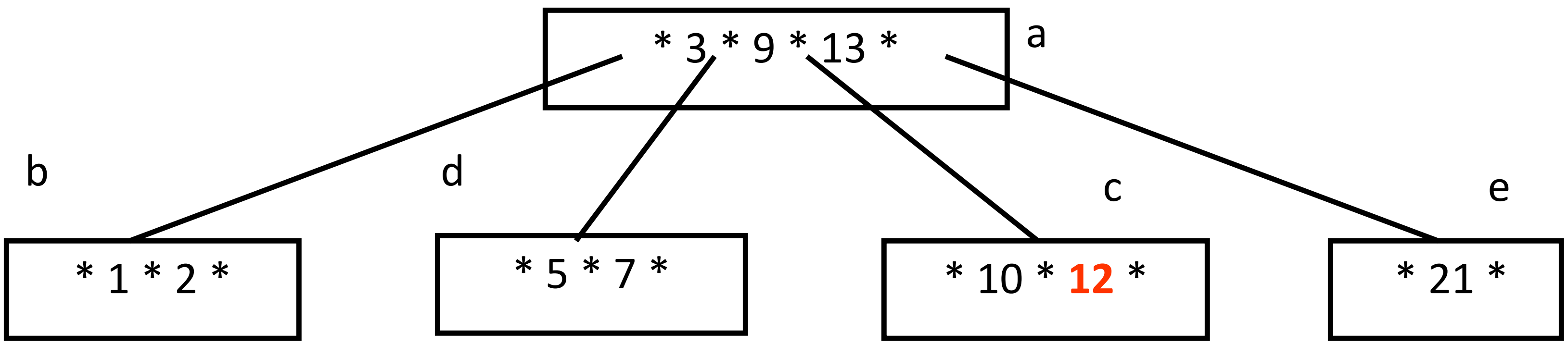
- Insert 7,10 :



Nodes d and c have room to add more elements

Physical DB Design

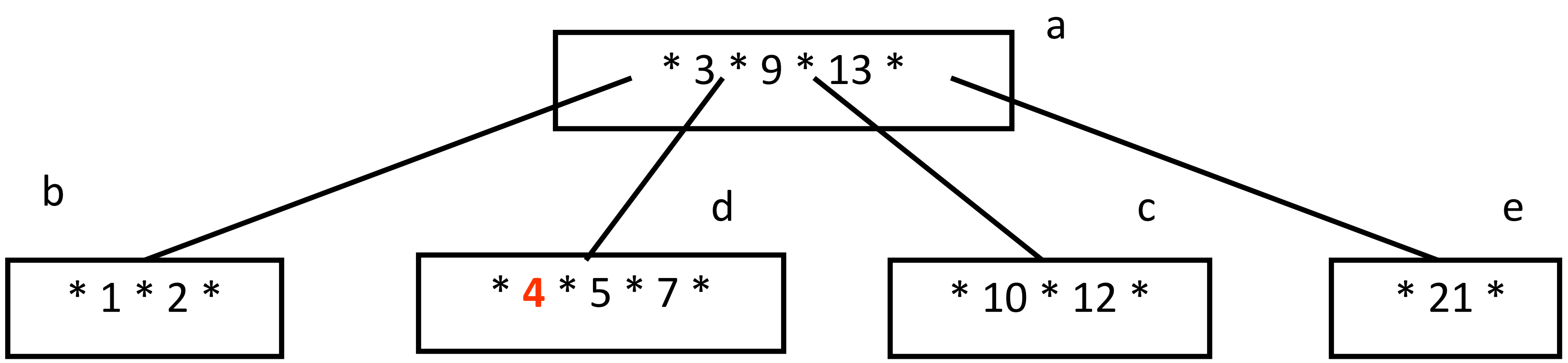
- Insert 12:



Nodes c must split into nodes c and e

Physical DB Design

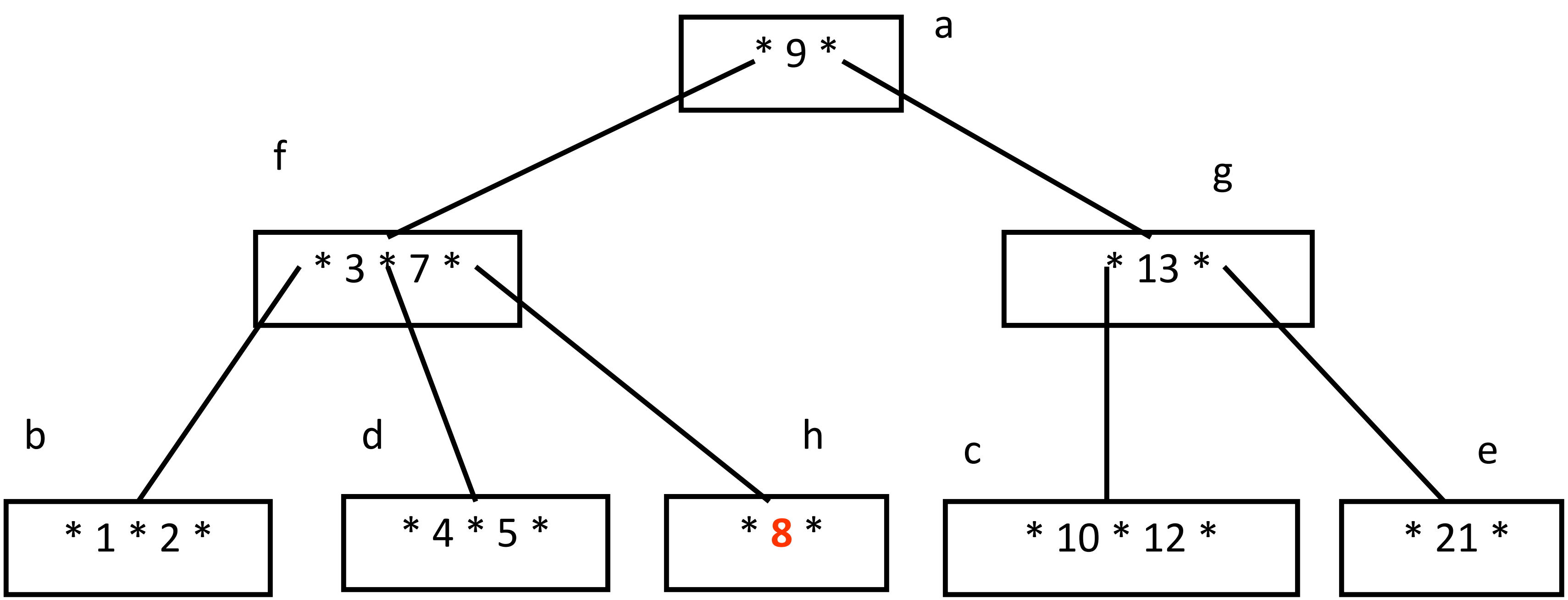
- Insert 4:



Node d has room for another element

Physical DB Design

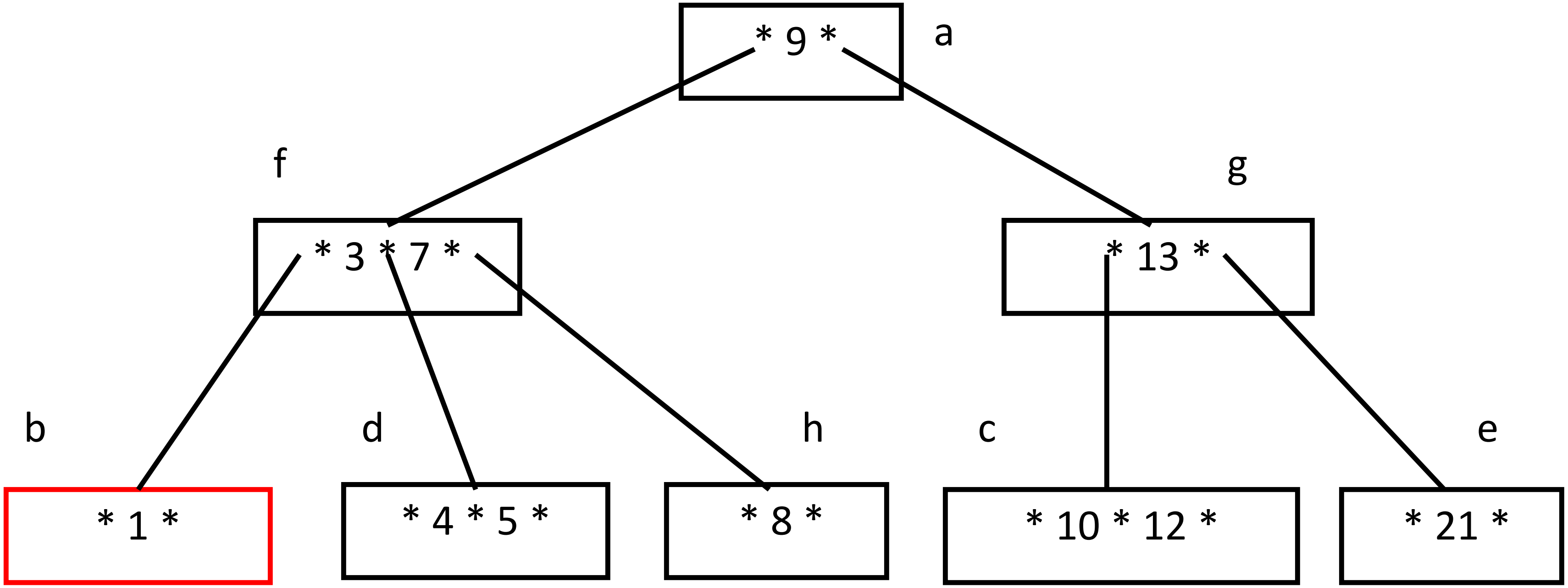
- Insert 8:



Node d must split into 2 nodes. This causes node a to split into 2 nodes and the tree grows a level.

Physical DB Design

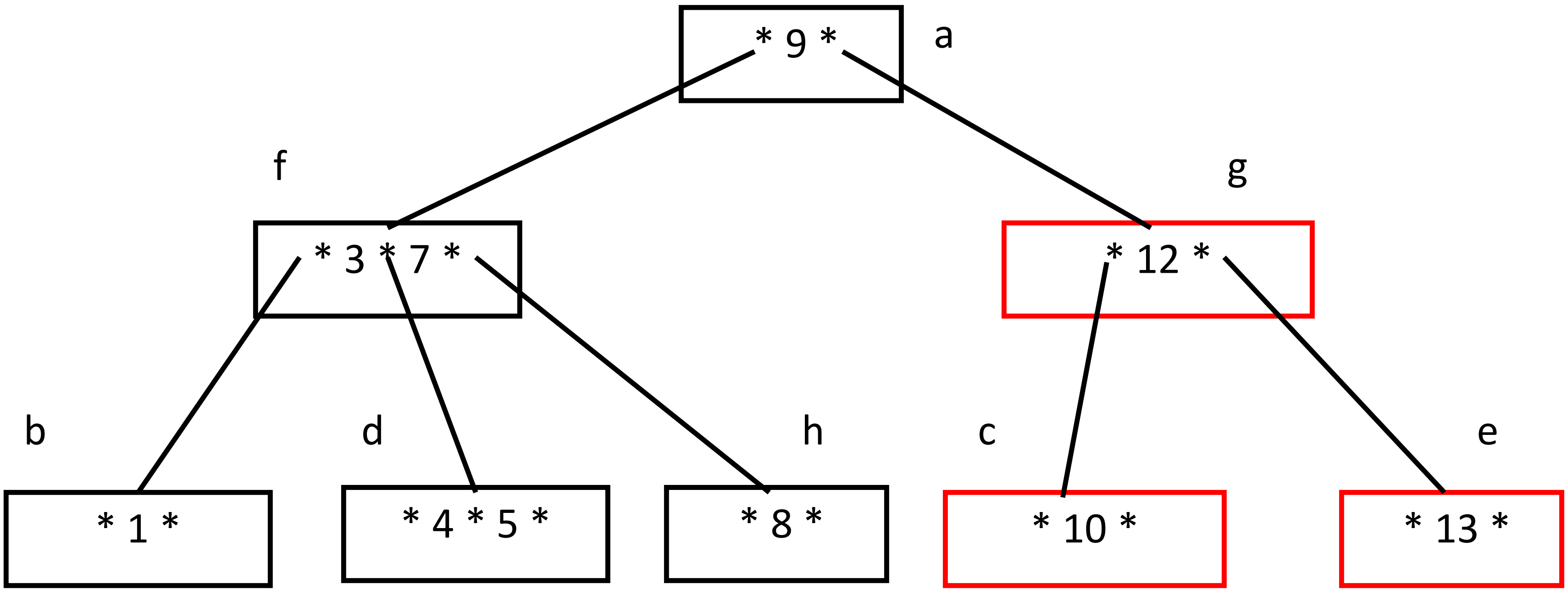
- Delete 2:



Node b can loose an element without underflow.

Physical DB Design

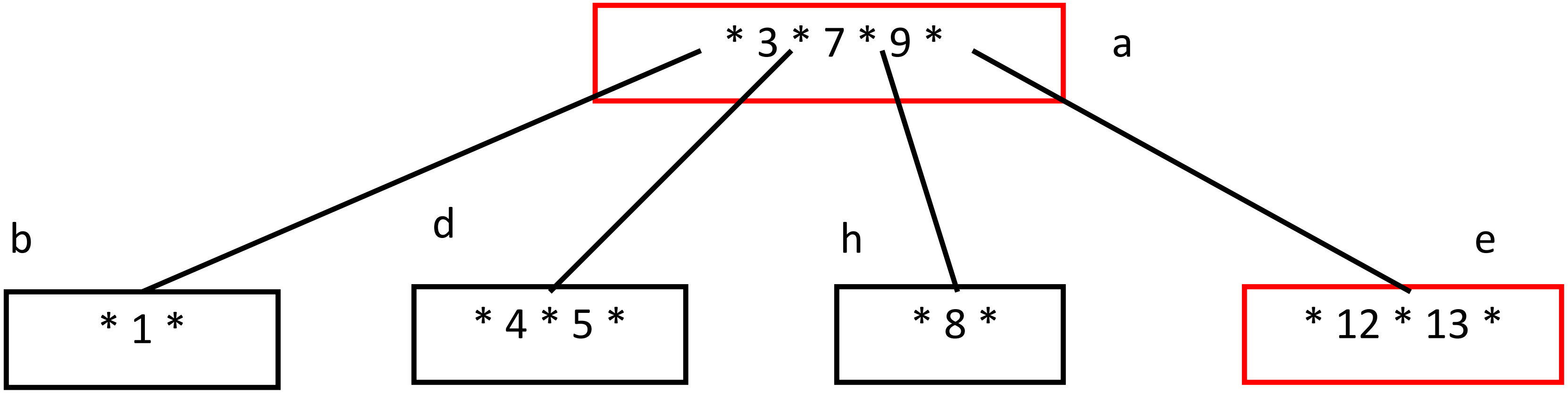
- Delete 21:



Deleting 21 causes node e to underflow, so elements are redistributed between nodes c, g, and e

Physical DB Design

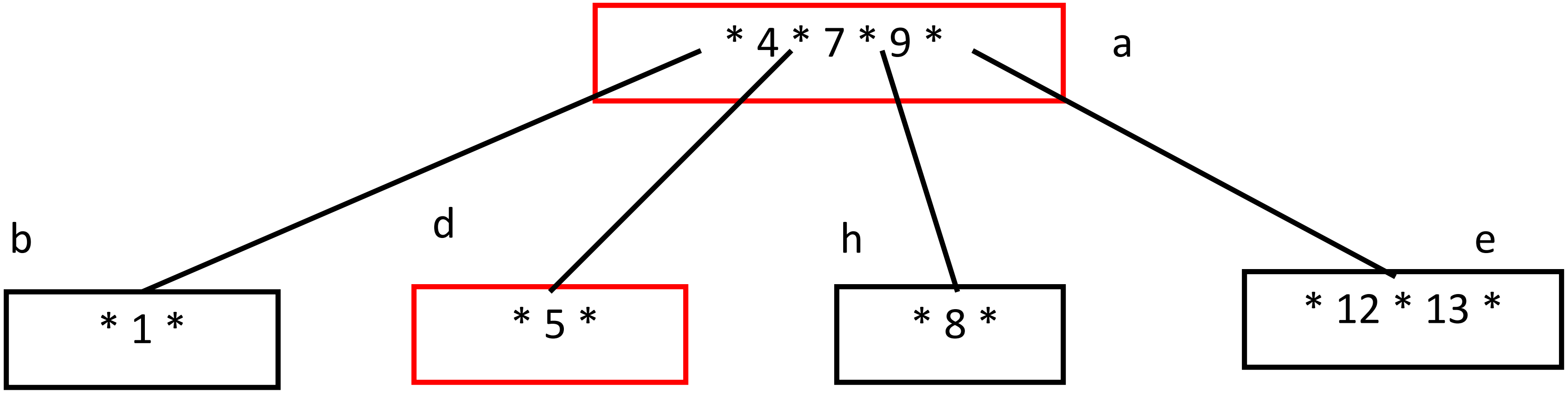
- Delete 10:



Deleting 10 causes node c to underflow. This causes the parent, node g to recombine with nodes f and a. This causes the tree to shrink one level.

Physical DB Design

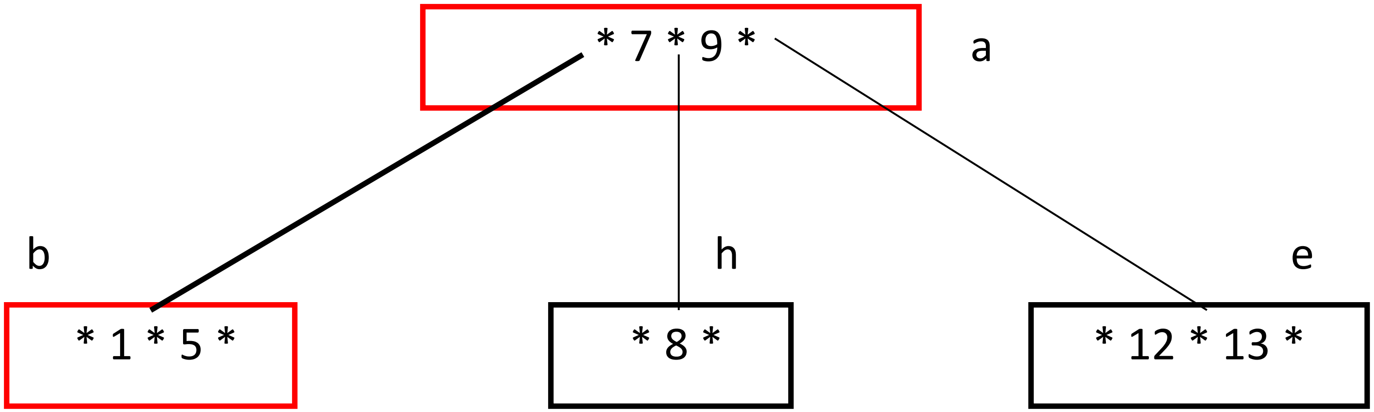
- Delete 3:



Because 3 is a pointer to nodes below it, deleting 3 requires keys to be redistributed between nodes a and d.

Physical DB Design

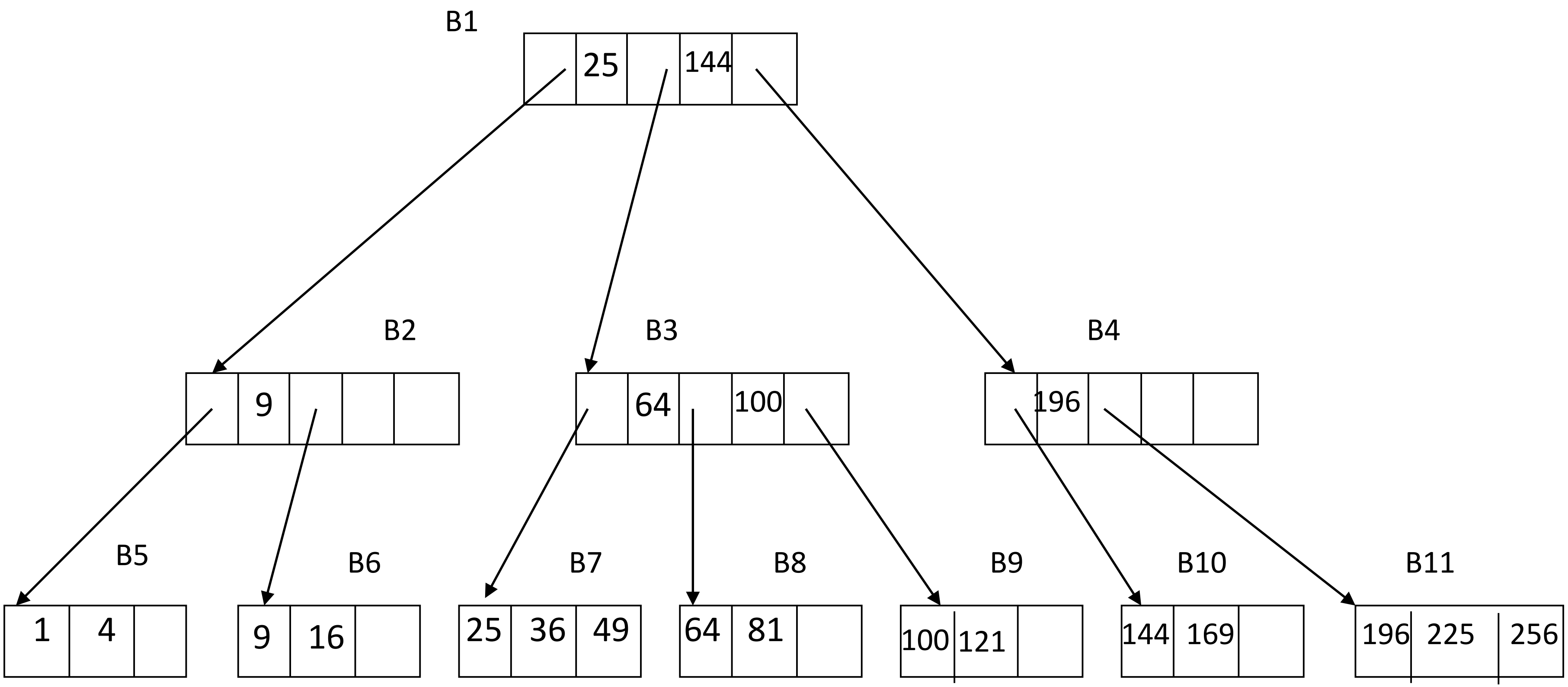
- Delete 4 :



Deleting 4 requires a redistribution of the keys in the subtrees of 4; however, nodes b and d do not have enough keys to redistribute without causing an underflow. Thus, nodes b and d must be combined.

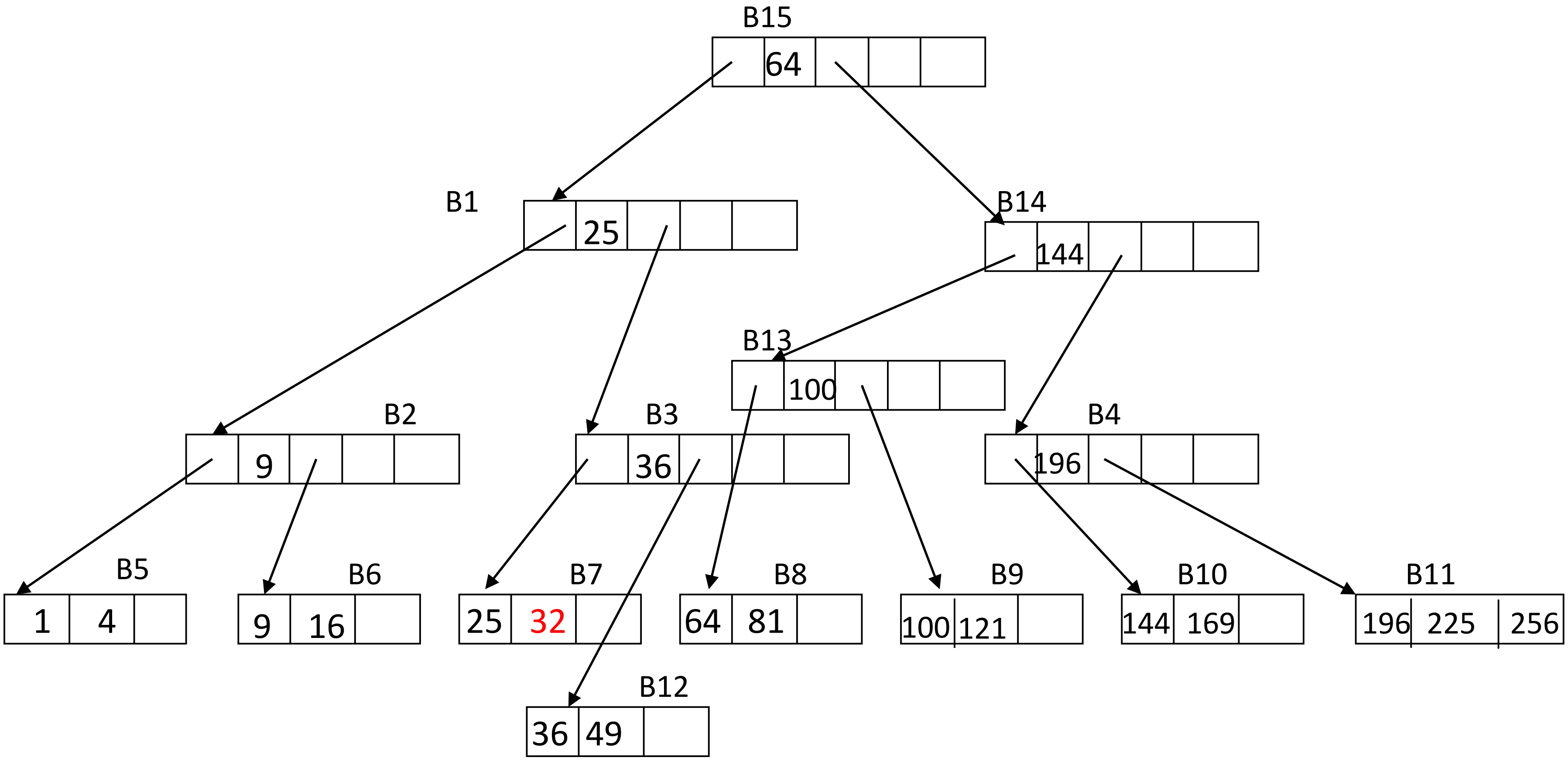
Physical DB Design

Example (2): B-tree



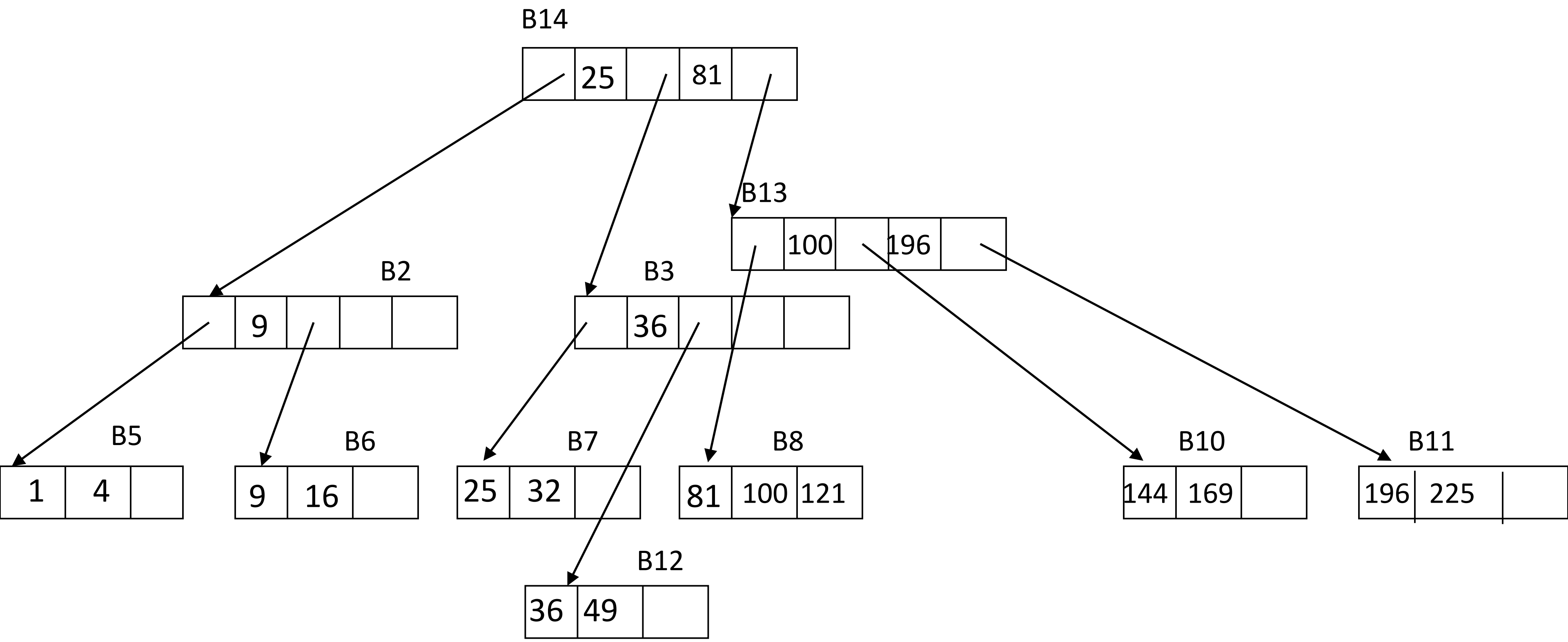
Physical DB Design

- Insert Record with Key 32:



Physical DB Design

- Delete record with key 64:



Physical DB Design

Cost of Operations on B-trees:

- Lookup: if there exist i nodes on a path from the root to a leaf node where a particular record is located, then i block accesses are needed.
- For insertion, deletion and modification, $2 + \log_d(n/e)$ accesses are required on average
- We will assume that all operations take $2 + \log_d(n/e)$ block accesses on average.

End of Slides