Physical DB Design

- Data Structures for Primary Indices
  - Structures that determine the location of the records of a file
  - A primary index is based on a key; the location of a record is determined by its key value.
  - Most common structures: heaps, hashed files, indexed files
- The Heap Organization
  - Trivial organization: records are packed into blocks in no special order and with no special organization of the blocks.
  - It is assumed that there exist pointers to every block of the file and these pointers are stored in main memory (except in the case that these pointers are too many to fit in main memory; then they are also stored in blocks in secondary storage)

Physical DB Design

- Basic Operations on Heaps:
  - Lookup: given a key find the record(s) with that value in their key fields.
  - Insertion: add a record to a file. Assume we know the record does not already exist (or we simply don’t care). If we want to avoid duplicates, insertion must be preceded by a lookup.
  - Deletion: delete a record from a file. Assume it is not known whether the record exists in the file, i.e., deletion includes lookup.
  - Modification: change the values in one or more fields of the record. Modification also includes a lookup.
- The time to perform operations will be measured as the number of blocks that must be retrieved. We assume that, initially, the entire file is in secondary storage.

Physical DB Design

- Efficiency of Heaps
  - Let n be the number of records we need to store and R is the number of records that can fit in a block.
  - (If records are pinned and deleted records cannot have their space reused, n is the number of records that have ever existed. Otherwise, n is the number of records currently in the file.)
  - If records are of variable length, R is taken to be the average number of records that can fit in one block.
  - The minimum number of blocks needed to store these records is $\lceil n/R \rceil$.

Physical DB Design

- Efficiency of Heaps
  1. Lookup: must retrieve n/2R blocks on average. If there is no record with they key value, we must retrieve all n/R records.
  2. 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.
  3. 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. If records are pinned, deletion is performed by setting the deleted bit.
  4. Modification: requires n/2R block accesses to find the record and 1 more to write the new values.

Physical DB Design

- Example: A file contains 1,000,000 records of 200 bytes each. Blocks are 2^{12}=4096 bytes long. R=20.
  - A successful lookup requires n/2R=25,000 accesses; an unsuccessful one requires 50,000 accesses.
  - Assuming that the retrieval of a block from disk takes .01 sec, a successful lookup takes more than 4 minutes.
  - Insertion only can be performed in fractions of a second.
  - The directory of blocks is quite large as well: if a block address is 4 bytes long, 200,000 bytes (50 blocks) are needed for storing the block addresses.

Physical DB Design

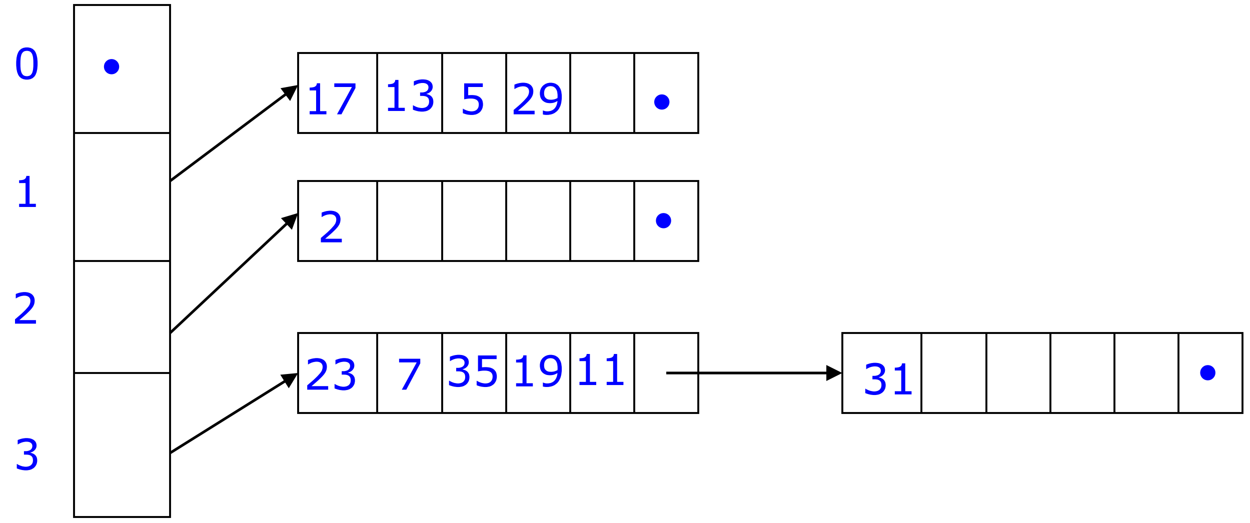
- Hashed Files
  - Records are divided into buckets according to the value of the key.
  - A hash function h takes as argument a value for the key and produces an integer in the range 0 to B-1, where B is the number of buckets.
  - Each bucket consists of a (usually small) number of blocks. The blocks 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 Functions
  - A hash function's range should be the set {0,1,…,B-1}. Moreover, it should take each value in the set with roughly equal probability.
  - A simple hash function :
    - Convert each key value into an integer and then take the remainder of that integer modulo B.
  - If the key value v is an integer, then h(v) = v mod B
  - If the key value is a character string, convert it into an integer by treating the bytes representing the characters as integers and taking the sum of these integers.

Physical DB Design

- Example: a file of numbers organized in a hashed file with 4 buckets. The hash function is h(v) = v mod 4.



- All numbers stored are prime. Hence, no number appears in bucket 0 and only number 2 appears in bucket 2.

Physical DB Design

- Operations on Hashed Files
  1. Lookup a record with key value v:

     compute h(v) and find the bucket header with that value; examine the list of blocks in the bucket, as if the bucket were a heap; if the desired record is not found there, there is no point in searching other buckets.
  2. Insert a record with key value v:

     compute h(v) and find the bucket header with that value; locate the last block of the bucket (either by traversing the list of blocks or by following a pointer to the last block); if there is space on the last block, the record is placed there; otherwise a new block is linked ot the end.

Physical DB Design

- Operations on Hashed Files

3. Delete a record with key value v:

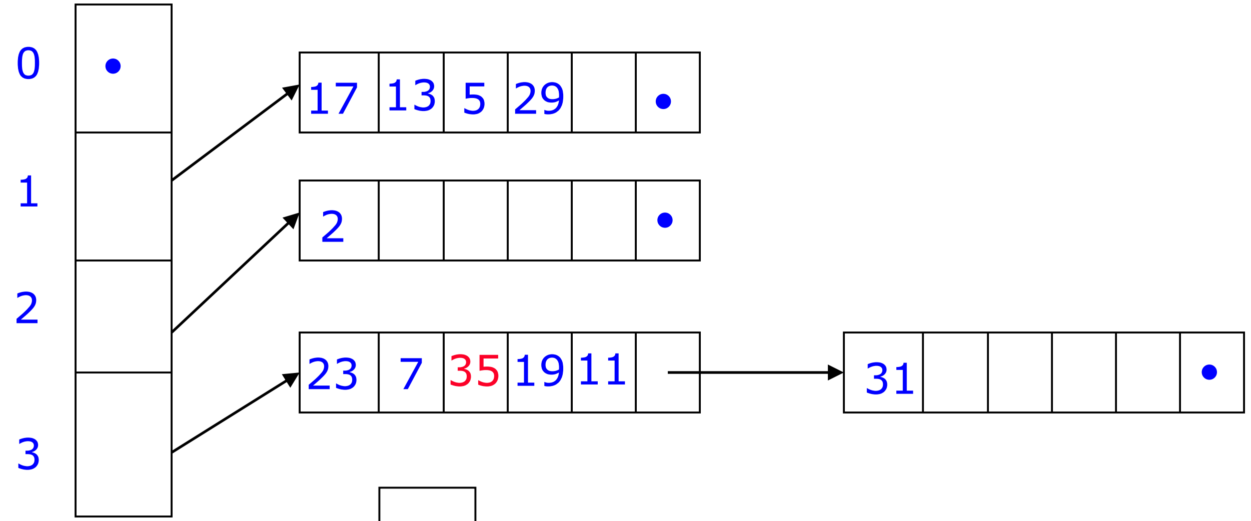
   locate the record to be deleted as in the lookup operation; if the records are pinned, set the deleted bit in the record; if they are unpinned, records may be compacted.

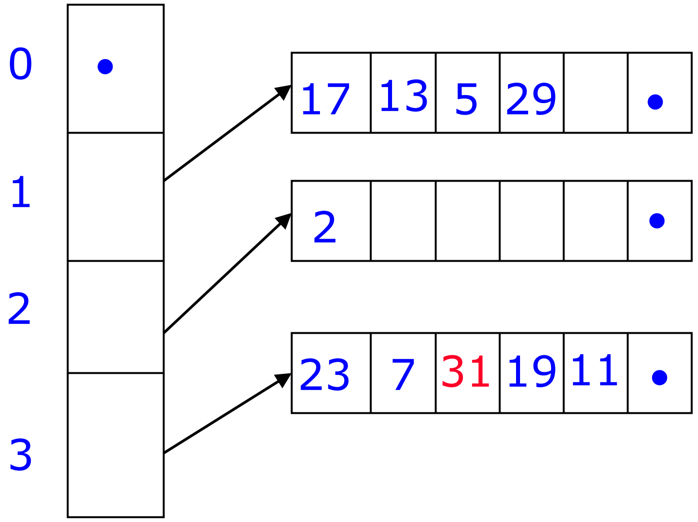
4. Modify a record with key value v:

   locate the record to be deleted as in the lookup operation; change the field(s) in the record; if the record is of variable length, it may have to be moved among the blocks in the bucket

Physical DB Design

- Example: delete record with key value 35





Physical DB Design

- Efficiency of Hashing
  - A hashed file with B buckets behaves as if it were a heap approximately B times shorter. Thus, we can speed-up operations by a factor of B.
  - 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 4000 bytes.
  - An unsuccessful lookup takes 50 block accesses.
  - A successful lookup requires 26 block accesses on the average.

Physical DB Design

- Indexed Sequential Access Method (ISAM)
  - File records are assumed to have unique keys.
  - File records must be kept sorted according to their key values.
- Sorting Keys
  - Independently of their domains, keys can be compared
  - Numerical order is used to compare integers or reals
  - Lexicographic order is used for character strings
  - To sort keys containing more than one field, an order must be imposed on the keys; then records are sorted according to the value of the first field, forming sequences of clusters.
  - Each cluster consists of records with the same value in the first field; clusters are sorted by the value of the second field, etc.

Physical DB Design

- Example: assume record keys consist of two integer-valued fields. Sorting the list of key values (2,3), (1,2), (2,2), (3,1), (1,3) results in the list (1,2),(1,3),(2,2),(2,3),(3,1).
- Accessing Sorted Files
  - The knowledge of the order of records in a sorted file can be exploited for providing efficient access to the file records.
  - 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. If b is the first block, v=-∞ is used.

Physical DB Design

- Accessing Sorted Files
  - The key for the index file is the first field of (v,b) and it is kept sorted according to this field.
  - Index files differ from general files in that index file records are not pinned by pointers from elsewhere.
  - Index files must be organized so that queries of the following sort can be answered efficiently:

    Given a key value v1 for the file being indexed, find that record (v2,b) in the index such that v2 <=v1 and either (v2,b) is the last record in the index, or the next record (v3,b') in the index has v1<v3. Value v2 is said to cover v1.
  - The result of the query is the block b that contains the record with key value v1, since the index is sorted at all times.

Physical DB Design

- Note: certain file organizations cannot provide such functionality. E.g., hashed files cannot be used, since the entire file must be searched in order to find the required values.
- 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
     - Linear search of the index is preferable over linear search of the file: if R records are on a block, the main file has R times as many records as the index file.
     - 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. 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 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

- Operations on Sorted Files with Unpinned Records
  - The operations insertion, deletion and modification require insertions, deletions and modifications to the index file.
  - Assumptions:
    - The original sorted file is kept on the sequence of blocks B_1, B_2, …B_k.
    - The records in each block are kept in sorted order.
    - The records of block B_i precede those of block B_{i+1}
    - Used/unused information is kept in a known area in the beginning of the file.

Physical DB Design

- Initialization:
  1. The initial file of records must be sorted and distributed among blocks.
  2. Create the index file by examining the first record in each block of the main file. Form the records of the index file by combining the key values of the file records with the block addresses.
  3. Distribute the index file records among blocks.
  4. Create a directory containing the addresses of the index blocks.

Physical DB Design

- Operations:
  1. Lookup: find the record with key value v1

     examine the index to find the record with a value v2 that covers v1. The index record containing v2 also contains a pointer to a block of the main file. If the record with key value v1 exists, it will be on that block.
  2. Modification: modify the record with key value v1

     use the lookup procedure to find the record. If the modification changes the key, treat the operation as a deletion followed by an insertion. If not, make the modification and rewrite the record.

Physical DB Design

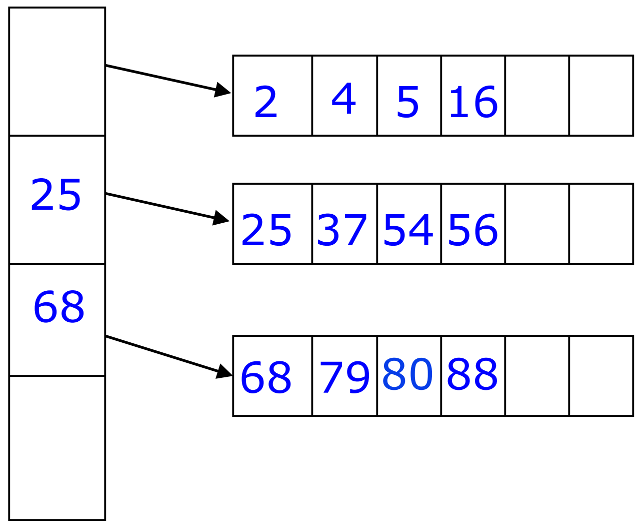
- Operations:
  1. Insertion: insert a record with key value v1

     use the lookup procedure to find the block Bi of the main file, on which a record with key value v1 would be found. Place the new record in Bi keeping the records sorted. If Bi does not have space for the new record, a new block must be created. One option is to use the next block (if it has space). Then the new record must become the block's first record.
  2. Deletion: delete the record with key value v1

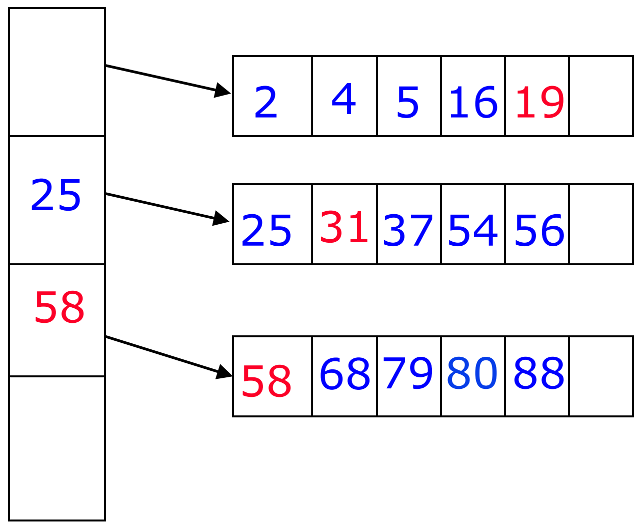
     use the lookup procedure to find the record. Shift the records that are located to the right of the deleted record one position to the left. If the block becomes empty after the deletion, the record for the block must be deleted from the index.

Physical DB Design

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

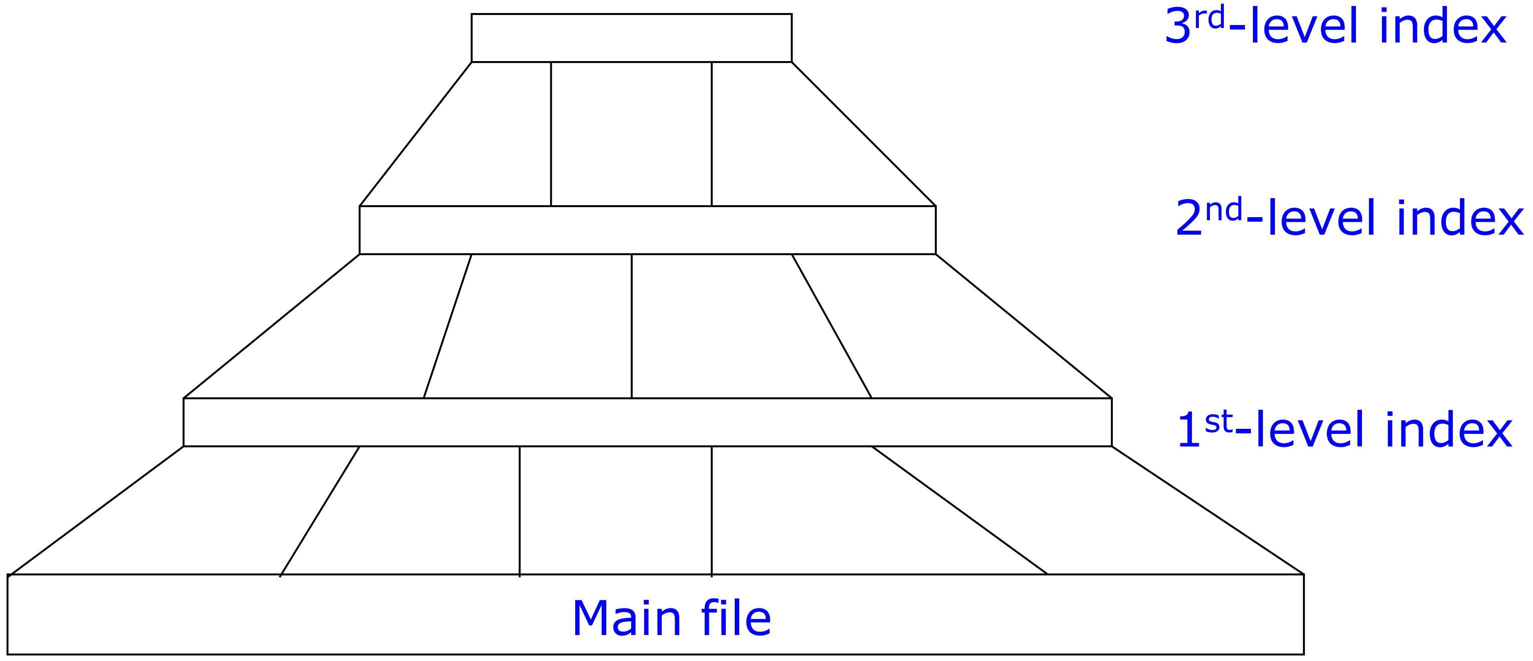


After insertion of numbers 19,58,31:



Physical DB Design

- B-Trees
  - Index files can become quite large for large main files
  - Indices on index files are possible



Physical DB Design

- The 1st-level index consists of pairs (v,b), where b is a pointer to a block B of the main file and v is the key of the first record in the block. The index is also sorted by key values.
- The 2nd-level index consists of pairs (v,b) where b points to a block of the 1st-level index whose first key is v, and so on.
- Multilevel indexing can be considerably more efficient than a single level of indexing.
- A multilevel index structure can have many forms. These are collectively referred to as balanced trees (B-trees).
- The main file is part of the B-tree and it is assumed that the file contains unpinned records.

Physical DB Design

- Although the insertion / deletion procedures of single-level index structures can be used, they do not result in the optimum organization of the B-tree: nodes can have between one and the maximum possible number of records.
- B-trees use a particular insertion / deletion strategy that ensures that no node, except possibly the root, is less than half full.
- A B-tree is characterized by two parameters d,e:
  - d and e are integers such that, the number of index records a block can hold is 2*d-1 and the number of records of the main file a block can hold is 2*e-1.
- Convention: the key value in the first record is omitted. It is assumed that all values that are less than the key value of the second record are covered by the missing value.

Physical DB Design

- Operations on B-trees:
  1. Lookup: to 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
  - Paths in B-trees: every search path begins at the root. When a block B is reached during the search, if B is a leaf node, then B has to be examined for a record with key value v. If B is not a leaf, then it is an index record. The key value that covers v has to be found and the associated pointer must be followed, leading to another index block or main file block.
  - Property: the key value in record i of block B is the lowest key value of any leaf descending from the i-th child of B.

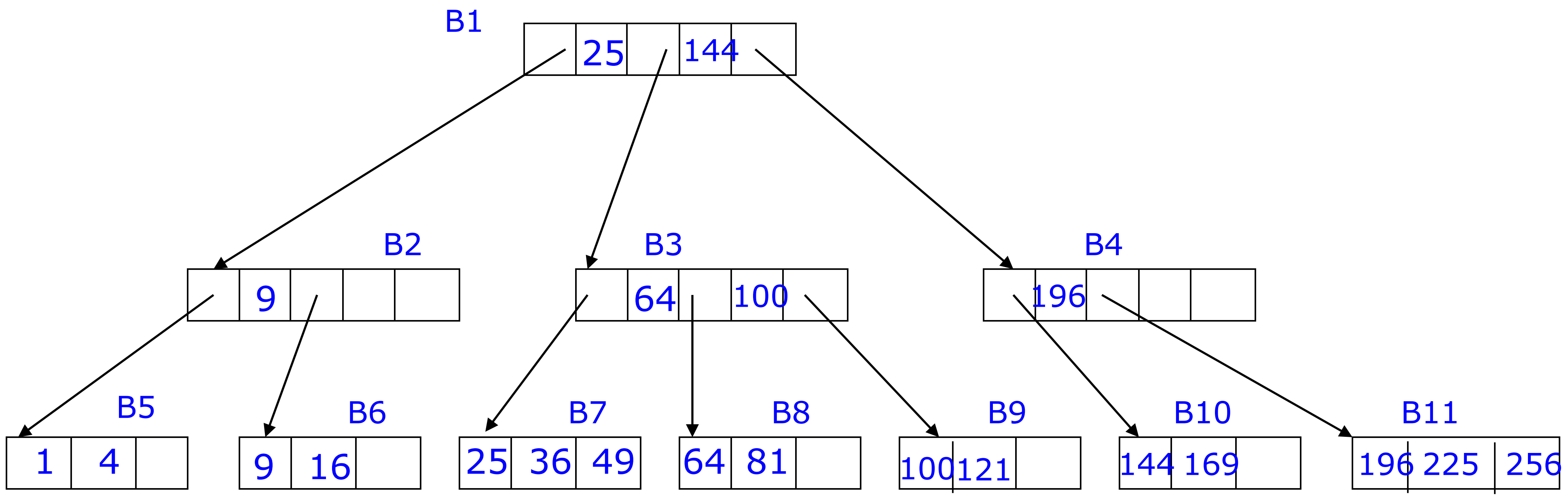
Physical DB Design

- This property and the fact that the main file is sorted, guarantee that, if a record exists in a leaf node, then it can only be found by following the pointers as described above.

2. Modification:
   - If a key field is to be modified, then modification is performed by a deletion followed by an insertion.
   - Otherwise, modification is a lookup followed by the rewriting of the record involved.

Physical DB Design

- Example: d=e=2, i.e., 3 records in blocks of the index and main files.



Physical DB Design

3. Insertion: to insert a record with key value v
   - Apply the lookup procedure to find the block B in which this record belongs
   - If there are fewer than 2e-1 records in B, insert the new record in sorted order
   - If there are already 2e-1 records in B, create a new block B1 and divide the records of B and the inserted record in two groups of e records each. The first e go to block B and the remaining e to block B1.
   - A record for B1 needs to be inserted in the parent record of B. The insertion procedure is applied recursively (with d in place of e) for inserting a record for B1 to the right of the record for B.

Physical DB Design

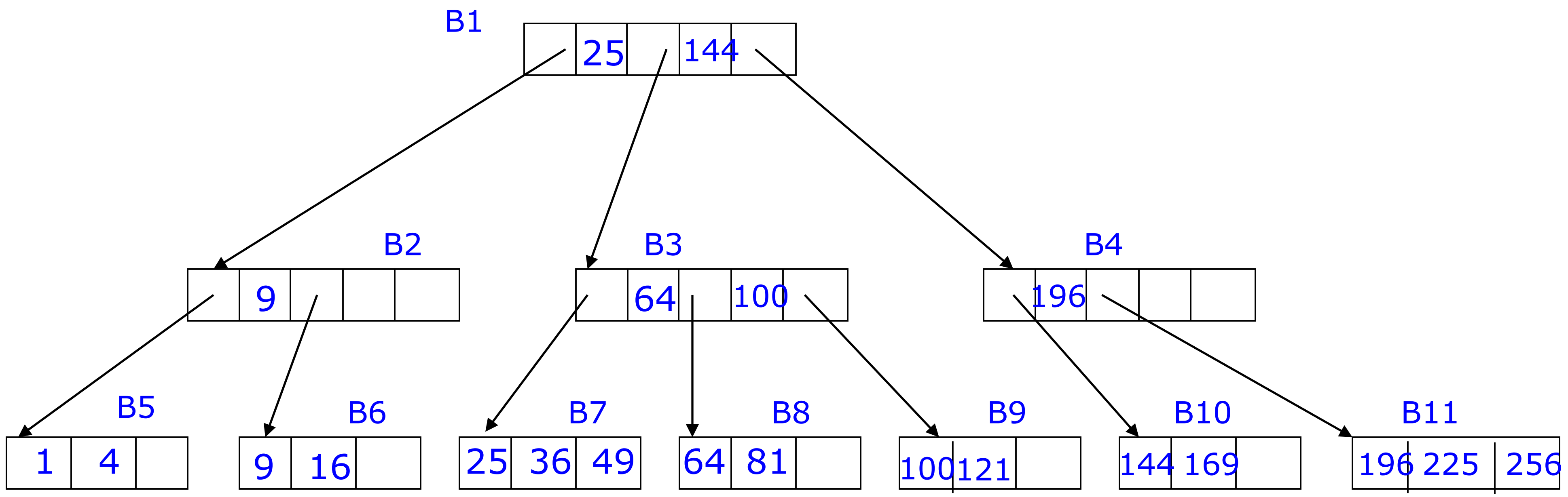
3. Insertion: to insert a record with key value v
   - If many ancestors of B have the maximum 2d-1 records, the effects of inserting a record may ripple up the tree. It is only ancestors of B that are affected.
   - If the insertion ripples up to the root, then the root node is split and a new node with two children is created.
   - Example: assume we want to insert the record with key value 32 in the B-tree of the previous example

     Record 32 belongs to B7, but B7 is full. A new block (B12) is created and a record for B12 must be inserted in B3.

     B3 is also full, so a new block (B13) is created.

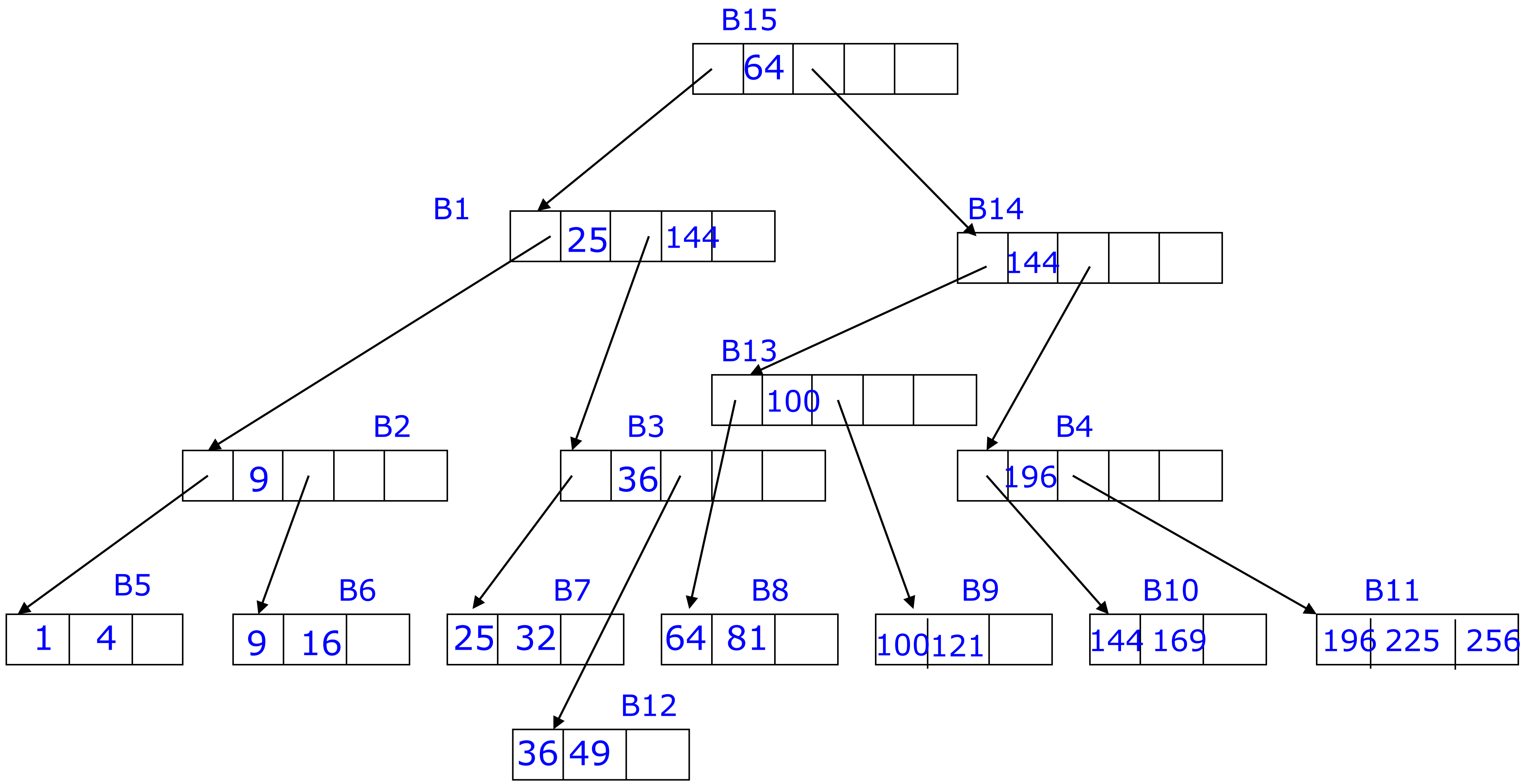
Physical DB Design

- Example: d=e=2, i.e., 3 records in blocks of the index and main files.



Record 32 belongs to B7, but B7 is full; a new block (B12) must be created and a record must be inserted in B3. But B3 is also full.

Physical DB Design



Physical DB Design

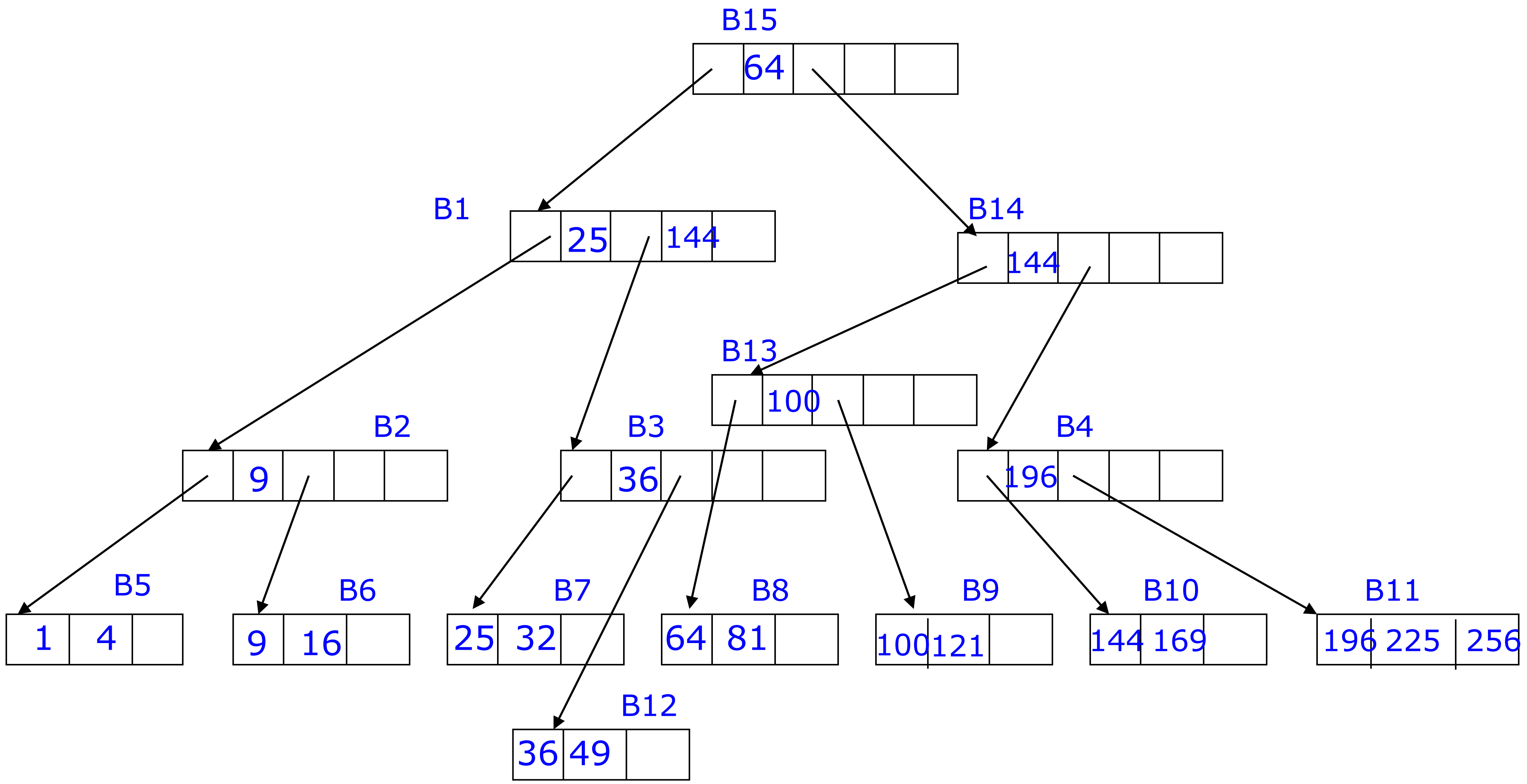
4. Deletion: to delete a record with key value v
   - Apply the lookup procedure to find the block B in which this record belongs
   - If after the deletion, B still has e or more records, we're done.
   - If the deleted record was the first in B, the value of the parent record must be changed to contain the value of the new first key of B.
   - If B is the first child of its parent node, the parent has no key value for B, so the parent's parent must be changed. The process continues until an ancestor A1 of B is found that is not the first child of its parent A2. Then, the new lowest value of B goes in the record of A2 that points to A1.

Physical DB Design

4. Deletion: to delete a record with key value v
   - If after deletion block B has e-1 records, we examine the block B1 that has the same parent as B and that is immediately next to B, If B1 has more than e records, we distribute the records of B and B1 as evenly as possible, keeping them sorted.
   - The key values in the parents of B and B1 may need to be modified.
   - If B1 has only e records, then combine the records of B and B1. This requires that a record be deleted from the parent node.

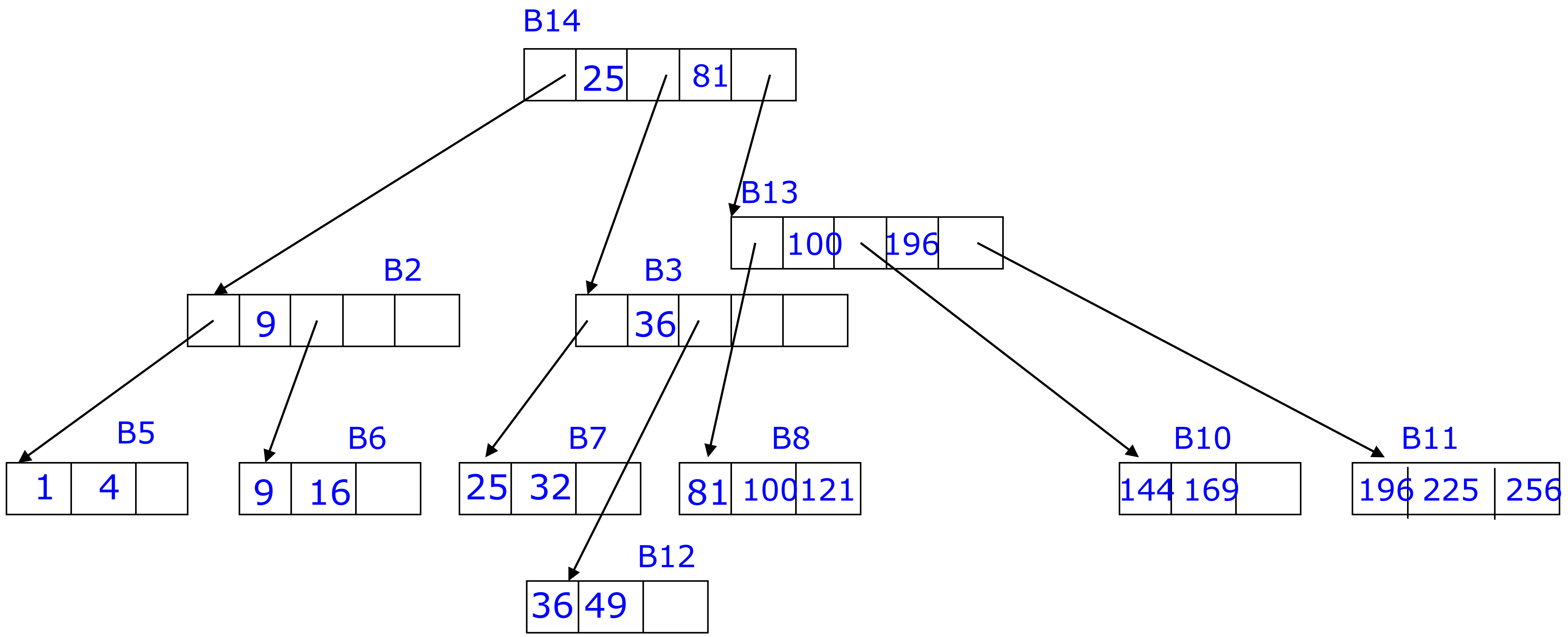
- Example: delete the record with key value 64 in the B-tree of the previous example.

Physical DB Design



Physical DB Design

B-tree after the deletion of record 64



Physical DB Design

- Cost of Operations on B-trees:
  - Assume that a file of n records is organized as a B-tree with parameters d and e. Then the tree will have at most n/e leaf nodes and n/de parent nodes of leaf nodes, n/d^2e parents of parents of leaf nodes, etc.
  - 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.

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. Pointers take 4 bytes.
  - e=10 (up to 20 records can fit in a block)
  - 171 index records can fit in a block (171*20 + 171*4 = 4084). Thus, d=86.
  - The expected number of block accessed per operation is $2 + \log_d (n/e) = 2 + \log_{86} (1000000 / 10) < 5$