Physical Data Organization

- Databases for modern applications are too large to fit in main memory
- The efficiency of operations on databases residing in secondary storage relies heavily on the availability of good storage organization techniques
- Performance measures include the time required to perform database operations such as selections or joins
    - E.g., selections should take time proportional to the number of tuples retrieved rather than the size of the relation
- Storage organization techniques include:
    - Heaps
    - Hashing
    - Indexed sequential access methods
    - B-trees, B$^+$-trees

The Physical Data Model

- **Records, Fields and Files**
    - At the conceptual or logical level, a database is a collection of entities and relationships, or a collection of tables representing entities and relationships.
    - At the physical level, a database is a stored collection of records, each consisting of one or more fields.
    - Fields contain values of elementary data types (e.g., integer, real)
    - A record is used to physically store each of the basic objects at the conceptual level
    - E.g., a tuple of a relation can be stored as a record with each component of the tuple stored in one field.
    - Records can be viewed as instances of a record scheme.

The Physical Data Model

- A database contains collections of records with the same number of fields, with corresponding fields having the same name and data type.

- The list of field names and their corresponding data types is called the format of the record.

- A file is a collection of records with the same format. For instance, a file physically represents a relation.

- Two-level storage

  - The physical store where records and files reside can be thought of as an array of bytes numbered sequentially.

  - Files normally reside in secondary storage. In order to perform operations on data found in file records, records must be moved from secondary storage to main memory. Once they are moved operations can be performed very fast compared to the speed of data transfer between main and secondary storage.

The Physical Data Model

- Blocks
  - Secondary storage is partitioned in segments (blocks) of a substantial number of bytes (2^9-2^{12}) with several records in a block
  - Transfer of data occurs in units of a block.
  - The cost of database operations depends on the number of blocks moved between main memory and secondary storage.
  - The efficiency of operations improves when records of a file lie within the same block or a relatively small number of blocks.
- Cost of DB Access
  - The unit cost of database operations is the block access, i.e., the time required for reading from or writing to a single block.
  - This cost of performing main memory operations is negligible compared to the cost of block transfer.

The Physical Data Model

- ■ With the buffering of blocks in main memory we can avoid having to transfer a block from secondary storage to main memory.

- ■ The time to access a block in secondary storage depends on the location where the last block access took place.

- Simplifying assumptions:

  - ■ There is a fixed probability that a block will need to be transferred.

  - ■ The cost of a block access does not depend on what accesses were made previously.

  - ■ Each block access costs the same.

The Physical Data Model

- Pointers
  - A pointer to a record r is data sufficient to locate r efficiently.
  - A pointer can be of different types such as, e.g., the absolute address of the beginning of record r.
  - Absolute addresses are often undesirable: we might permit records to move around within a block or group of blocks. If absolute addresses are used as pointers and record r is moved, all pointers to r must be updated.
  - A pair (b,k) can be used as a pointer, where b is the block in which r resides and k is the value of the key for r.

The Physical Data Model

- Pointers
  - To locate a record r given a pair (b,k) it is sufficient to know that:
    - All records in block b have the same format as r; therefore none can agree with r in its key fields.
    - The beginnings of all the records in block b can be found.
    - Each record in block b can be decoded into its field values given the beginning of the record.
- Pinned Records
  - When records may have pointers to them from unknown locations, we say that the records are pinned.
  - If records are unpinned, they can be moved around within a block or from block to block.

The Physical Data Model

- ## Pinned Records

  - When records are pinned, they cannot be moved if the pointers to them are absolute addresses. They can only be moved within the same block if a pair (b,k) is used as a pointer.

  - Records cannot be deleted if they are pinned: if there exists a pointer p to a record r and r is deleted, then if at some later point in time a record r' is put in the same place as r, pointer p will point to r'. Pointer p is called a dangling pointer.

  - Even in the case where block-key pairs are used, the problem cannot be avoided: if r' has the same key as r, we would still have an unintended reference to a record.

  - To avoid this problem, each record includes a bit (called deleted bit) which is set to 1 if the record is deleted. If we reach the record by following a pointer, we will know if it is deleted or not.
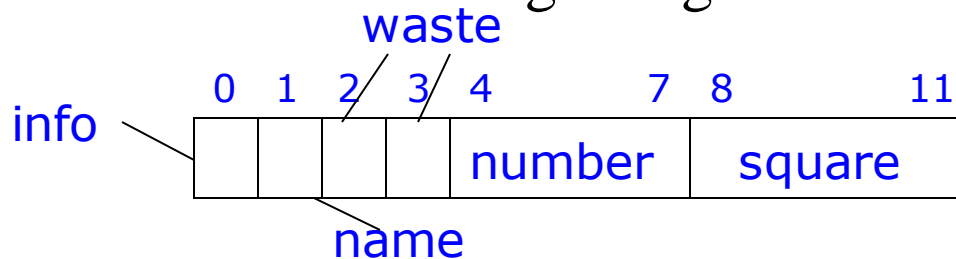
The Physical Data Model

- ## Record Organization
    - The fields in a record must be arranged in such a way that their values can be accessed. If all fields have fixed length, then only an order must be chosen: each field starts at a fixed number of bytes (offset) from the beginning of the record.
    - Once a record is located, the field can be found by moving forward a number of bytes equal to the offset for that field
    - Additional bytes may be required for each record:
        - Some bytes denote the format of the record. For instance if the record belongs to more than one relation, we may wish to store a code indicating the relation of each record.
        - Alternatively, we can store only one type of records in any block and let the block indicate the type of its records.

The Physical Data Model

- **Additional bytes** may be required for each record:
  - One or more bytes may denote the length of the record. If the record involves only fixed-length fields, then the length is implicitly derived.
  - A byte including a deleted bit and a used/unused bit: the latter is needed when blocks are divided in areas, each of which holds a record. This bit indicates if there actually exists a record or if it is empty space.
  - Waste space: useless bytes may be added so that the bytes where the records begin have convenient addresses.

The Physical Data Model

- Example: record type: NUMBER
    - Fields:
        - Number: integer (key field) – always holds a positive integer
        - Name: a single byte indicating the first letter of the English name for the number
        - Square: integer – holds the square of the number
    - If an integer takes 4 bytes, a total of 9 bytes is required for the record. In addition, a deleted bit and a used/unused bit is needed. Waste space is needed if integers are required to be stored in records whose beginning address is a multiple of 4.
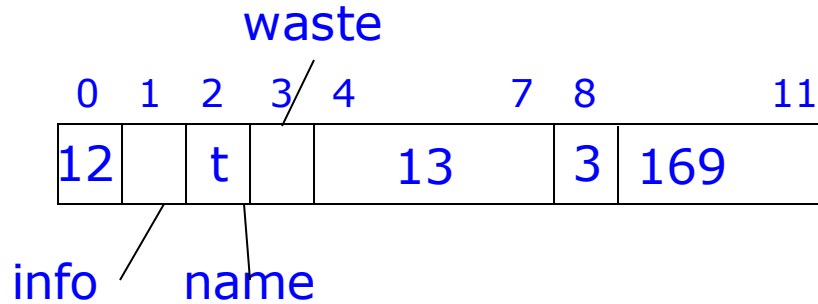
The Physical Data Model

- Variable-length records

  - Record formatting problems occur when fields are allowed to vary in length: fields do not begin at the same offset for every record of a particular format

  - Record formatting strategies:

    - Attach a count at the beginning of each field. Its value is the number of bytes the field occupies. Although redundant, the total length of the entire record is also stored in the beginning of the record.

    - Place, in the beginning of each record, pointers to the beginning of each variable length field and a pointer to the end of the last field. All fixed-length fields precede all variable-length fields.
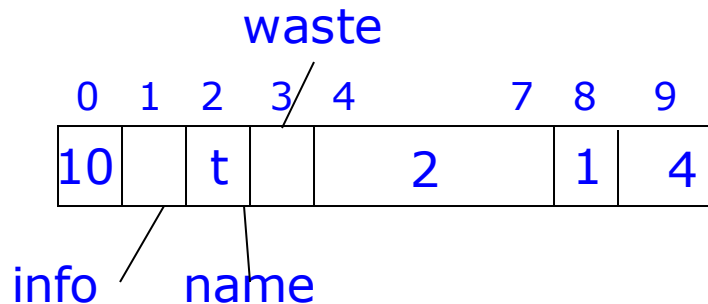
The Physical Data Model

- ■ Comparison:
  - The former strategy uses less space but is less efficient in locating fields: to locate a variable-length field beyond the first, we must examine all previous variable-length fields.
  - With the latter, fields can be located faster but pointers have to be stored with the record.
- ■ Example: assume the square of a number is represented as a character string instead of an integer (variable-length field).

  A count for the length of this field and the total length of the record must be used according to the first strategy.

The Physical Data Model

waste

0	1	2	3	4			7	8			11
12		t			13			3	169		

info   name

Record for number 13

waste

0	1	2	3	4			7	8	9
10		t			2			1	4

info   name

Record for number 2

The value of byte 0 is always 9 more than the value of byte 8. Hence, either one can be dispensed with (but not both).

If there were additional fields following the square of the number, then byte 8 would have to be read before that field could be found.

The Physical Data Model

- Block formats
  - Records within blocks need to be located efficiently as well.
  - Just as records require additional space for formatting information, so do blocks of records. E.g., blocks often have pointers in fixed positions to link blocks in lists of blocks.
  - The formatting of blocks must take into account the alignment requirements of record fields.
    - E.g., if integers within the records are required to start at an address divisible by 4, then we require that the offsets of integers within the records be divisible by 4 and that records begin at some offset that is also divisible by 4. Normally blocks begin at an address that is a multiple of a power of 2.

The Physical Data Model

- If a block contains fixed-length records, then the block has to be partitioned in as many areas, each holding one record, as will fit in a block.

- Space must be reserved for special fields in a known place in each block.

- Example: Assume blocks of length 64. We wish to store records of fixed length representing numbers.
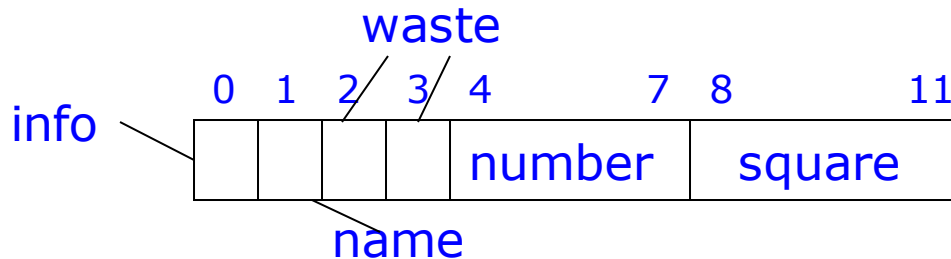
The Physical Data Model

- Moreover, blocks must have a pointer of 4 bytes to be used as a link to another block.

0	11 12	23 24	35 36	47 48	59 60	63
Rec 1	Rec 2	Rec 3	Rec 4	Rec 5	link	
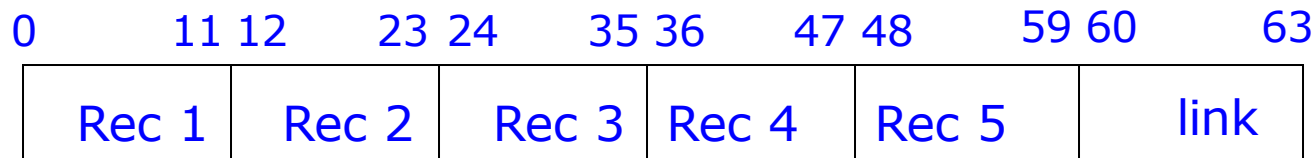
- Every record contains a used/unused bit. To find an empty area in which to insert a record, we must visit each record area in turn.

- Alternatively, all used/unused bits can be grouped in one or more bytes at the beginning of the block.

- For the block shown above, only byte 0 is needed for storing the used/unused bits. Then the block can only contain 4 records because of the alignment requirements. (record length cannot be reduced below 12. Records stored in positions, 4-15,16-27,28-39,40-51. Bytes 60-63 for link. Bytes 52-59 are waste space.)
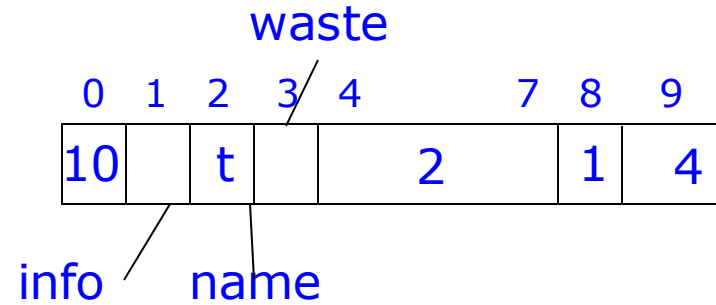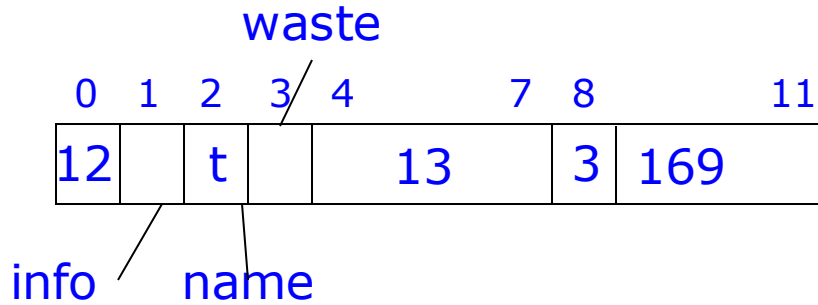
The Physical Data Model

- Blocks with variable length records
  - Locating variable-length records:
    - Assume that the first record starts at byte 0. The length of the record is found there. The beginning of the second record is at the next multiple of 4 (after the number indicating the length of the first record). The third record is found at the next multiple of 4 following the second record, etc.
  - Alternatively, a directory can be placed at the beginning of the block, consisting of an array of pointers to the records in the block. These pointers are actually offsets from the beginning of the block to the location where the particular record begins.
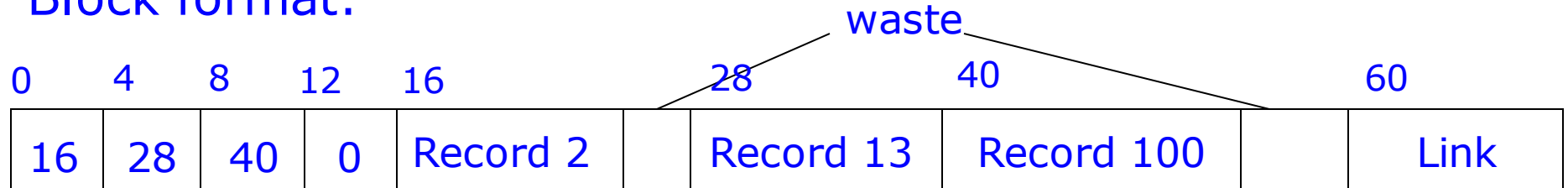
The Physical Data Model

- **Directory Representation**

  1. Precede the directory by a byte denoting the number of pointers in the directory

  2. Use a fixed number of fields at the beginning of the block for pointers to records. If there are fewer records in the block than fields for pointers, these fields will contain 0.

  3. Use a variable number of fields for pointers to records, with the last such field containing 0.

- **Example:** Suppose we want to store variable-length records of the following format:

The Physical Data Model

waste

0	1	2	3	4	7	8	11
12		t		13		3	169

info   name

waste

0	1	2	3	4	7	8	9
10		t		2		1	4

info   name

Block format:

waste

0	4	8	12	16	28	40	60
16	28	40	0	Record 2	Record 13	Record 100	Link

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.
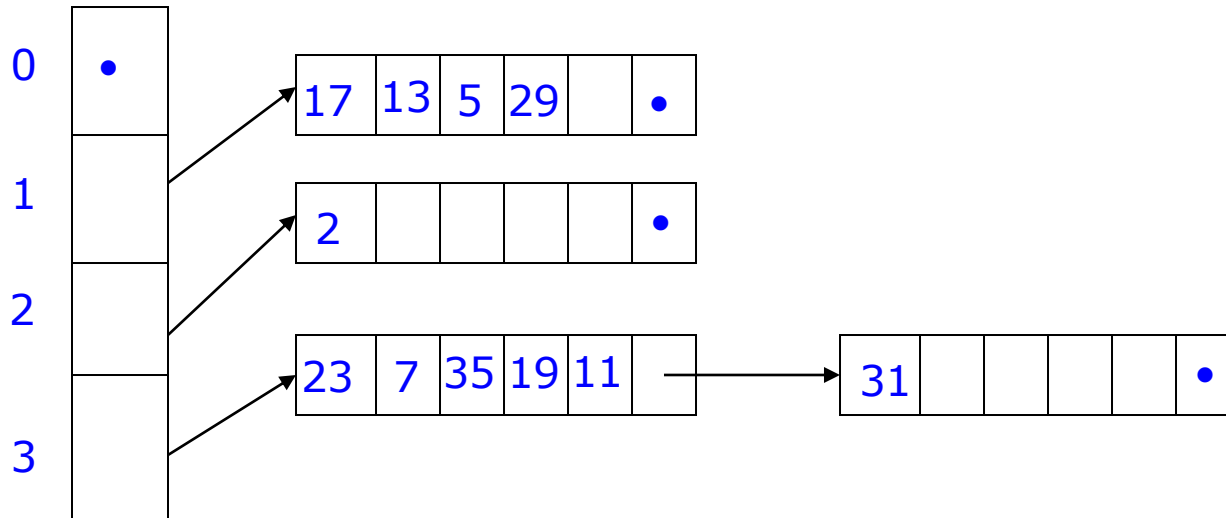
Physical DB Design

- Hash Functions

  - A hash function's range should be the set $\{0,1,\ldots,B-1\}$. 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 $h(v) = v \bmod B$

  - 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 modified 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
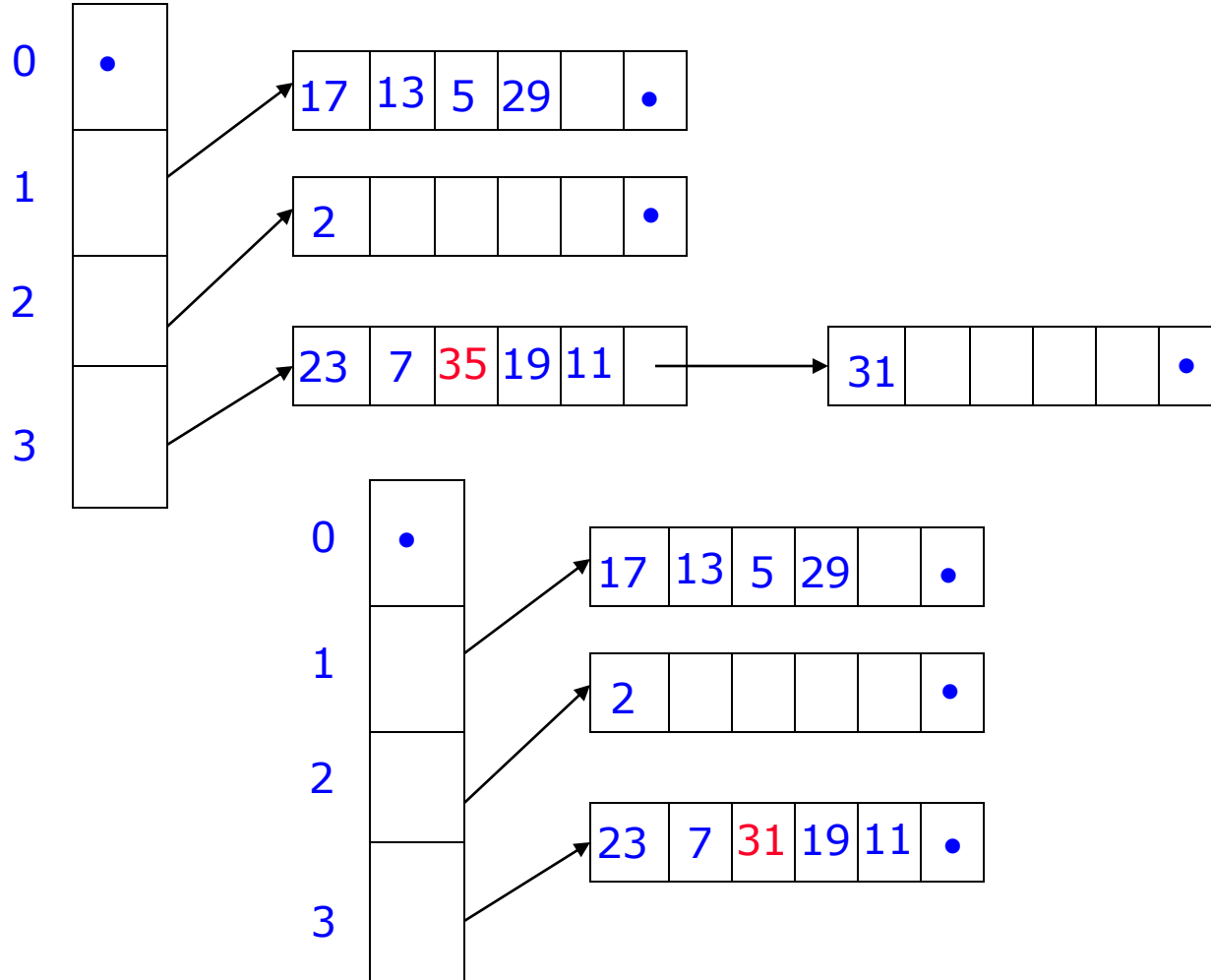
Physical DB Design

- **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.