Physical File Organization and Indexing

ACS 575: Database Systems

Instructor: Dr. Jin Soung Yoo
Department of Computer Science
Purdue University Fort Wayne

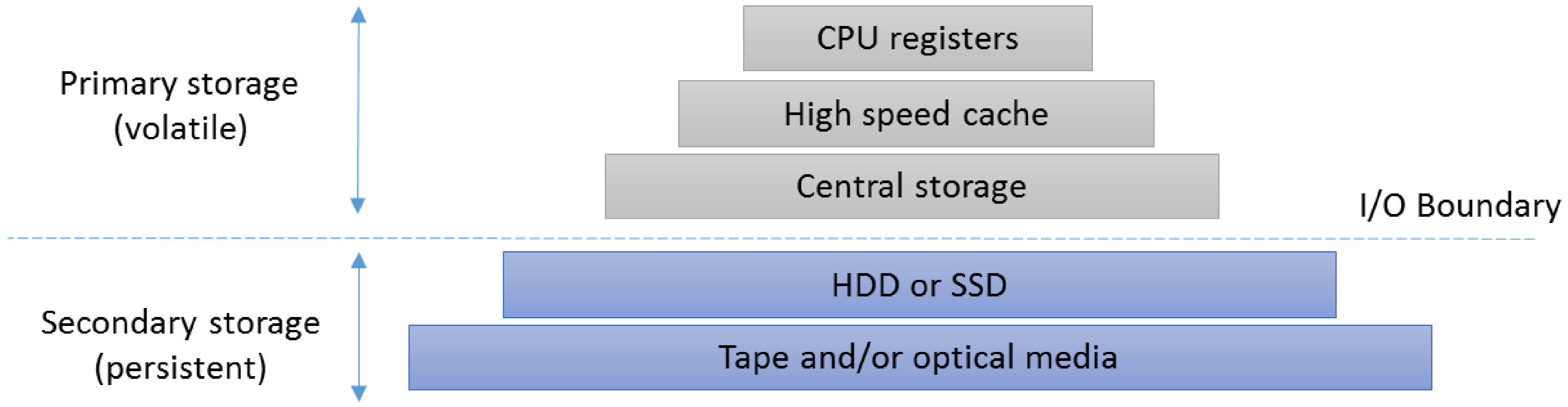
References

- W. Lemanhieu, et al., Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data, Ch 12

Outline

- Storage Hardware and Physical Databases
- Record Organization
- File Organization

Storage Hierarchy


Primary storage (volatile)
CPU registers
High speed cache
Central storage
I/O Boundary
Secondary storage (persistent)
HDD or SSD
Tape and/or optical media

- **Primary storage**
  - high speed memory, expensive and limited in capacity
  - CPU registers, cache memory, central storage (a.k.a main memory)
  - **Volatile memory**
- **Secondary storage**
  - consists of **persistent storage media**
  - slower memory, relatively cheap and larger in size
  - hard disk drives (HDD), solid-state drives (SSD) based on flash memory
  - Tertiary storage: Removable media

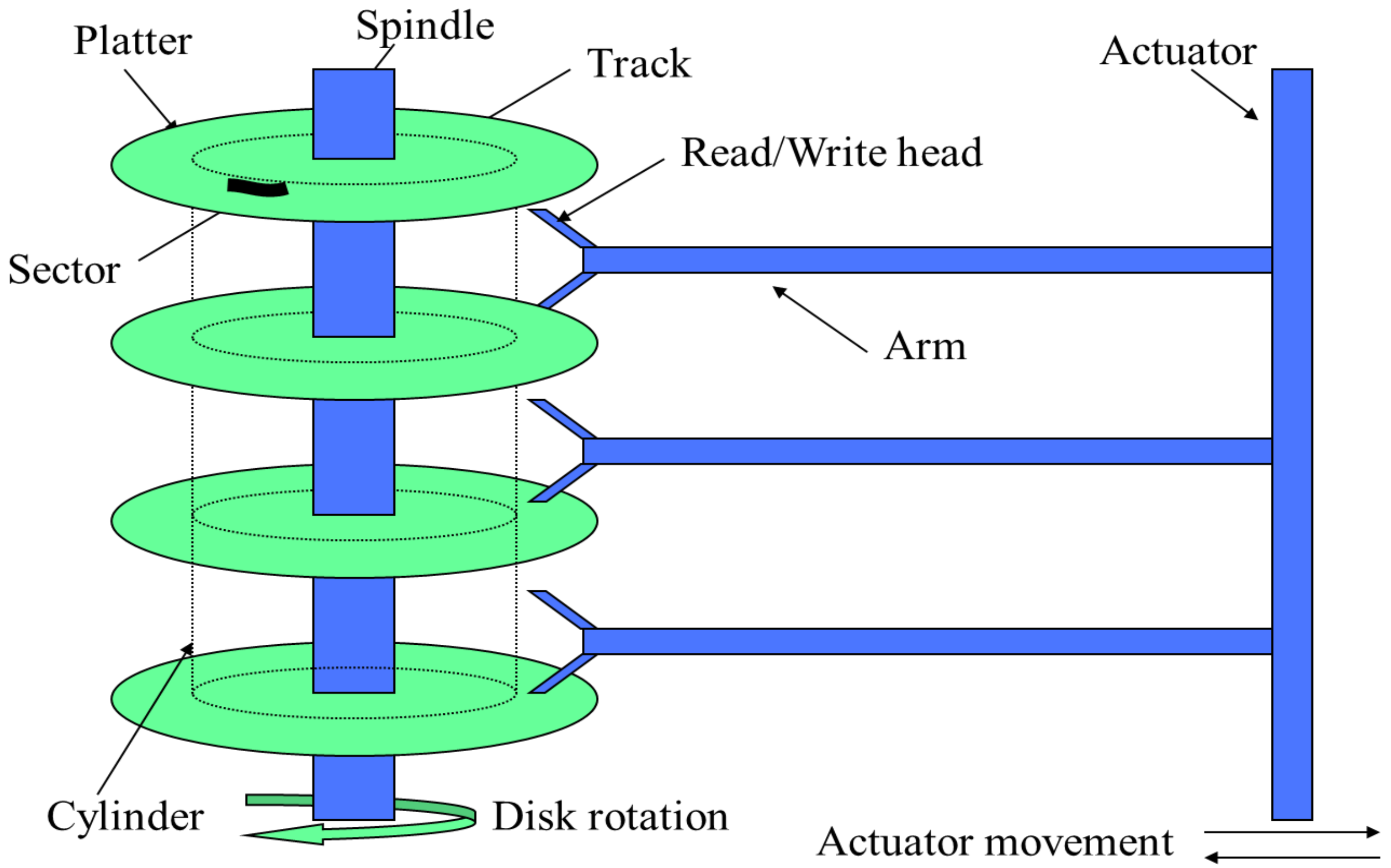
Storage Hierarchy

- Primary and secondary storage divided by what's known as the **I/O boundary**
  - Exchange of data between secondary storage and primary storage is called **I/O** (input/output) and is supervised by the operating system
- **Primary storage** contains database buffer and runtime code of the applications and DBMS
- **Secondary storage** contains physical data files.

Outline

- Storage Hardware and Physical Databases
  - The Storage Hierarchy
  - ☞ **Internals of Hard Disk Drives**
  - From Logical Concepts to Physical Constructs
- Record Organization
- File Organization

Internals of Hard Disk Drives



Internals of Hard Disk Drives

- Hard Disk Drive (HDD) stores data on circular **platters**, which are covered with magnetic particles
- **Platters** are secured on a **spindle**, which rotates at a constant speed
- A HDD also contains a **hard disk controller**
- **Read/write heads** can be positioned on arms, which are fixed to an **actuator**
- By combining disk rotation with actuator movement, each individual section of the disk is directly reachable

Internals of Hard Disk Drives

- Magnetic particles on **platters** are organized in concentric circular **tracks**, with each track consisting of **sectors**
- **Sector** is the smallest addressable unit on hard disk drive
  - traditionally: 512 bytes; recently: 4096 bytes
- A set of tracks, with the same diameter, is called a **cylinder**
- **Disk blocks** (aka clusters, **pages**, allocation units) consist of 2 or more physically adjacent sectors

Time to Retrieve a Disk Block

- **The response time to retrieve a disk block from a disk drive** is :

  Response time = service time + queueing time

  where, service time = **seek time**

  + **rotational delay**

  + transfer time

  They have considerable impact on overall performance of data retrieval

- Reading from a block, or writing to a block implies
  - positioning the actuator (**seek time**)
  - wait until the desired sector has rotated under the read/write head (**rotational delay**, or **latency**)
- **Transfer time** is typically fixed and depends on block size, density of magnetic particles and rotation speed of disks

Time to Retrieve a Disk Block

Suppose block size (BS), rotation time (ROT) and transfer rate (TR)

- **T_{rba}**: Expected time to retrieve/write disk block **randomly** (independently of previous read/write):

  $$\mathbf{T_{rba} = Seek + ROT/2 + BS/TR}$$

- **T_{sba} :** Expected time to **sequentially** retrieve disk block with R/W head already in correct position:

  $$\mathbf{T_{sba} = ROT/2 + BS/TR}$$

- Physical file organization can be optimized to **minimize expected seek time** and rotational delay

Example

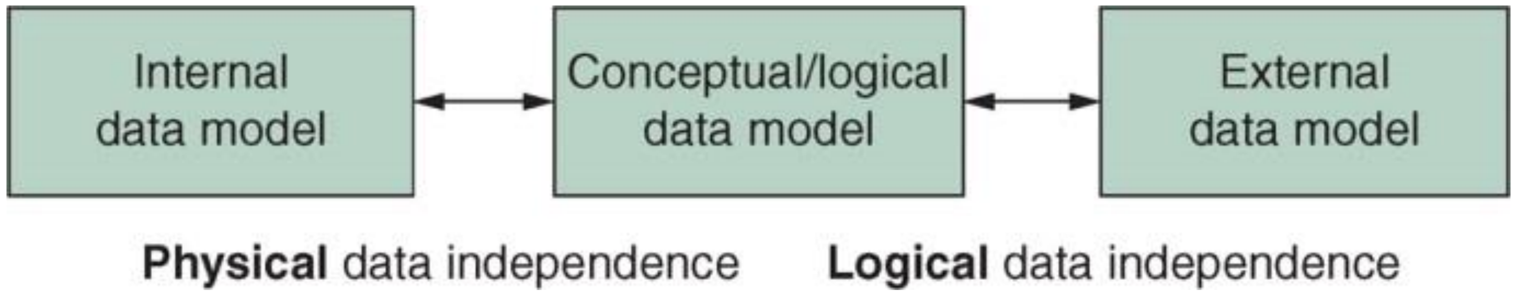
- For a HDD with the following characteristics:
  - Average seek time: 8.9 ms
  - Spindle speed (ROT): 7200 rpm (8.333 rpmillisecond)
  - Transfer rate (TR): 150 MBps (157286400 Bps)
  - Block size (BS): 4096 Bytes
- **Time to retrieve a disk block**
  - T_{rba} = Seek + ROT/2 + BS/TR
    = 8.9 ms + 4.167 ms + 0.026 ms = 13.093 ms
  - T_{sba} = ROT/2 + BS/TR
    = 4.167 ms + 0.026 ms = 4.193 ms
- $\mathbf{T_{rba} >> T_{sba}}$. It is recommended to organize physical files onto tracks and cylinders in a way that seeks and rotational delay are minimized as much as possible.

Outline

- Storage Hardware and Physical Databases
  - The Storage Hierarchy
  - Internals of Hard Disk Drives
  - ☞ **From Logical Concepts to Physical Constructs**
- Record Organization
- File Organization

From Logical Concepts to Physical Constructs

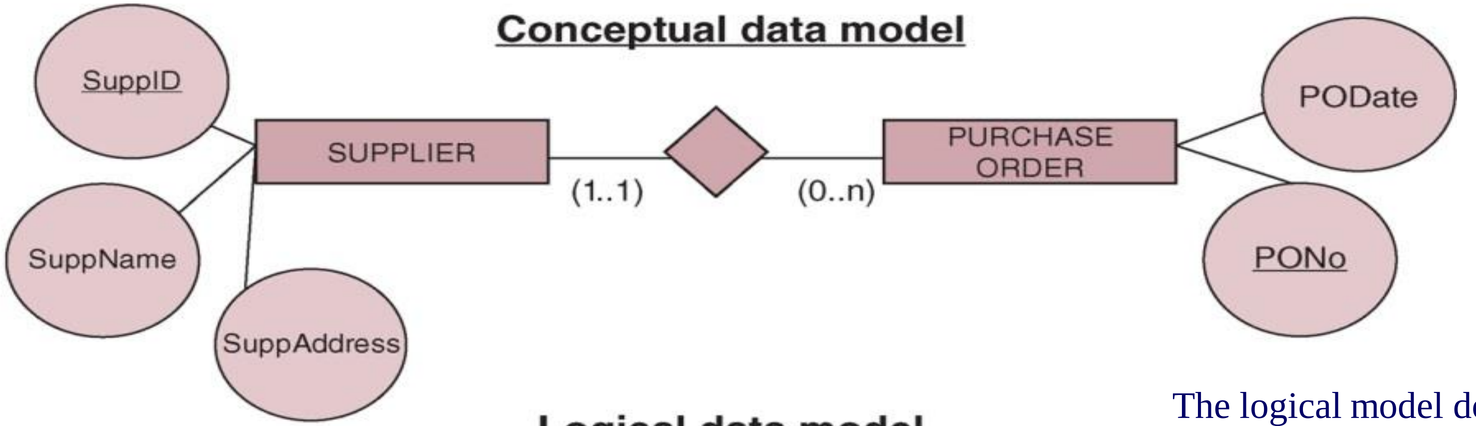
- **Internal data model** (a.k.a. **physical data model**) is about how a database is realized as a set of physical files and other constructs.
- **Physical database design** translates logical data model into physical data model


Internal data model
Conceptual/logical data model
External data model
Physical data independence
Logical data independence

Physical Database Design

- The **purpose of physical database design** is mostly to optimize update and retrieval efficiency by minimizing the number of required data block accesses, especially random block accesses.
- The inputs are:
  - the statistical properties of the data
  - the types of operations (search, insert, update, delete) that executed on the data
  - the physical properties of the storage media into account
- The internal model aims to provide adequate support for the most frequent and/or most time critical operations.

Example: Conceptual, Logical and Internal data model

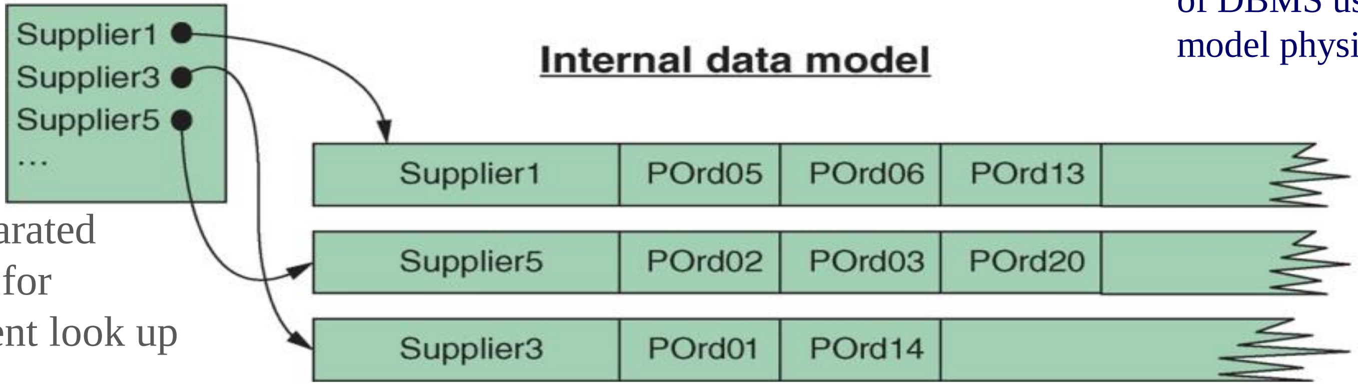


Logical data model

Supplier (<u>SuppID</u>, SuppName, SuppAddress)

PurchaseOrder (<u>PONo</u>, PODate, *SuppID*)

The logical model does not contain any concrete implementation related specification, but it does make an assumption about the actual type of DBMS used to implement the model physically.



A separated index for efficient look up

Corresponding Physical Concepts

Logical data model (general terminology)	**Logical data model (relational terminology)**	**Internal data model**
Attribute type and attribute	Column name and (cell) value	Data item or field
(Entity) record	Row or tuple	Stored record
(Entity) record type	Table or relation	Physical file or data set
Set of (entity) record types	Set of tables or relations	Physical database or stored database
Logical data structures	Foreign keys	Physical storage structures

Table. Corresponding logical (relational) and physical concepts

- We focus on physical organization of structured, relational data!

Physical Concepts

- A **data item** (also called **field**) is a collection of bits or characters that represents a specific value on a physical storage medium.
- A **stored record** is the physcal representation of a tuple in a reational table.
- A **physical file** implements a relational table.
  - In most cases, all records in a physical file have a similar structure. On some occasions, it may be required to combine stored records representing differnet real-world concepts into a single file.
- A **physical database** is an integrated collection of stored files.

Outline

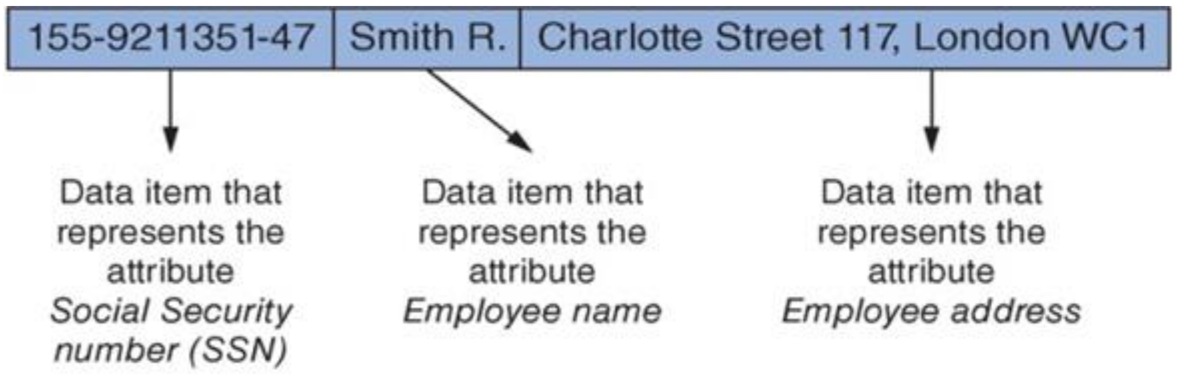
- Storage Hardware and Physical Databases
- ☞ **Record Organization**
- File Organization

Record Organization

- **Record organization** refers to organization of data items into physical records
  - Physical implementation of data item is a series of bits; the actual format depends on the attribute's data type (numeric, character, date and time, BLOB and CLOB, etc.).
- **Common techniques for record organization.**
  - relative location
  - embedded identification
  - pointers and lists

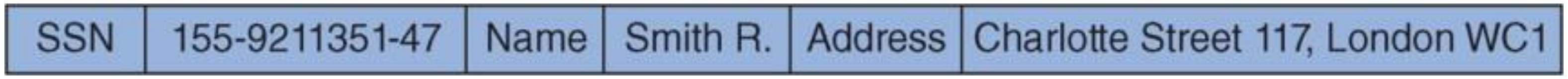
Record Organization – Relative Location

- **Relative location** is the simplest and most widespread technique for record organization
- Only the attributes are stored. - Data items that represent attributes of same entity are stored <u>on physically adjacent addresses</u>
- Attribute types (attribute names) are determined implicitly by relative ordering
- Example: EMPLOYEE (SSN, name, address, …)


155-9211351-47
Smith R.
Charlotte Street 117, London WC1
Data item that represents the attribute *Social Security number (SSN)*
Data item that represents the attribute *Employee name*
Data item that represents the attribute *Employee address*

Record Organization – Embedded Identification

- In **embedded identification**, data items representing attributes are always preceded by attribute type

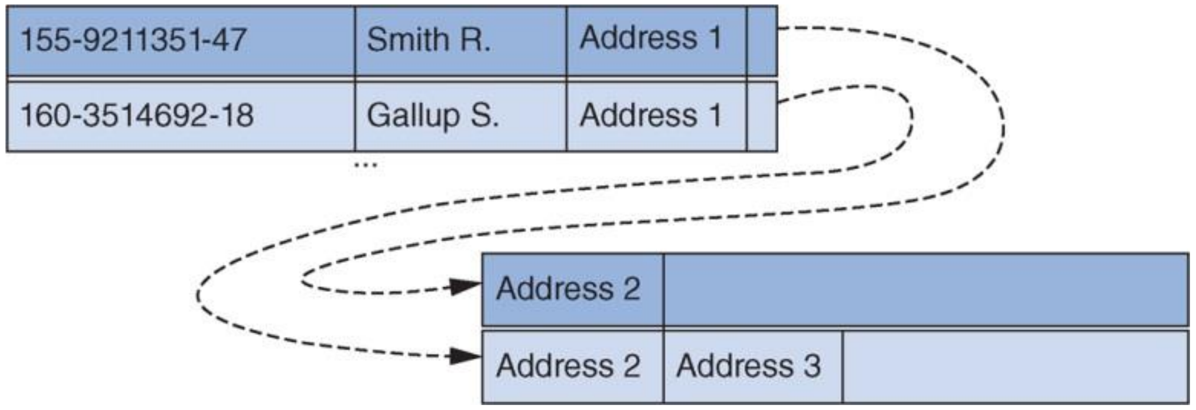


SSN	155-9211351-47	Name	Smith R.	Address	Charlotte Street 117, London WC1

- No need to store attributes in fixed order to identify them
- Only non-empty attributes of record are included.
- Similar to XML and JSON

Record Organization – Pointers and Lists

- **Pointers and lists** are ideal for dealing with variable length records, e.g., with variable length data type, multivalued attribute type, optional attribute type, etc.
- Example: A person may have different number of addresses.



- Pointers are also used for BLOB and CLOB data types which are mostly stored separately from the other data types, as they are much larger in size.

Dealing with Variable-length Records

- Another alternative for dealing with variable length records is to use **delimiters** that explicitly separate the respective attributes.

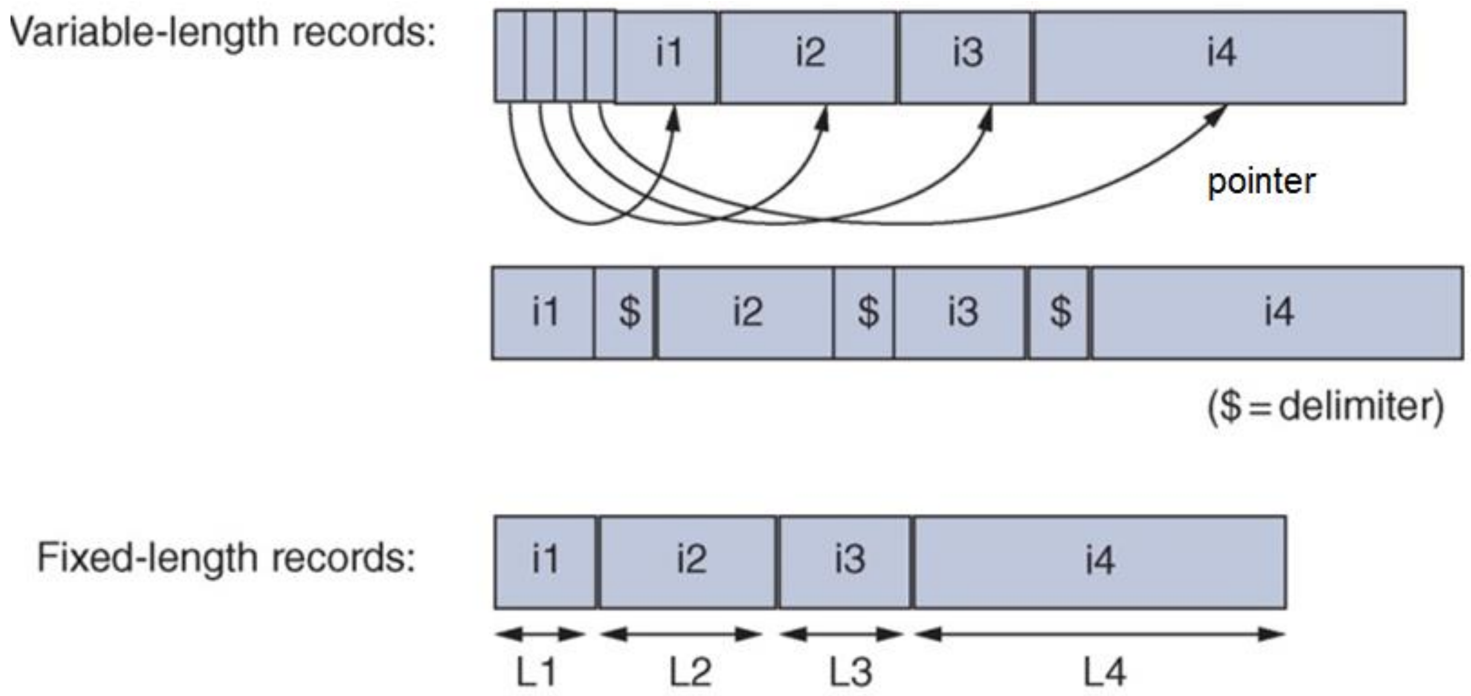


Figure: Dealing with fixed- and variable-length records

Blocking Factor

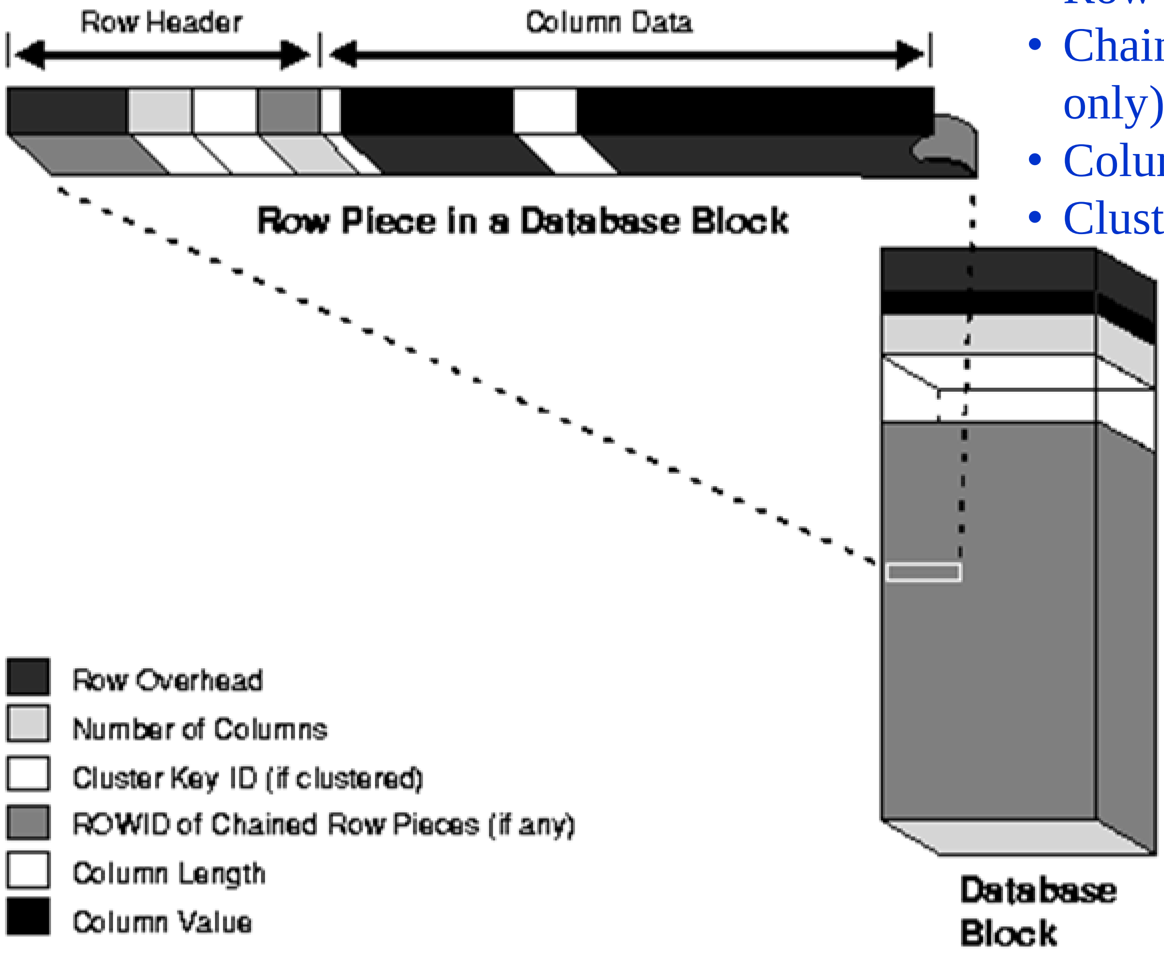
- **Blocking factor (BF)** indicates how many records are stored in single disk block
  - Because DBMS reads data in the unit of disk block, we can know how many records are retrieved with a single read operation from the blocking factor
- **For a file with fixed length records,** BF is calculated as:

$$\mathbf{BF = \lfloor BS/RS \rfloor}$$

where, BS: data block size, and RS: record size

- **For variable length records,** BF denotes the average number of records in a block

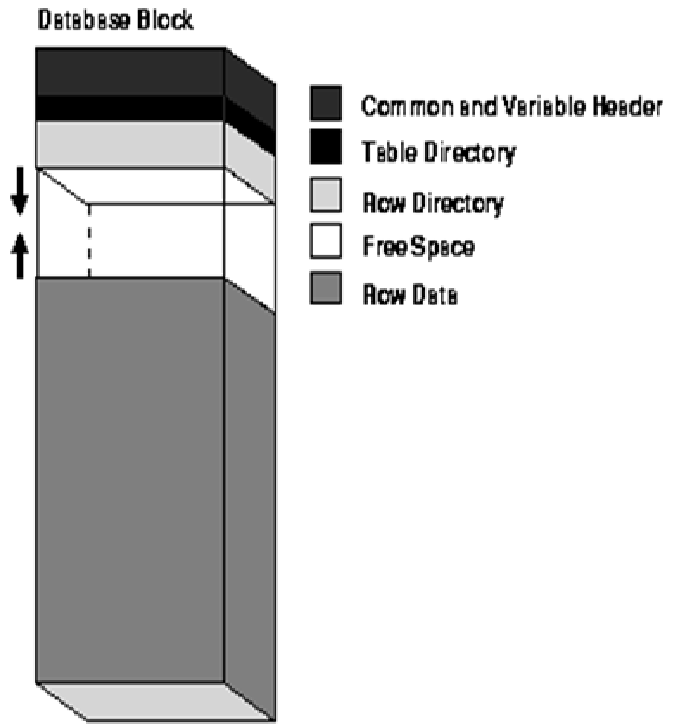
Example: Oracle Physical Record


Row Header
Column Data
Row Piece in a Database Block
Database Block
Row Overhead
Number of Columns
Cluster Key ID (if clustered)
ROWID of Chained Row Pieces (if any)
Column Length
Column Value

The **row header** precedes the data and, contains information about:

- Row pieces
- Chaining (for chained row pieces only)
- Columns in the row piece
- Cluster keys (for clustered data only)

Example: Oracle Data Block



Header (Common and Variable)
The header contains general block information, such as the block address and the type of segment (for example, data or index).
Table Directory
This portion of the data block contains information about the table having rows in this block.
Row Directory
This portion of the data block contains information about the actual rows in the block (including addresses for each row piece in the row data area).
After the space has been allocated in the row directory of a data block's overhead, this space is not reclaimed when the row is deleted. Therefore, a block that is currently empty but had up to 50 rows at one time continues to' have 100 bytes allocated in the header for the row directory. Oracle reuses this space only when new rows are inserted in the block.
Overhead
The data block header, table directory, and row directory are referred to collectively as **overhead**. Some block overhead is fixed in size; the total block overhead size is variable. On average, the fixed and variable portions of data block overhead total 84 to 107 bytes.
Row Data
This portion of the data block contains table or index data. Rows can span blocks.
Free Space
Free space is allocated for insertion of new rows and for updates to rows that require additional space (for example, when a trailing null is updated to a nonnull value). Whether issued insertions actually occur in a given data block is a function of current free space in that data block and the value of the space management parameter PCTFREE.
In data blocks allocated for the data segment of a table or cluster, or for the index segment of an index, free space can also hold transaction entries. A **transaction entry** is required in a block for each INSERT, UPDATE, DELETE, and SELECT...FOR UPDATE statement accessing one or more rows in the block. The space required for transaction entries is operating system dependent; however, transaction entries in most operating systems require approximately 23 bytes.

Outline

- Storage Hardware and Physical Databases
- Record Organization
- ☞ **File Organization**
  - Introductory Concepts
  - Heap File Organization
  - Sequential File Organization
  - Hashing File Organization (Random File Organization)
  - Indexed Sequential File Organization
  - Other Index Types

Terminology Search Key

- **Search key** is a single attribute type, or a set of attribute types, whose values determine criteria according to which records are retrieved.
- Search key is not the same as *key* (minimal set of fields that uniquely identify)
- Search key
  - can be primary key, alternative key, or one or more non-key attribute types
  - can be composite, e.g. (country, gender)
  - can also be used to specify range queries, e.g. YearOfBirth between 1980 and 1990

File Organization Methods

- Two categories of file organization methods
  - Primary file organization methods
  - Secondary file organization methods
- **Primary file organization methods** determine physical positioning of stored records on storage medium
  - e.g., **heap file organization, hash file organization, sequential file organization**
- **Secondary file organization methods** provide constructs to efficiently retrieve records according to search key that was not used for primary file organization (i.e., based on **secondary index)**

Heap File Organization

- **Heap file** is the most basic primary file organization method
- New records are inserted at end of file
- No relationship between record's attributes and physical location. Adding records is fairly efficient
- Only option for record retrieval is ***linear search***
- **Expected number of block accesses to retrieve a record**
  - For a file with NBLK blocks, it takes on average **NBLK /2** sba (sequential block access) to find a record according to unique search key, where NBLK represents number of blocks in data
  - Searching records according to non-unique search key requires scanning entire file

Example: Oracle Heap-Organized Table

- In Oracle, the default file organization for a table is heap file.
  - In the **heap-organized table**, the data rows are stored in no particular order on disk.
  - By default, `CREATE TABLE` creates a heap-organized table

```
CREATE TABLE t1
    (c1 NUMBER, c2 VARCHAR2(30));
```

Alternatively

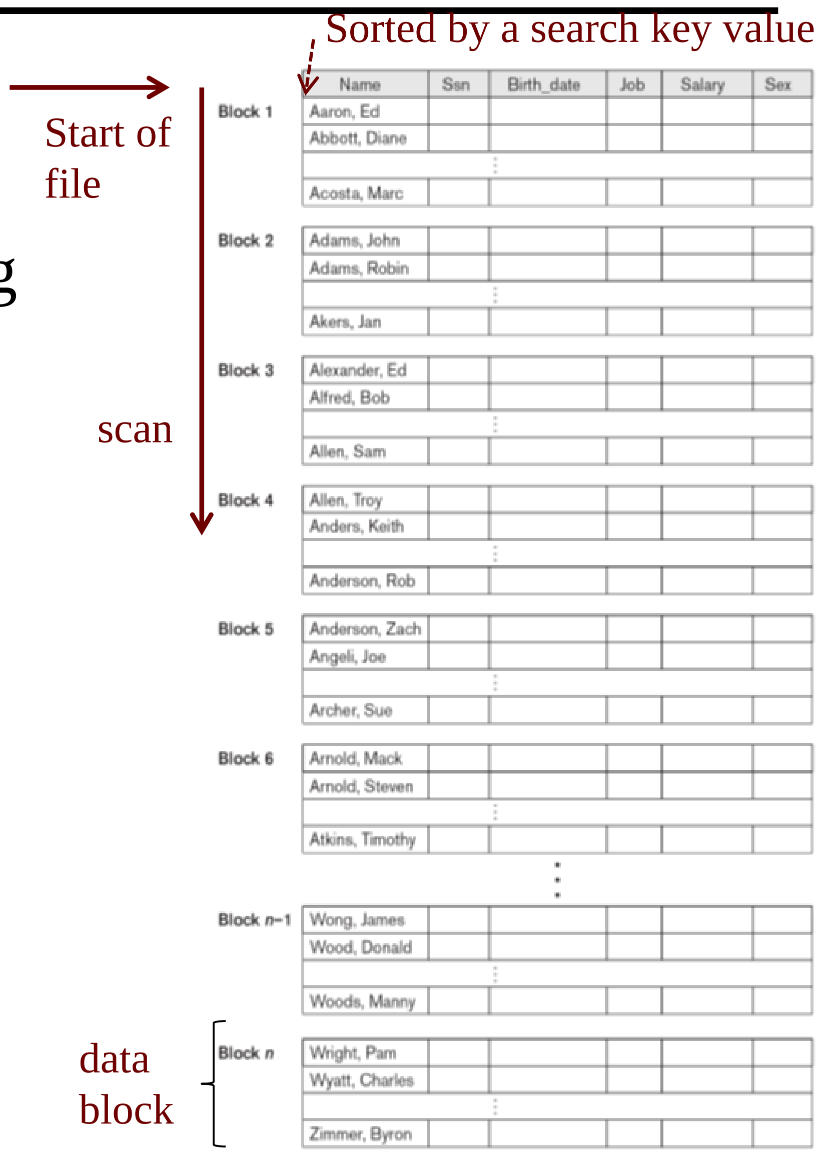
```
CREATE TABLE t1
    (c1 NUMBER, c2 VARCHAR2(30))
    ORGANIZATION HEAP;
```

Outline

- Storage Hardware and Physical Databases
- Record Organization
- File Organization
  - Primary/Secondary File Organization Methods
  - Heap File Organization
  - ☞ **Sequential File Organization**
  - Hash File Organization (Random File Organization)
  - Indexed Sequential File Organization
  - Other Index Types

Sequential File Organization

- With **sequential file organization**, records are stored in ascending /descending order of search key
- The search key is often the primary key, but a non-key attribute type or set of attribute types can also be used as ordering criterion.
- **Advantage**: When retrieving records in order determined by the search key, it is efficient



Sequential File Organization

- Records can still be retrieved by means of ***linear search***, or ***binary search*** technique
  - A binary search algorithm is applied recursively, halving the search interval with each iteration.
- **Expected number of block accesses to retrieve a record** according to the search key by means of
  - **linear search**: **NBLK/2** sba (sequential block access)
  - **binary search**: **$\log_2$(NBLK)** rba (random block access)

, where NBLK is the number of blocks in a file

Example

- Suppose that

Number of records (NR)	30000
Block size (BS)	2048 bytes
Records size (RS)	100 bytes

  - Blocking factor: $BF=\lfloor BS/RS \rfloor = \lfloor 2048/100 \rfloor = 20$
  - Number of blocks in data: NBLK=30000/20=1500
  - If single record is retrieved according to primary key using linear search, expected number of required block accesses is 1500/2 = 750 sba (sequential block access)
  - If binary search is used, expected number of block accesses is $\log_2(1500) \approx 11$ rba (random block access)

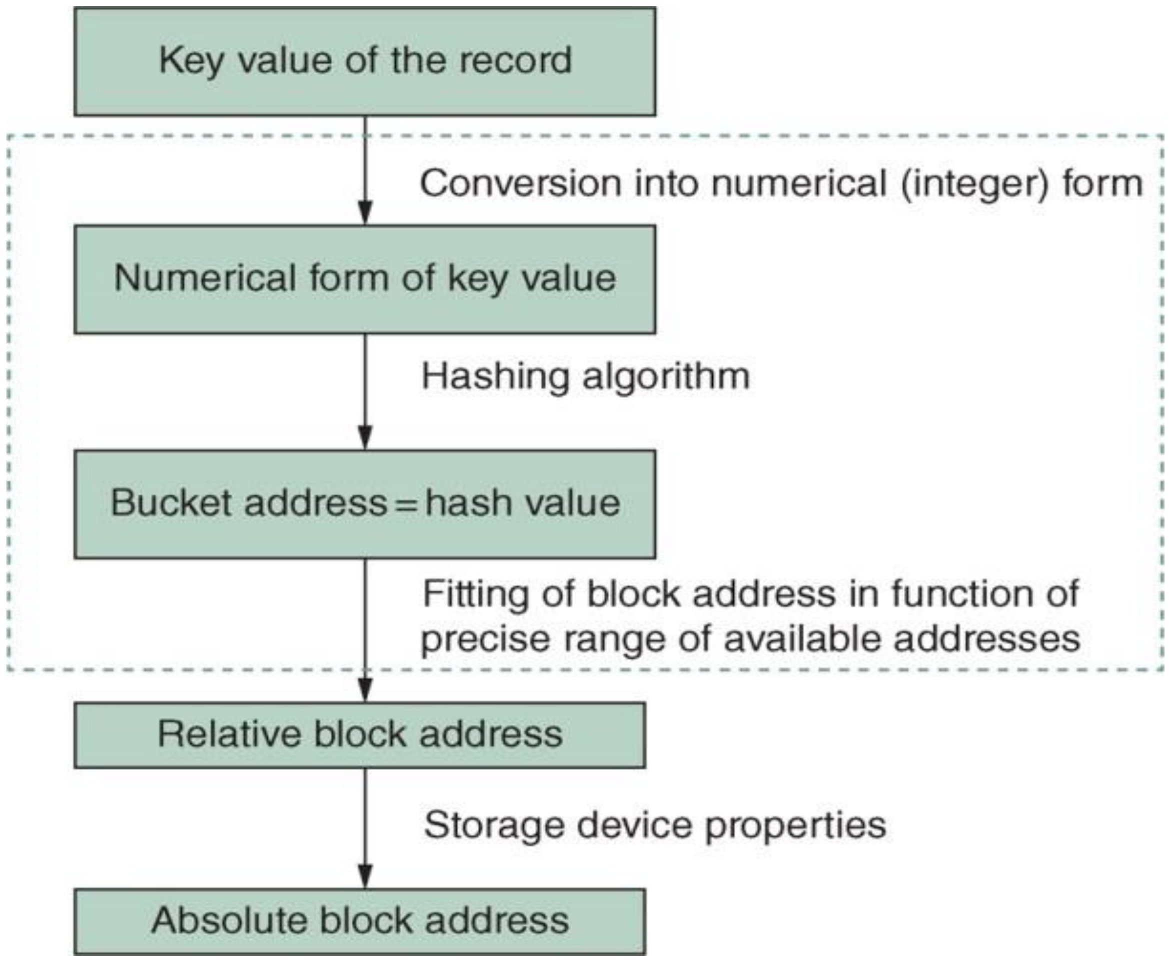
Outline

- Storage Hardware and Physical Databases
- Record Organization
- File Organization
  - Primary/Secondary File Organization Methods
  - Heap File Organization
  - Sequential File Organization
  - ☞ **Hash File Organization** (Random File Organization)
  - Indexed Sequential File Organization
  - Index Tree File Organization
  - Other Index Types

Hash File Organization (/ Radom File Organization)

- **Hash file organization** assumes direct relationship between value of search key and physical location
- A **hashing algorithm** defines key-to-address transformation, such that the record's physical address can be calculated from its key value.
- Each time a new record is to be added to the file, this transformation is applied to its key, returning the physical address where the record should be stored.
- If later on the record is to be retrieved based on this search key, applying the same transformation to the key returns the address where the record can be found

Key-to-Address Transformation


Key value of the record
Conversion into numerical (integer) form
Numerical form of key value
Hashing algorithm
Bucket address = hash value
Fitting of block address in function of precise range of available addresses
Relative block address
Storage device properties
Absolute block address

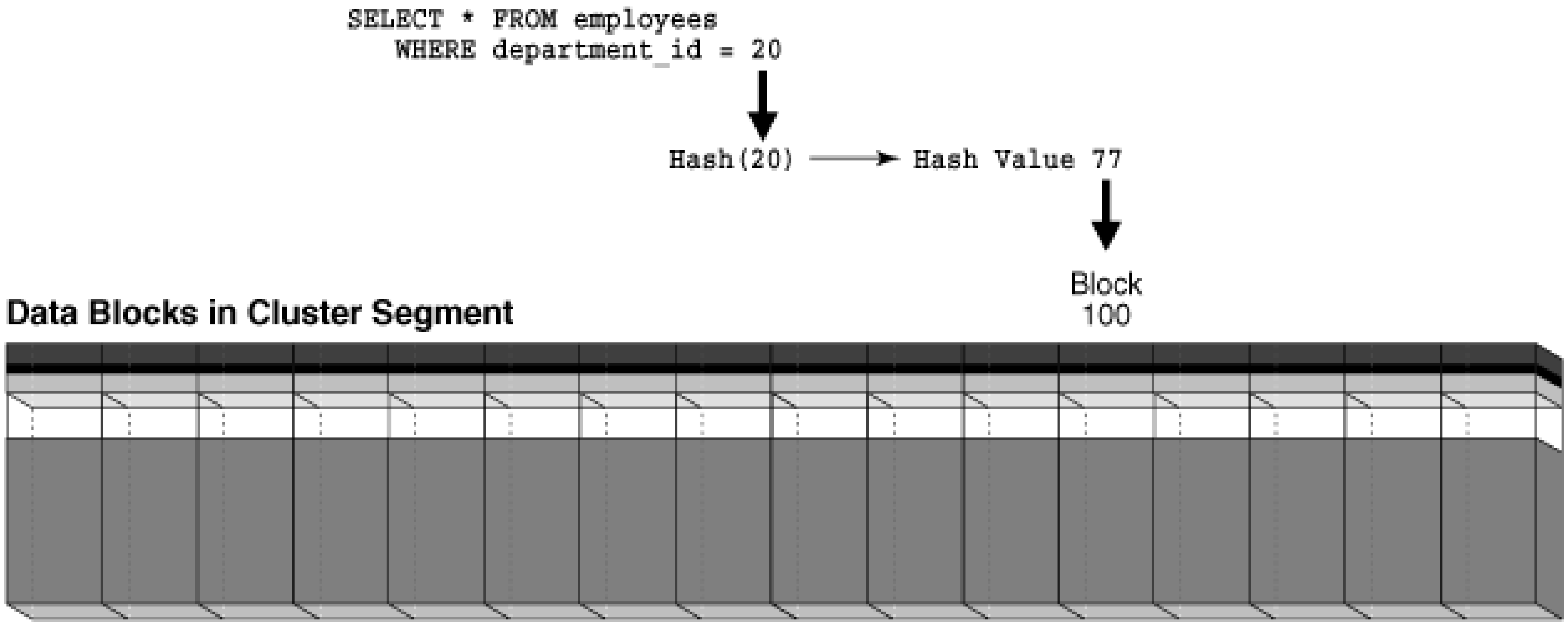
Key-to-Address Transformation

- The search key value (of number or alphanumerical) is converted into an integer numerical format
- A **hashing algorithm** is applied to the key.
  - A popular hashing algorithm is *division*:

    $$\mathbf{address(key_i) = key_i \bmod M}$$

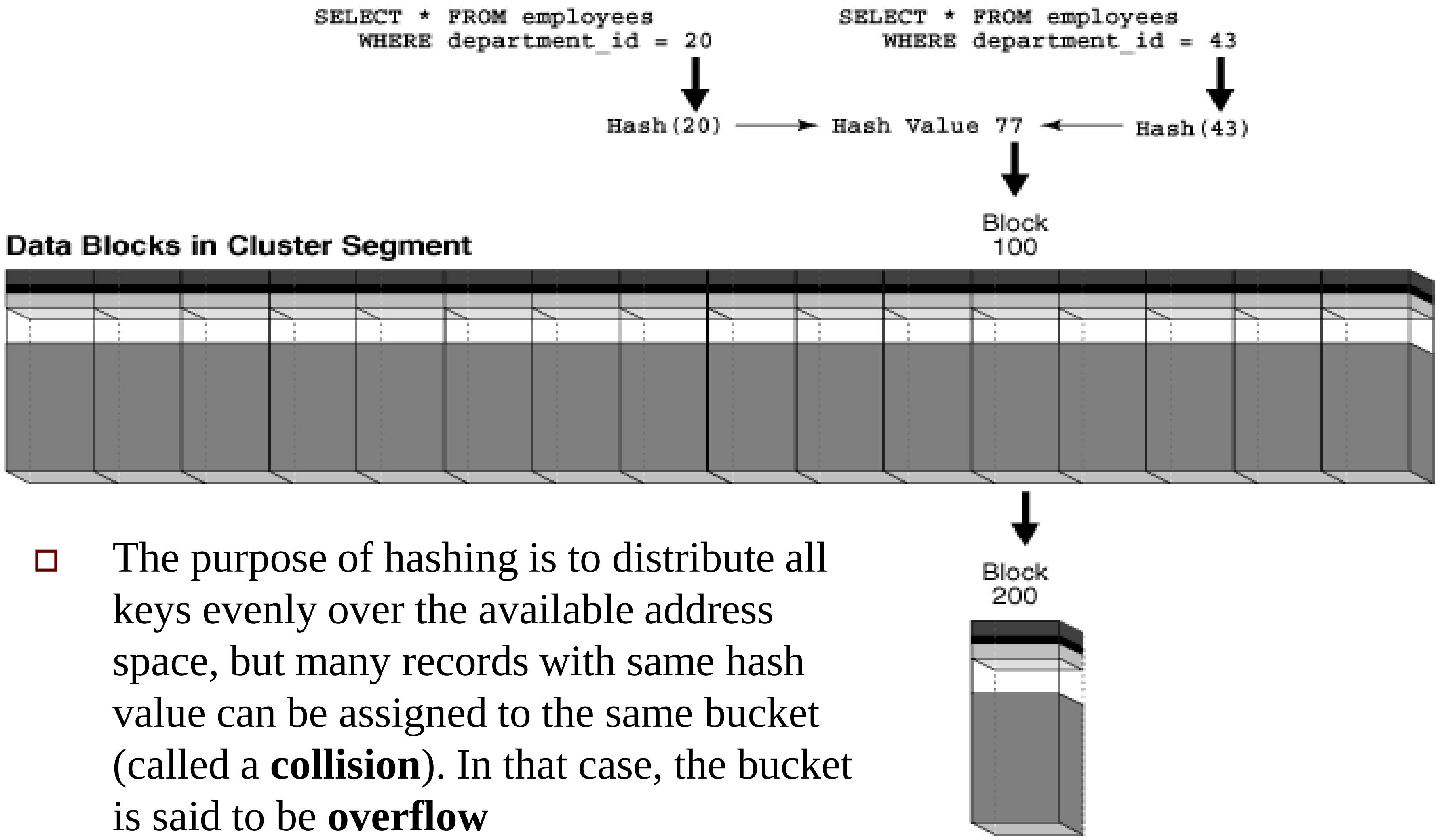
    , where M is often a prime number (close to, but a bit larger than, the number of available addresses)
  - The generated **hash value** pertains to a **bucket** (contiguous area of record addresses) **address**
  - The bucket address is translated into an actual block address

Illustration of Retrieving Data with Hashing



- When the record is to be retrieved based on the search key, applying the same transformation to the key returns the address where the record can be found

Illustration When a Hash Collision Occurs



- The purpose of hashing is to distribute all keys evenly over the available address space, but many records with same hash value can be assigned to the same bucket (called a **collision**). In that case, the bucket is said to be **overflow**

Hash File Organization Performance

- Retrieving non-overflow record:
  - 1 rba (random block access) to first block of bucket denoted by hashing algorithm, possibly followed by **1 or more** sba (sequential block accesses)
- Hash file organization is very efficient especially for finding a record with an equality condition with the search key
- Additional block accesses needed for overflow record depending on percentage of overflow records and overflow handling technique

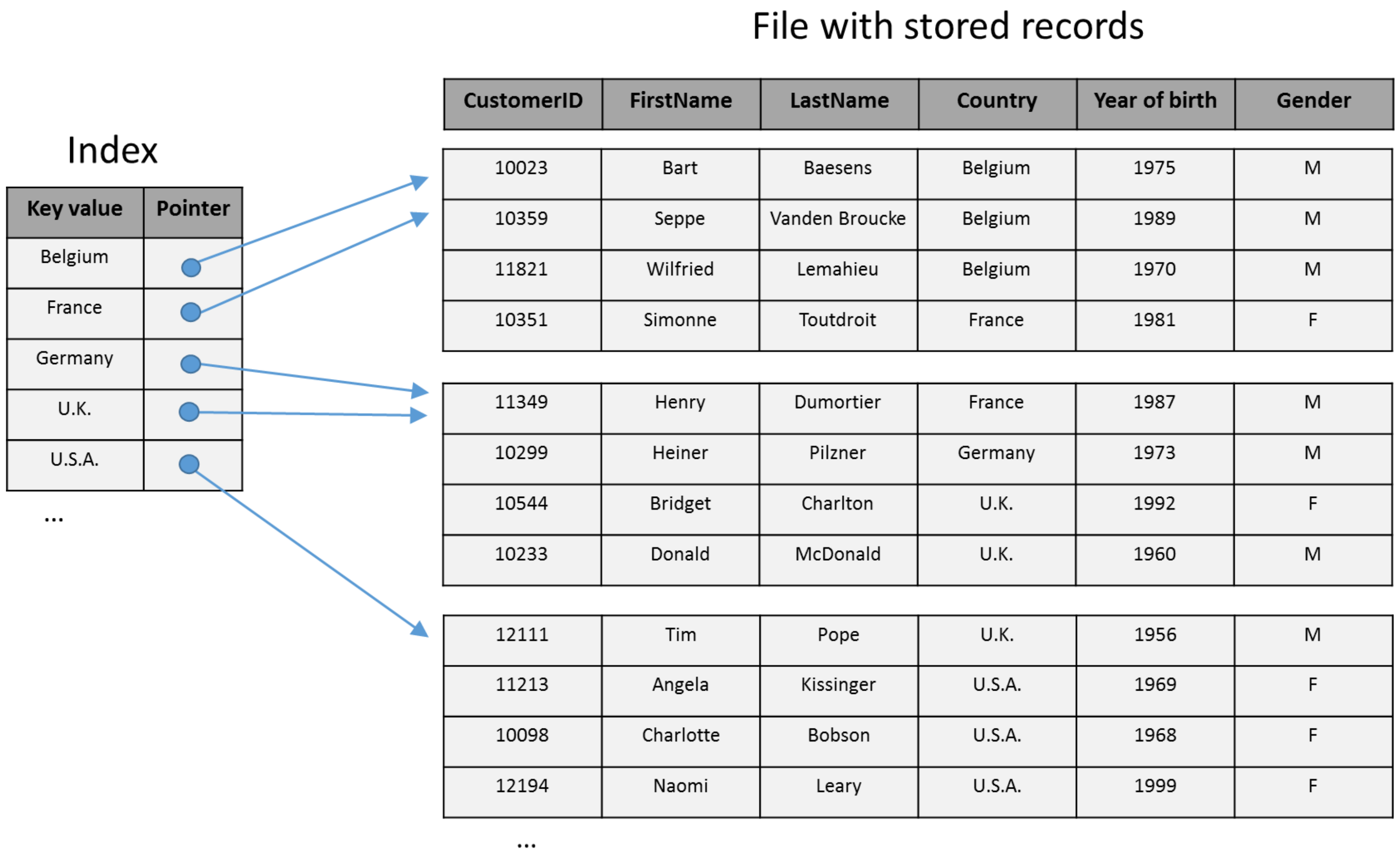
Outline

- Storage Hardware and Physical Databases
- Record Organization
- File Organization
  - Primary/Secondary File Organization Methods
  - Heap File Organization
  - Sequential File Organization
  - Hash File Organization (Random File Organization)
  - ☞ **Indexed Sequential File Organization**
  - Other Index Types

Indexed Sequential File Organization

- Heap file organization is mostly efficient to retrieve individual records by search key value (equality search) but not for range query.
- Sequential file organization is efficient if many records are to be retrieved in certain order (range search)
- **Indexed sequential file organization** method is good for both equality search and range search.
- Indexed sequential file organization combines sequential file organization with one or more indexes

Example: Indexed Sequential File Organization



Index

Key value	Pointer
Belgium	●
France	●
Germany	●
U.K.	●
U.S.A.	●

…

File with stored records

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10023	Bart	Baesens	Belgium	1975	M
10359	Seppe	Vanden Broucke	Belgium	1989	M
11821	Wilfried	Lemahieu	Belgium	1970	M
10351	Simonne	Toutdroit	France	1981	F
11349	Henry	Dumortier	France	1987	M
10299	Heiner	Pilzner	Germany	1973	M
10544	Bridget	Charlton	U.K.	1992	F
10233	Donald	McDonald	U.K.	1960	M
12111	Tim	Pope	U.K.	1956	M
11213	Angela	Kissinger	U.S.A.	1969	F
10098	Charlotte	Bobson	U.S.A.	1968	F
12194	Naomi	Leary	U.S.A.	1999	F

…

Index Entry

- An **index entry** contains the search key value of first record in interval and a pointer to physical position of first record in interval

  Index entry

  = <search key value, **block pointer** or **record pointer**>

  - e.g., <77788, pointer>, <(1980, M), pointer>

- The index entries are much smaller than actual stored records.

Type of Indexes

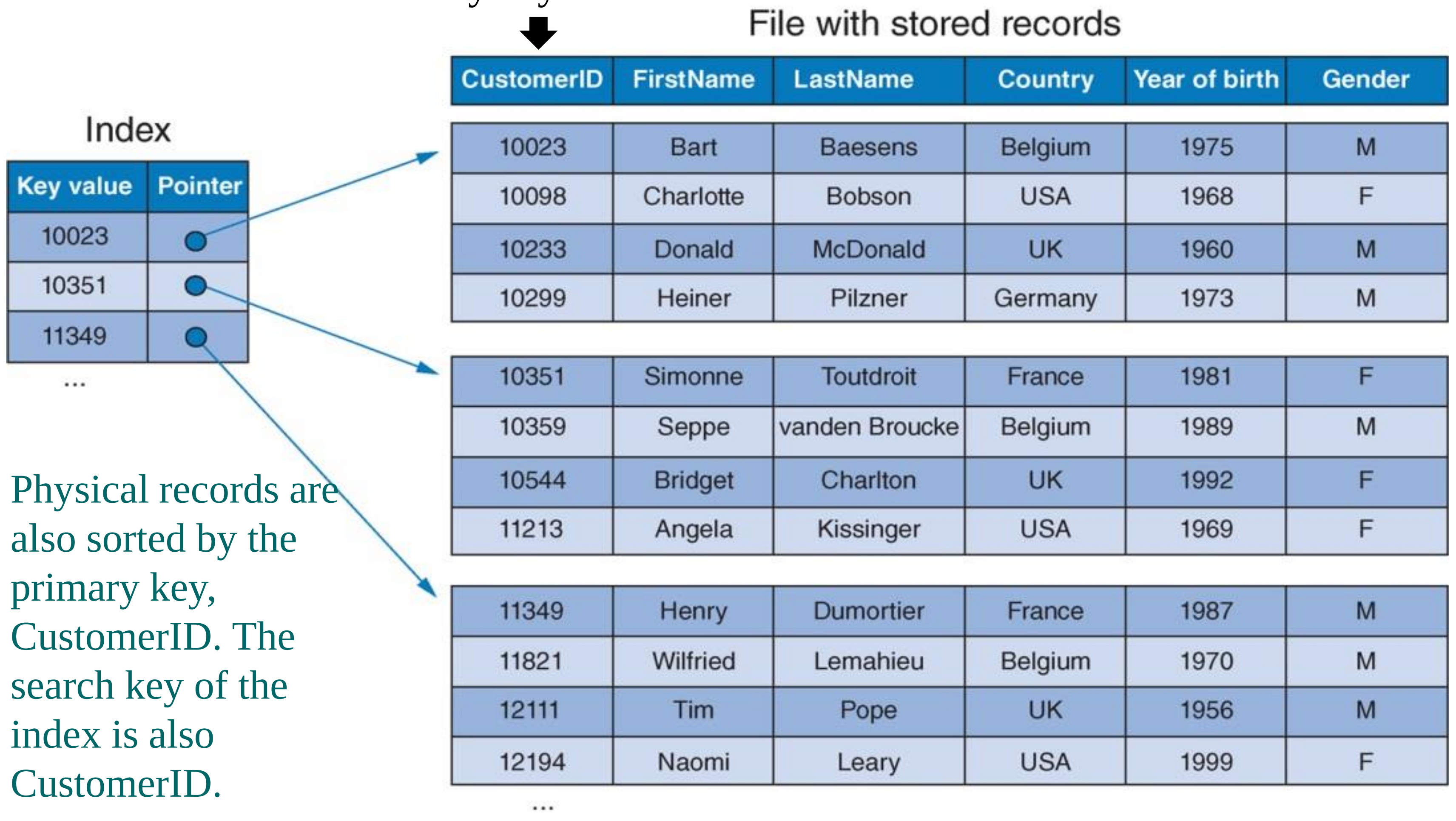
- Primary index
- Clustered index
- Secondary index
- Multicolumn index
  - Multicolumn primary index,
  - Multicolumn clustered index, Multicolumn secondary index
- Multilevel index
  - Single-level tree index, Multilevel tree index
- Bitmap index
- etc.

Primary index

- With **primary index** file organization, the data file is **ordered on a unique key** (primary key or another candidate key)
- There is an index entry for each disk block, not for each key value. So the number of index entries is small than other index structure.
- For finding a record, only the index and a single block of the actual data file are accessed.
- Either the record is found that block or it isn’t present in the file.

Example: Primary Index

Primary key attribute



Index

Key value	Pointer
10023	●
10351	●
11349	●
…	

File with stored records

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10023	Bart	Baesens	Belgium	1975	M
10098	Charlotte	Bobson	USA	1968	F
10233	Donald	McDonald	UK	1960	M
10299	Heiner	Pilzner	Germany	1973	M
10351	Simonne	Toutdroit	France	1981	F
10359	Seppe	vanden Broucke	Belgium	1989	M
10544	Bridget	Charlton	UK	1992	F
11213	Angela	Kissinger	USA	1969	F
11349	Henry	Dumortier	France	1987	M
11821	Wilfried	Lemahieu	Belgium	1970	M
12111	Tim	Pope	UK	1956	M
12194	Naomi	Leary	USA	1999	F
…					

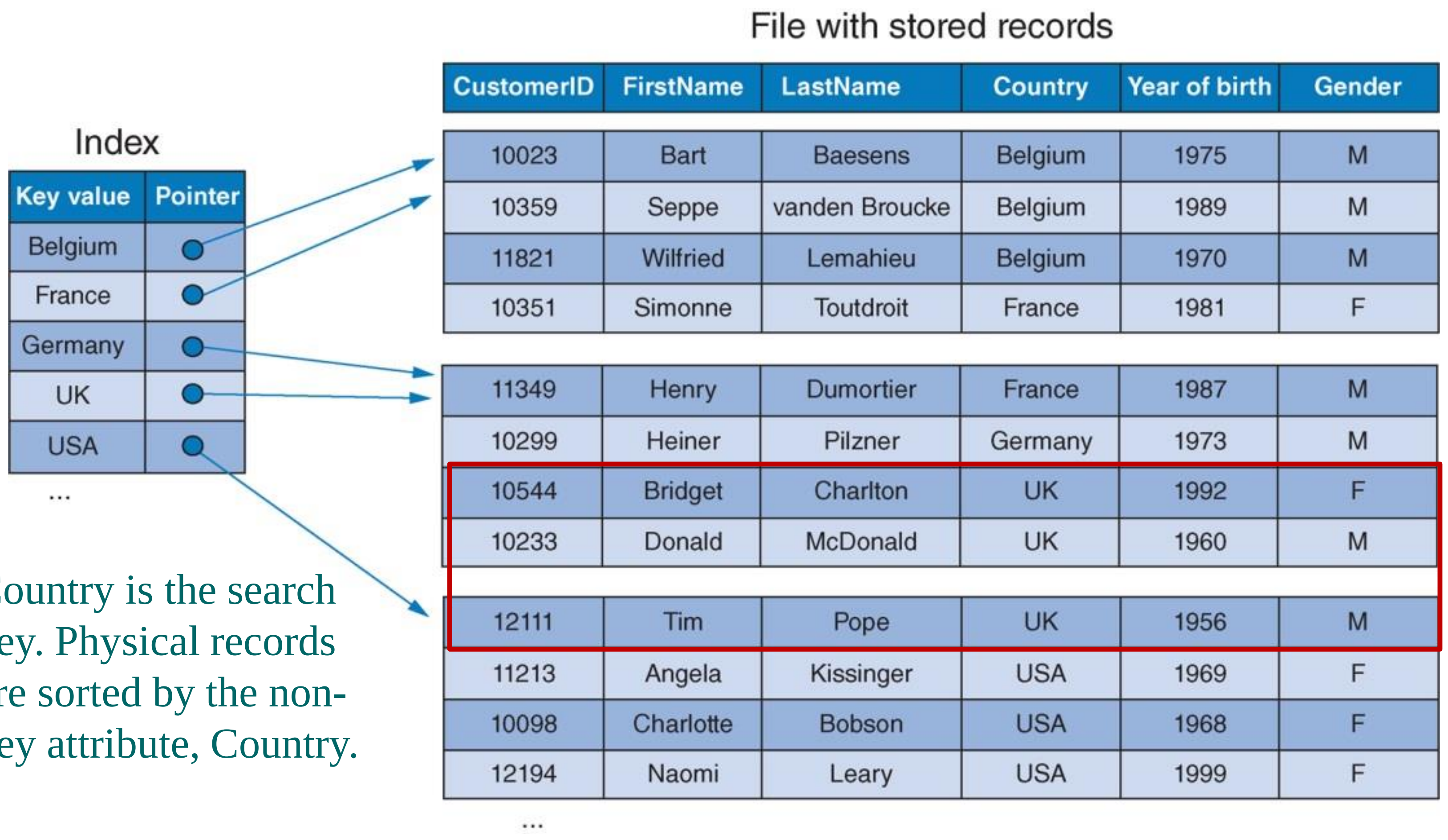
Physical records are also sorted by the primary key, CustomerID. The search key of the index is also CustomerID.

Either the record is found that block or it isn't present in the file.

Clustered Index

- A **clustered index** is similar to a primary index, with difference that the search key is a non-key attribute type or set of attribute types.
- Records are physically ordered on the search key
- After a block access with a search, additional sequential block accesses may be required to retrieve all subsequent records with the same search key value.

Example: Clustered Index



Index

Key value	Pointer
Belgium	●
France	●
Germany	●
UK	●
USA	●
...	

File with stored records

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10023	Bart	Baesens	Belgium	1975	M
10359	Seppe	vanden Broucke	Belgium	1989	M
11821	Wilfried	Lemahieu	Belgium	1970	M
10351	Simonne	Toutdroit	France	1981	F
11349	Henry	Dumortier	France	1987	M
10299	Heiner	Pilzner	Germany	1973	M
10544	Bridget	Charlton	UK	1992	F
10233	Donald	McDonald	UK	1960	M
12111	Tim	Pope	UK	1956	M
11213	Angela	Kissinger	USA	1969	F
10098	Charlotte	Bobson	USA	1968	F
12194	Naomi	Leary	USA	1999	F
...					

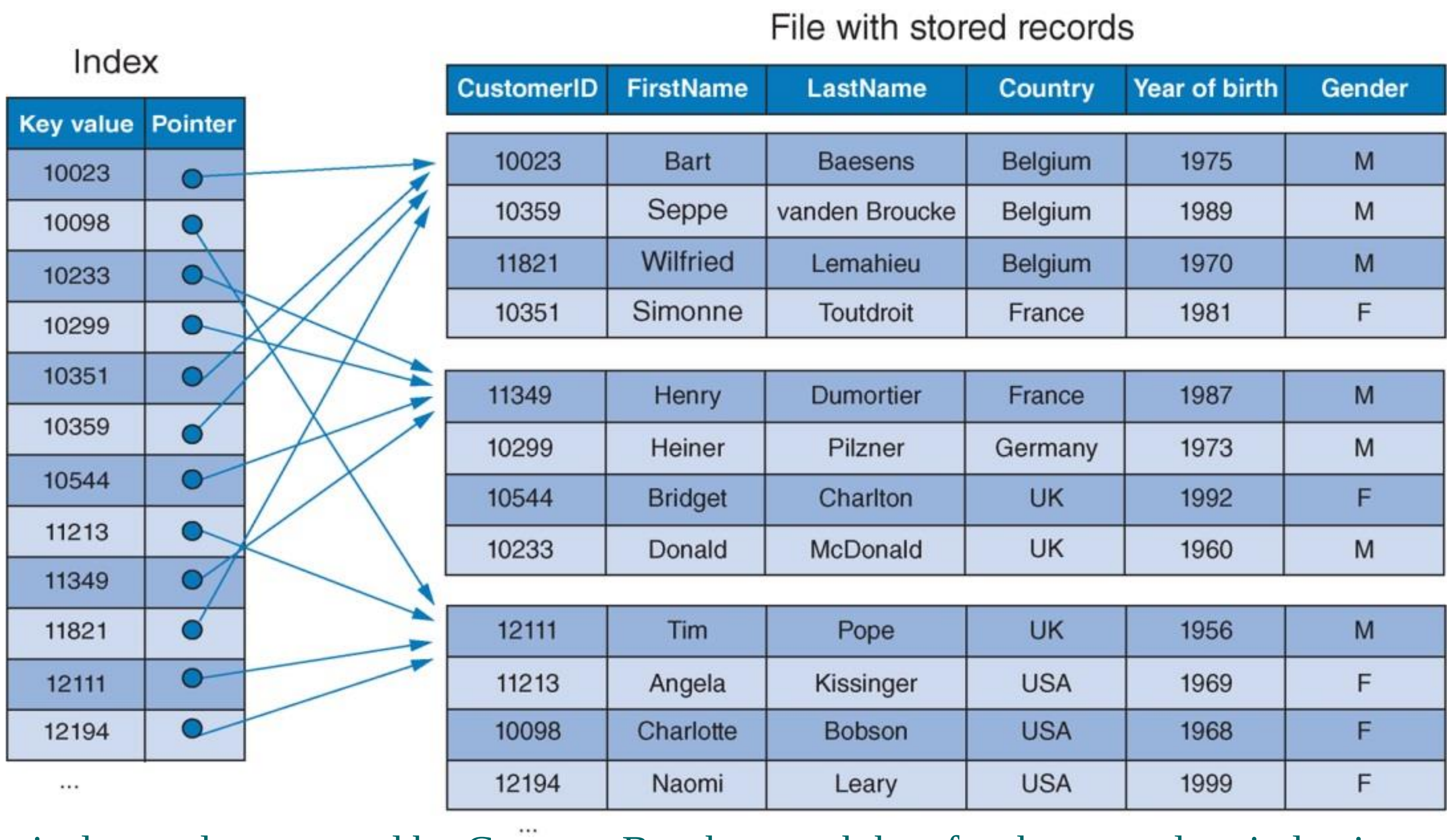
Country is the search key. Physical records are sorted by the non-key attribute, Country.

For finding records with UK, after a block access with a search, additional sequential block accesses are required to retrieve all subsequent records with “UK”.

Secondary Index

- In contrast to a primary index or a clustered index, a **secondary index** is based on an attribute type or set of attribute types that is/are <u>not used as ordering criteria of the actual data file.</u>
- <u>Secondary indexes have no impact on the physical ordering of the records</u>, but do allow speeding up retrieval according to criteria other than one used for the primary file organization.
- <u>There can be several secondary indexes for the same data file.</u>

Secondary Index with Unique Search Key



Index

Key value	Pointer
10023	●
10098	●
10233	●
10299	●
10351	●
10359	●
10544	●
11213	●
11349	●
11821	●
12111	●
12194	●

...

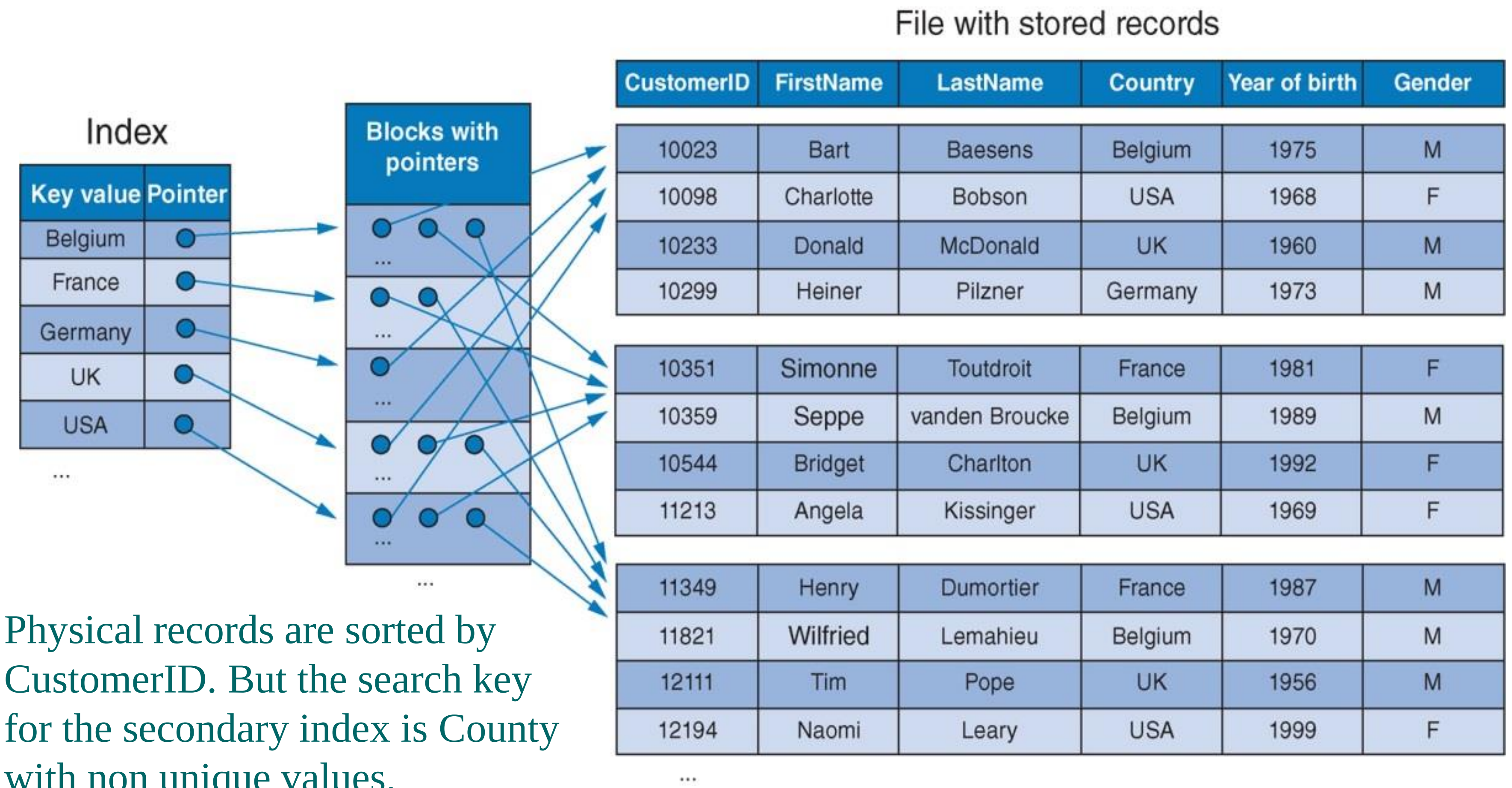
File with stored records

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10023	Bart	Baesens	Belgium	1975	M
10359	Seppe	vanden Broucke	Belgium	1989	M
11821	Wilfried	Lemahieu	Belgium	1970	M
10351	Simonne	Toutdroit	France	1981	F
11349	Henry	Dumortier	France	1987	M
10299	Heiner	Pilzner	Germany	1973	M
10544	Bridget	Charlton	UK	1992	F
10233	Donald	McDonald	UK	1960	M
12111	Tim	Pope	UK	1956	M
11213	Angela	Kissinger	USA	1969	F
10098	Charlotte	Bobson	USA	1968	F
12194	Naomi	Leary	USA	1999	F

...

Physical records are sorted by Country. But the search key for the secondary index is CustomerID. For unique search key, one index entry is needed for each record (= each key).

Secondary Index with No Unique Search Key



Index

Key value	Pointer
Belgium	●
France	●
Germany	●
UK	●
USA	●
...	

File with stored records

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10023	Bart	Baesens	Belgium	1975	M
10098	Charlotte	Bobson	USA	1968	F
10233	Donald	McDonald	UK	1960	M
10299	Heiner	Pilzner	Germany	1973	M
10351	Simonne	Toutdroit	France	1981	F
10359	Seppe	vanden Broucke	Belgium	1989	M
10544	Bridget	Charlton	UK	1992	F
11213	Angela	Kissinger	USA	1969	F
11349	Henry	Dumortier	France	1987	M
11821	Wilfried	Lemahieu	Belgium	1970	M
12111	Tim	Pope	UK	1956	M
12194	Naomi	Leary	USA	1999	F
...					

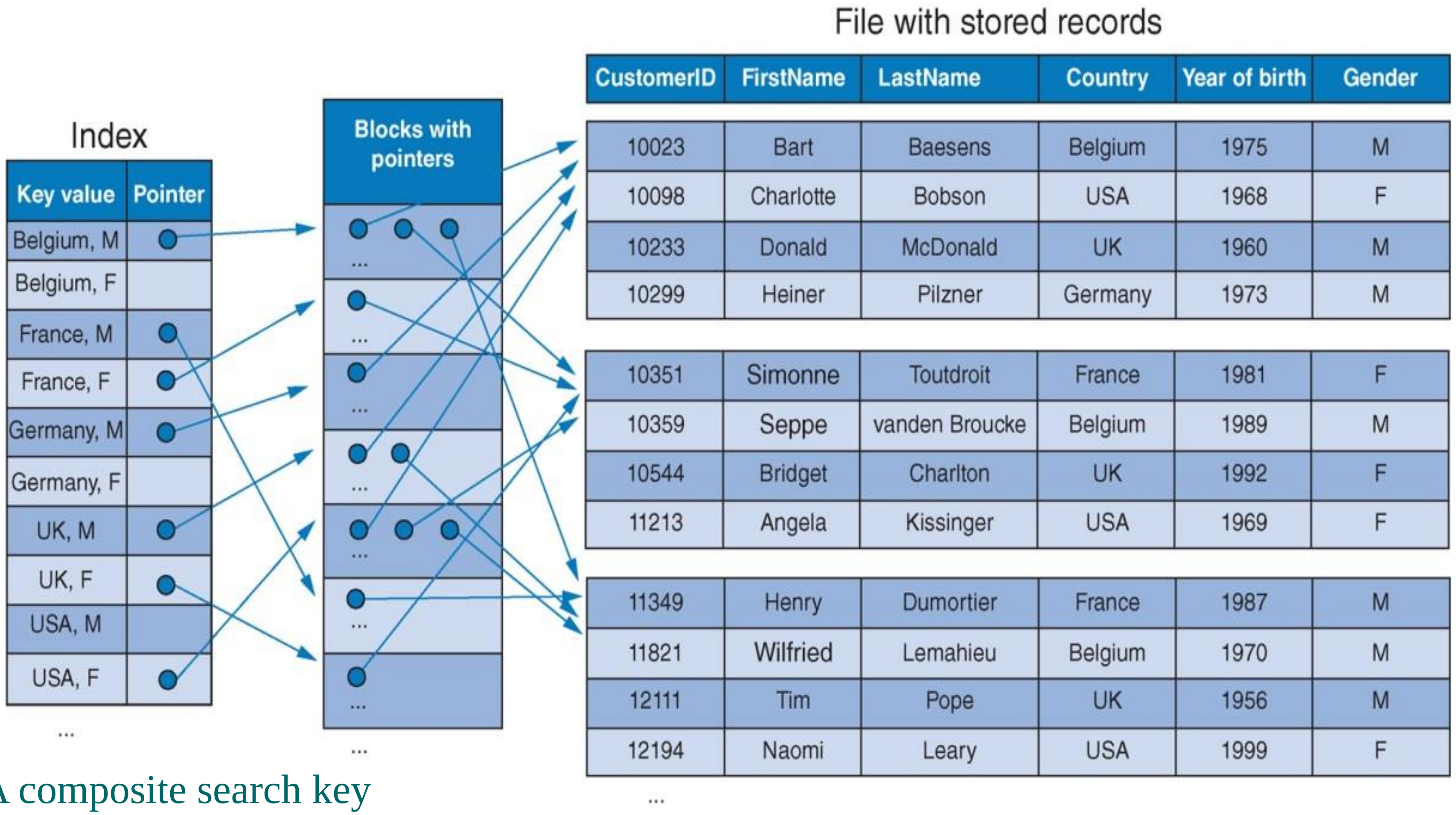
Physical records are sorted by CustomerID. But the search key for the secondary index is County with non unique values.

There is one index entry per key value and hence each entry may refer to multiple records with that same key value. An **inverted file** can be used for an index over a non-unique, non-ordering search key of a dataset.

Multicolumn Index

- An index over a composite search key is a so-called **multicolumn index**.
- A composite search key can be applied to primary, clustered, and secondary indexes.
  - Multicolumn primary index,
  - Multicolumn clustered index
  - Multicolumn secondary index

Example: Multicolumn Secondary Index



Index

Key value	Pointer
Belgium, M	●
Belgium, F	
France, M	●
France, F	●
Germany, M	●
Germany, F	
UK, M	●
UK, F	●
USA, M	
USA, F	●
...	

File with stored records

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10023	Bart	Baesens	Belgium	1975	M
10098	Charlotte	Bobson	USA	1968	F
10233	Donald	McDonald	UK	1960	M
10299	Heiner	Pilzner	Germany	1973	M
10351	Simonne	Toutdroit	France	1981	F
10359	Seppe	vanden Broucke	Belgium	1989	M
10544	Bridget	Charlton	UK	1992	F
11213	Angela	Kissinger	USA	1969	F
11349	Henry	Dumortier	France	1987	M
11821	Wilfried	Lemahieu	Belgium	1970	M
12111	Tim	Pope	UK	1956	M
12194	Naomi	Leary	USA	1999	F
...					

A composite search key with Country and Gender.

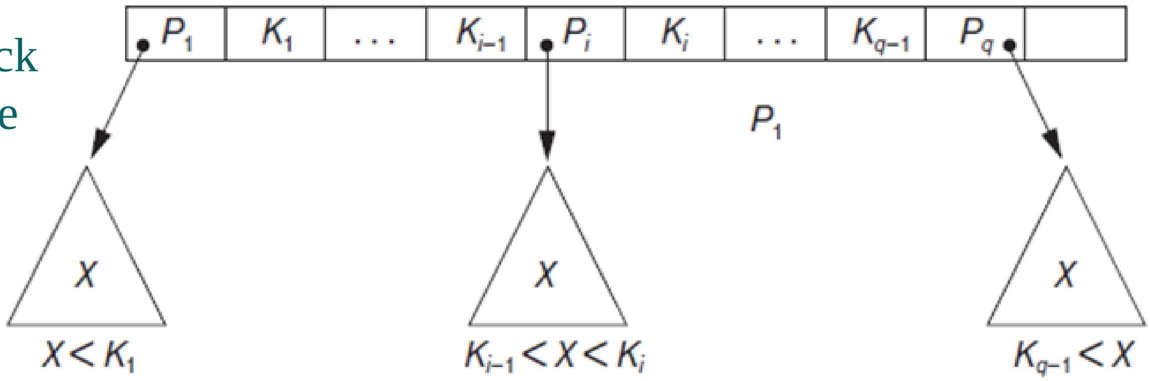
Multilevel Index

- In a single-level index, at some point, the index itself may grow too large to be searched efficiently.
- Building an index-to-the-index improves the index search. So **multilevel indexes** are introduced.

Multilevel Index

- A **search tree** used to guide search for a record, given value of one of record's fields

An index block == a tree node

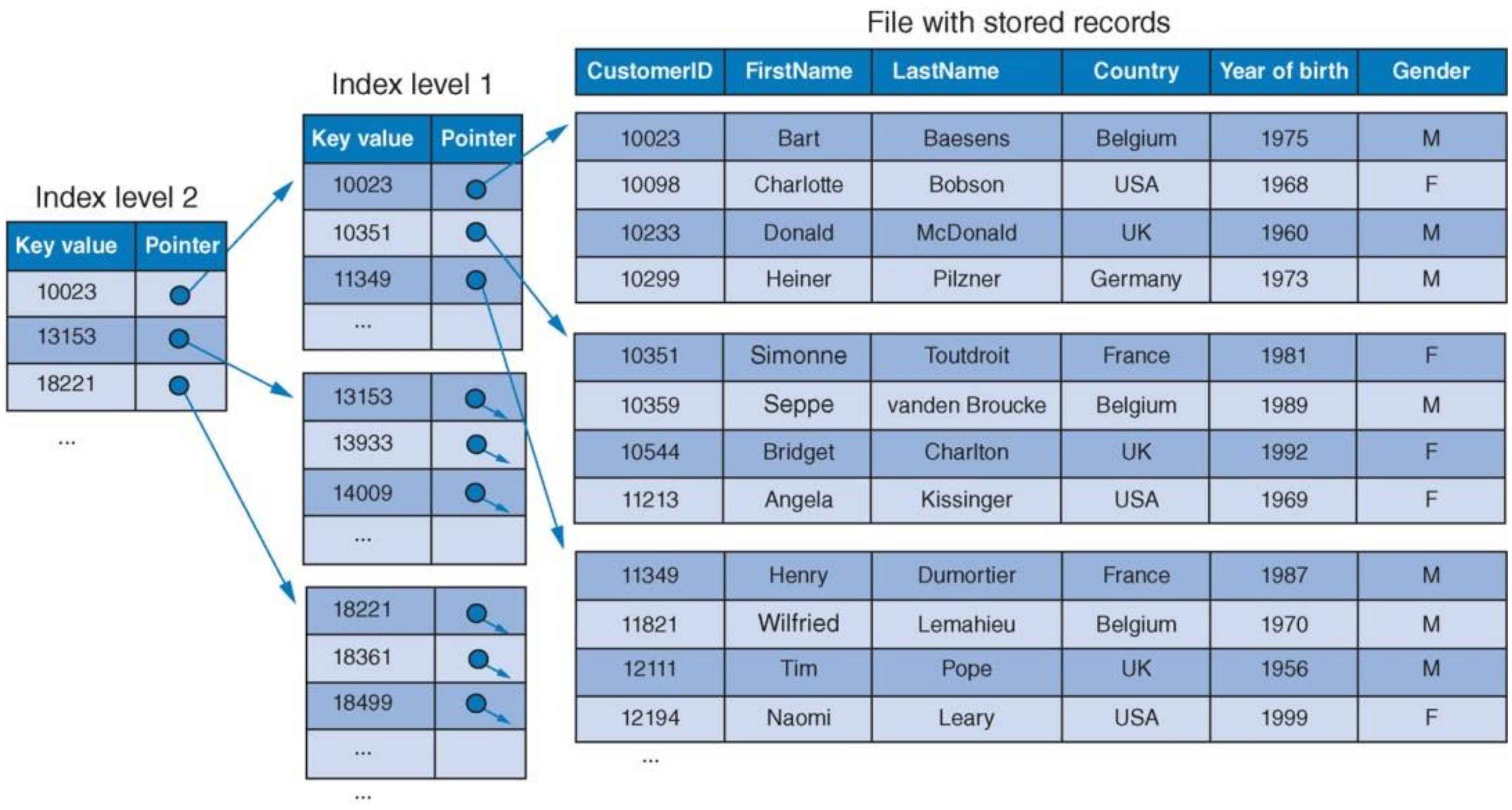


- A **multilevel index** organizes the index entries using a search tree.
- The performance gain induced by higher-level indexes is because an individual index is searched according to the search tree retrieval technique.

Multilevel Index

- A multilevel index can be considered as a search tree, with each index level representing a level in the tree
- Each index block represents a node of the tree
- Each index entry in an internal node consists of a search key value and a reference to the corresponding block (a node) in the lower-level index.
- Each access to the index results in navigation toward a subtree in the tree, hence <u>reducing the search </u>interval.
- The lowest-level index entries in a leaf node contain pointers to disk blocks or individual records.

Example: Multilevel Index



Index level 2

Key value	Pointer
10023	●
13153	●
18221	●
...	

Index level 1

Key value	Pointer
10023	●
10351	●
11349	●
...	

13153	●
13933	●
14009	●
...	

18221	●
18361	●
18499	●
...	

...

File with stored records

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10023	Bart	Baesens	Belgium	1975	M
10098	Charlotte	Bobson	USA	1968	F
10233	Donald	McDonald	UK	1960	M
10299	Heiner	Pilzner	Germany	1973	M
10351	Simonne	Toutdroit	France	1981	F
10359	Seppe	vanden Broucke	Belgium	1989	M
10544	Bridget	Charlton	UK	1992	F
11213	Angela	Kissinger	USA	1969	F
11349	Henry	Dumortier	France	1987	M
11821	Wilfried	Lemahieu	Belgium	1970	M
12111	Tim	Pope	UK	1956	M
12194	Naomi	Leary	USA	1999	F

...

Search Performance of Various File Organizations

Linear search	**NBLK** sba
Binary search	**$\log_2$(NBLK)** rba
Single-level index based search	**$\log_2$(NBLKI) + 1** rba, (1 for additional block access to actual data file)
Multilevel index based search	**$\log_{BFI}$(NBLKI) + 1+ 1** rba, (1 for root index, 1 for data file access)

- NBLK : Number of blocks in data
- NBLKI : Number of blocks in index, **NBLKI << NBLK**
- BFI (also called the *fan-out* of index) : blocking factor in index, The search interval reduced by a factor BFI with every index level, typically, **BFI >> 2**

- Index file occupies fewer disk blocks than data file and can be searched much quicker
- BFI is typically much higher than two, so using a multilevel index is more efficient than a binary search on a single-level index

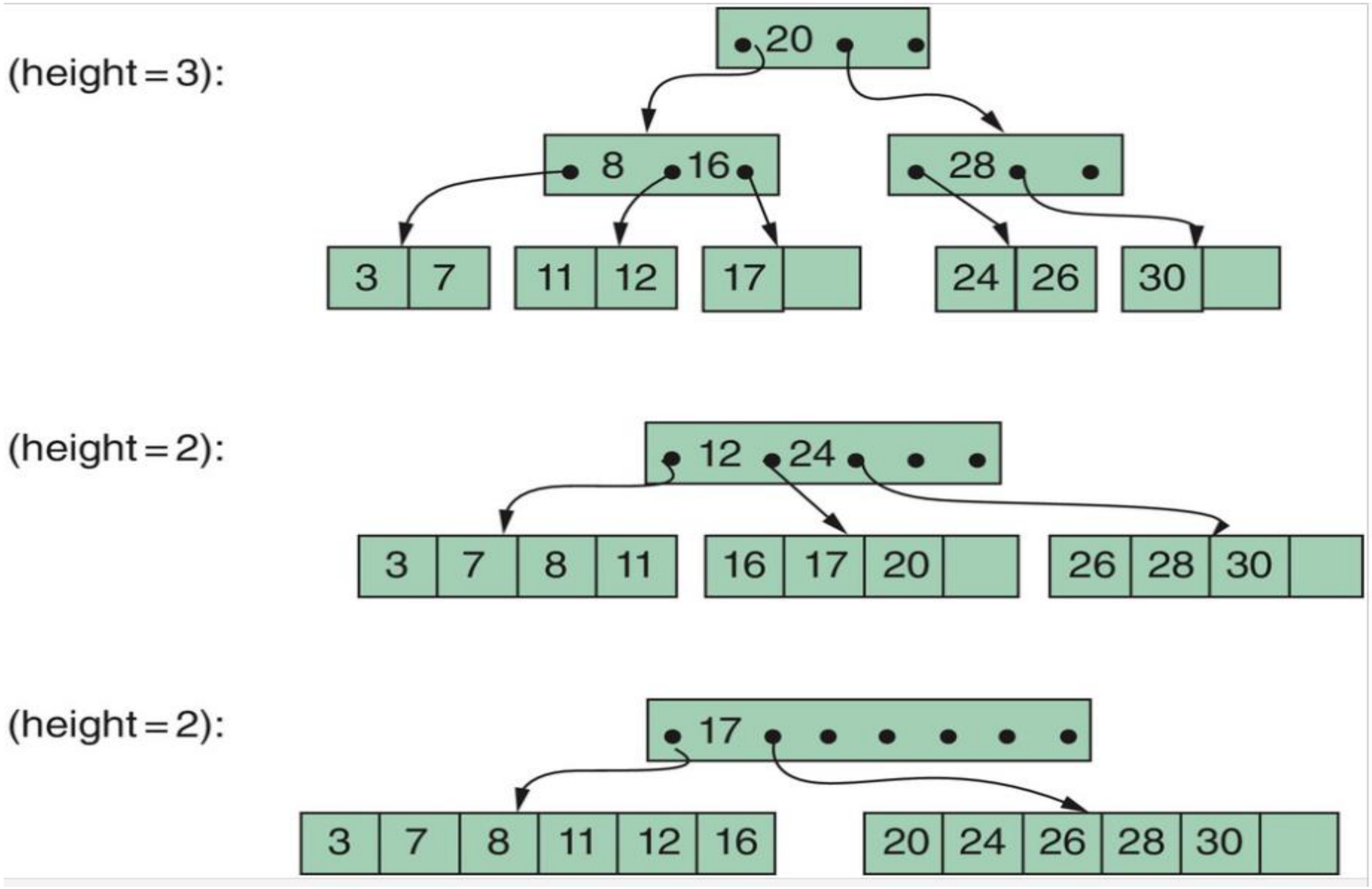
Types of Multilevel Index (Tree Index)

- There are many different types of multilevel indexes.
  - B-tree
  - B+tree
  - B*tree
  - Quadtree
  - R-tree
  - R+tree, etc.
- Rather than traditional multilevel indexes, B-tree and B+-tree index structures are popularly used for traditional data types (such as char, varchar, date, number) in many commercial database products.

B-Tree

- A **B-tree** is a variation of search trees, where
  - each node corresponds to a disk block and
  - nodes are kept between half full and full to cater for accommodating changes in the data file without the need for too extensive rearrangements of the index.
  - Every node contains a set of search key values, a set of tree points that refer to child nodes, and a set of data points that refer to data records or blocks with data records, that correspond to the search key values.
- A B-tree is a **balanced tree**; all leaf nodes are at the same level in the tree.

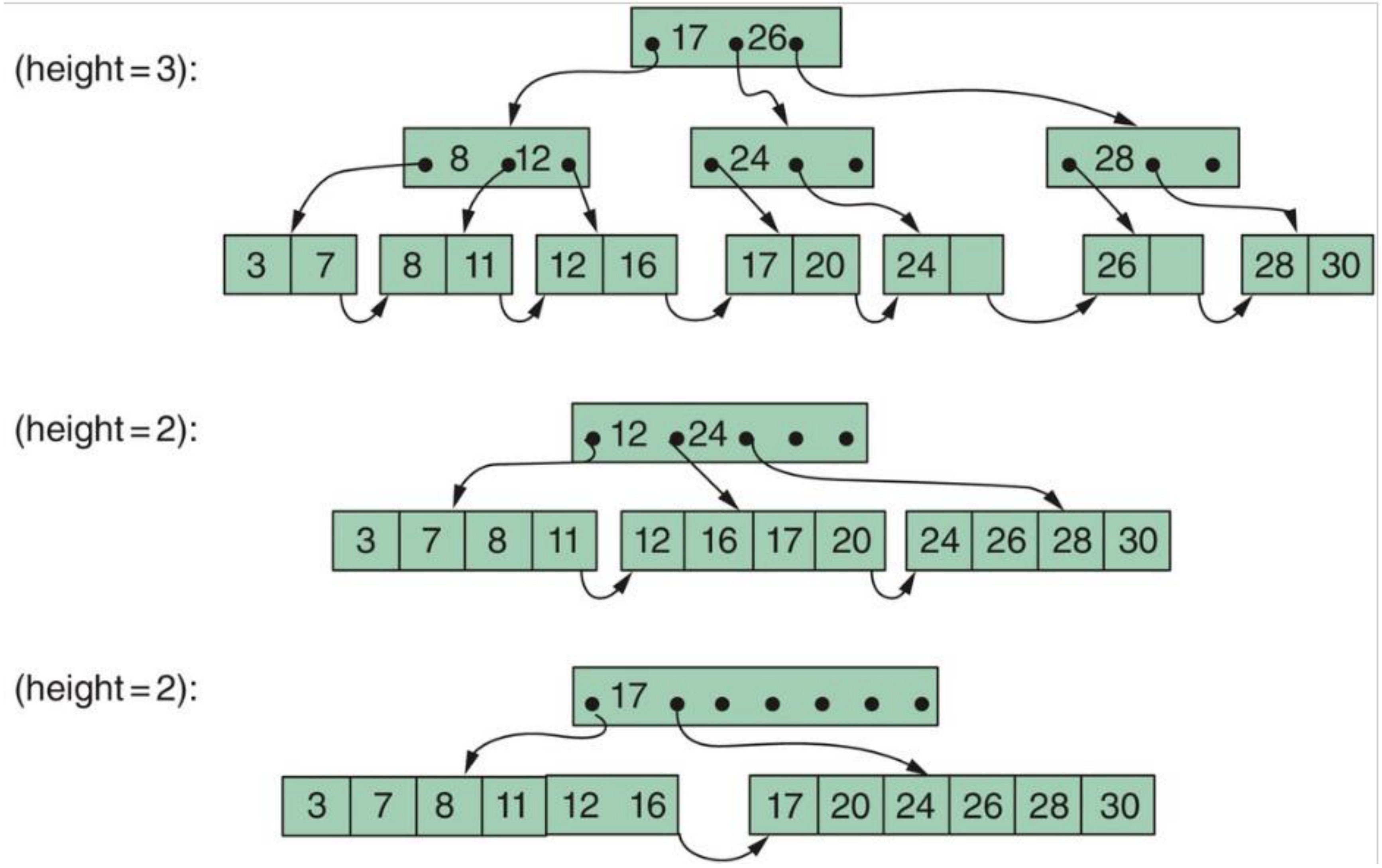
Example of B-Trees



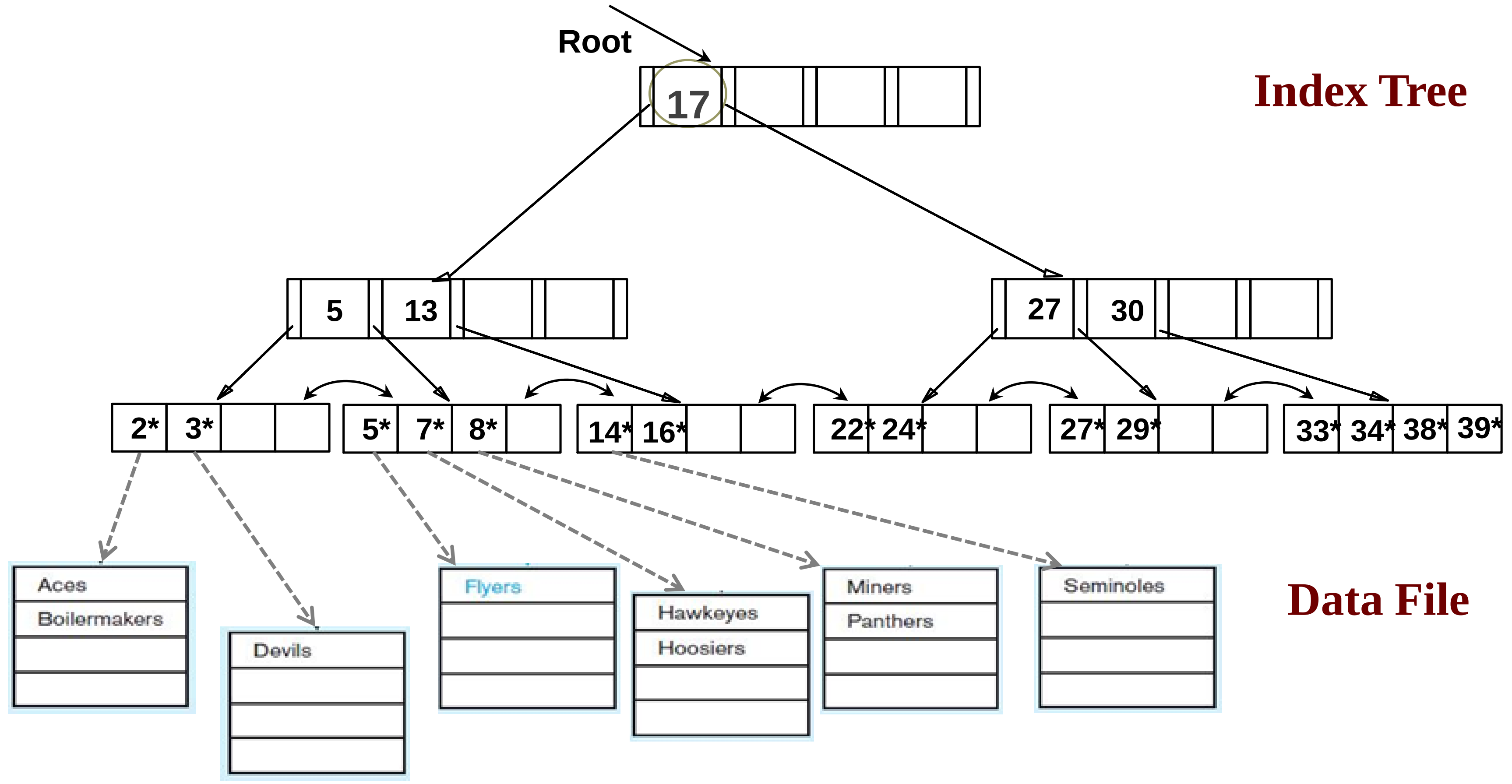
B+-Trees

- Most DBMS implementations use indexes based on **B+-trees** rather than B-trees.
- In a B+-tree,
  - Only the leaf nodes contain data points.
  - All key values that exist in the non-leaf nodes are repeated in the leaf nodes, such that every key value occurs in a leaf node, along with a corresponding data pointers.
  - Every leaf node also has one tree pointer, pointing to its next sibling or two tree pointers, pointing to its previous and next siblings

Example of B+-Trees



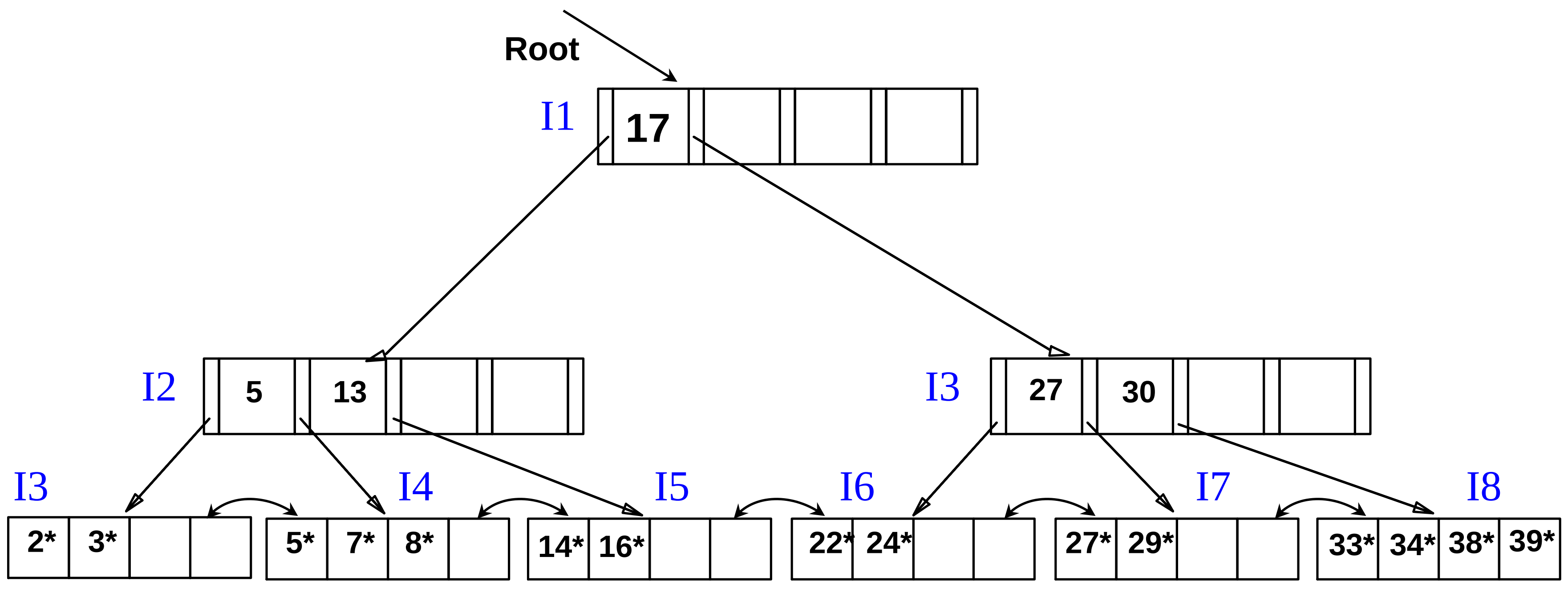
Example: B+Tree Index with Data File



B+ Tree: Search

- A B+-tree is searched starting from the root.
- A tree pointer is followed to the subtree that contains the appropriate range of key values.
- The same procedure is repeated for this subtree until a leaf node is reached.
- By accessing several leaf nodes consecutively, a range of multiple search key values can be retrieved.

Example of B+ Tree Search



- Find 5*? 29*?
- Find All > 15* and < 30*

Characteristics of B+Trees

- Most widely used tree index
- More flexible and dynamic structure that adjust inserts and deletes
- Height of B+-tree is often smaller, resulting in less block accesses to search
- Non-leaf nodes do not contain data points
- Leaf nodes have an entry for every value of the search field
- Tree is always balanced
  - keep tree *height-balanced*
- Leaf nodes are linked using (tree) page pointers.
- Supports equality and range-searches efficiently.

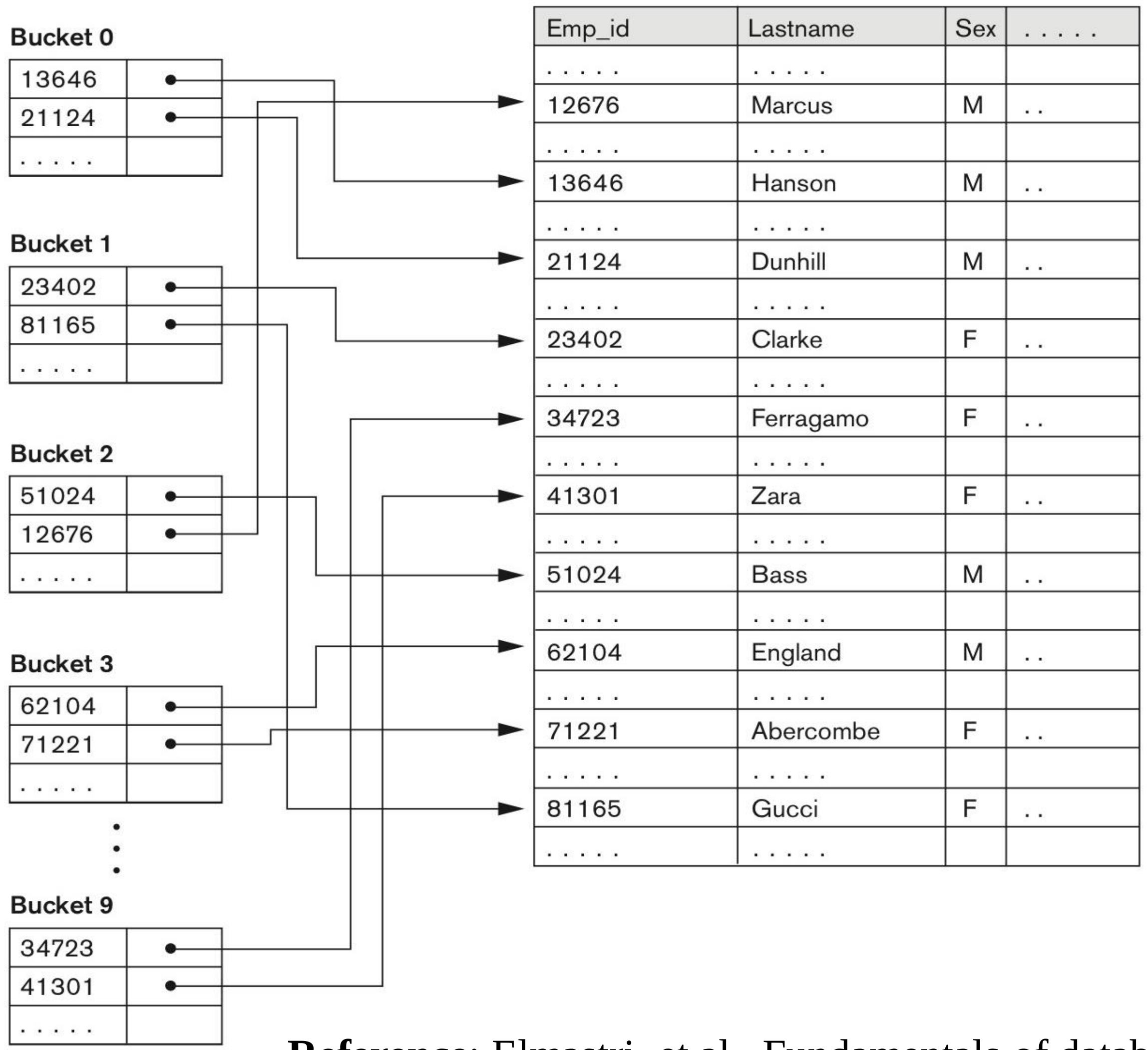
Outline

- Storage Hardware and Physical Databases
- Record Organization
- File Organization
  - Primary/Secondary File Organization Methods
  - Heap File Organization
  - Sequential File Organization
  - Hash File Organization (Random File Organization)
  - Indexed Sequential File Organization
  - ☞ **Other Index Types**

Hash Indexes

- **Hash indexes** provide a secondary file organization method that combines hashing with indexed retrieval.
- The index entries have the same format as in a normal secondary index, <key value, point> pairs.
- The index is organized not as a sequential file, but a hash file.
- Applying the hash function to the search key yields the index block where the corresponding index entry can be found.
- Based on the pointer in this entry, the actual record(s) can be retrieved.

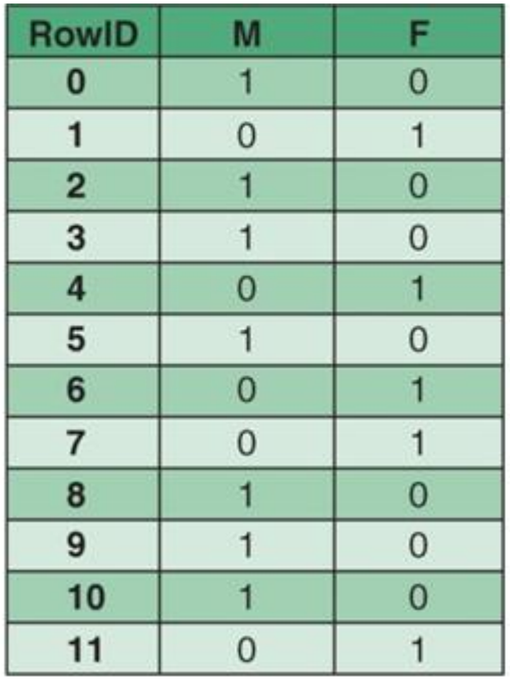
Example of Hash-based Index



Reference: Elmastri, et al., Fundamentals of database Systems, Figure 17.15

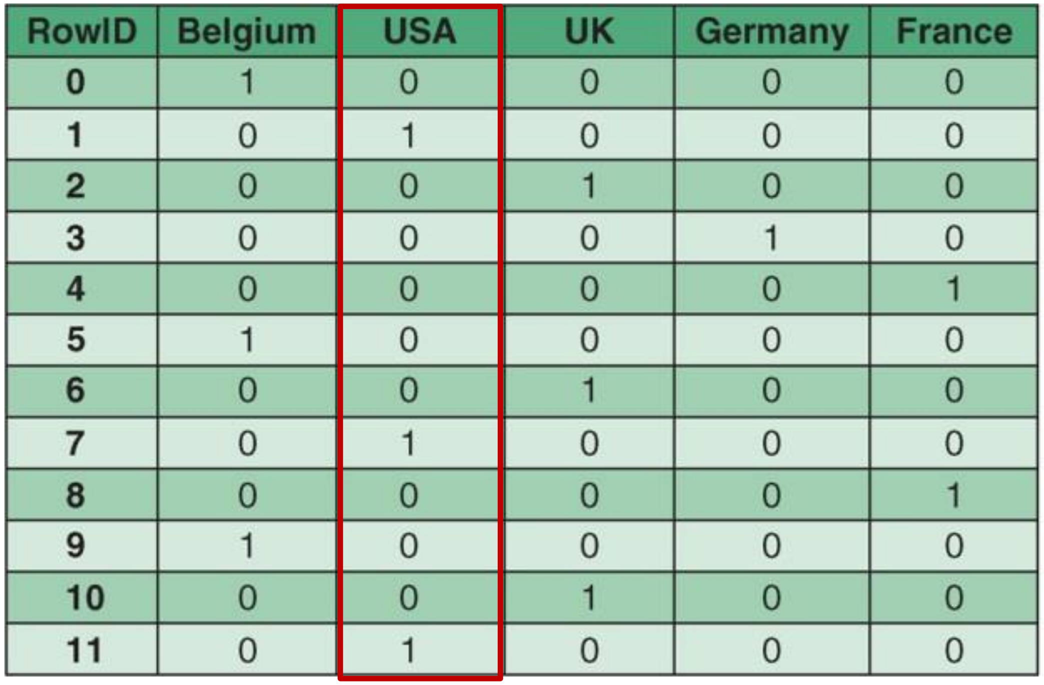
Bitmap Index

- **Bitmap index** are mostly efficient for attribute types with only limited set of values
- Instead of these values, bitmap indexes contain a row ID and a series of bits—one bit for each possible value of indexed attribute type
- For each entry, the bit position that corresponds to the actual value for the row at hand is set to 1.
- The row IDs can be mapped to record pointers
- By applying **Boolean operations** to bit vectors from multiple bitmap indexes, it becomes very efficient to identify records that satisfy certain criteria

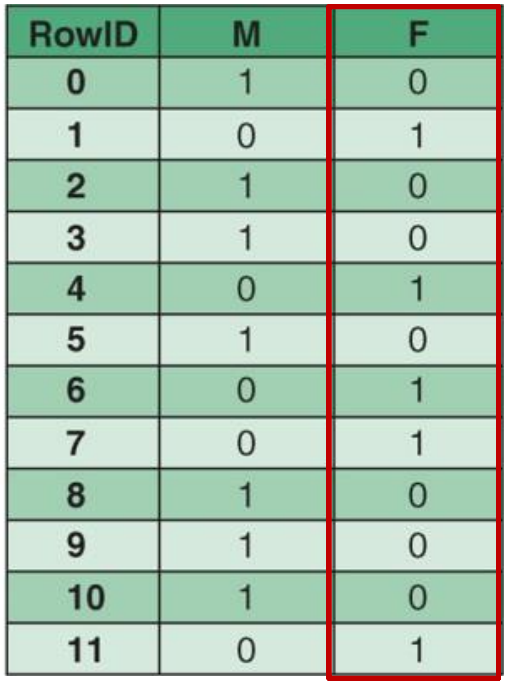


RowID	M	F
0	1	0
1	0	1
2	1	0
3	1	0
4	0	1
5	1	0
6	0	1
7	0	1
8	1	0
9	1	0
10	1	0
11	0	1

Example: Bitmap Indexes



RowID	Belgium	USA	UK	Germany	France
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1
5	1	0	0	0	0
6	0	0	1	0	0
7	0	1	0	0	0
8	0	0	0	0	1
9	1	0	0	0	0
10	0	0	1	0	0
11	0	1	0	0	0



RowID	M	F
0	1	0
1	0	1
2	1	0
3	1	0
4	0	1
5	1	0
6	0	1
7	0	1
8	1	0
9	1	0
10	1	0
11	0	1

Figure. Bitmap indexes for the "Country" and "Gender" attribute types

- In each table, each column can be considered as a bitmap or bit vector indicating which tuples have the values indicated by the column.
- By applying Boolean operations to bit vectors from multiple bitmap indexes, we can efficiently identify records such as female customers who live in the U.S.
  ➔ **Record 1, 7, 11**

Conclusion

- Storage Hardware and Physical Database Design
- Record Organization
- File Organization
  - Primary/Secondary File Organization Methods
  - Heap File Organization
  - Sequential File Organization
  - Hash File Organization (Random File Organization)
  - Indexed Sequential File Organization
  - Other Indexes - Hash Index, Bitmap Index