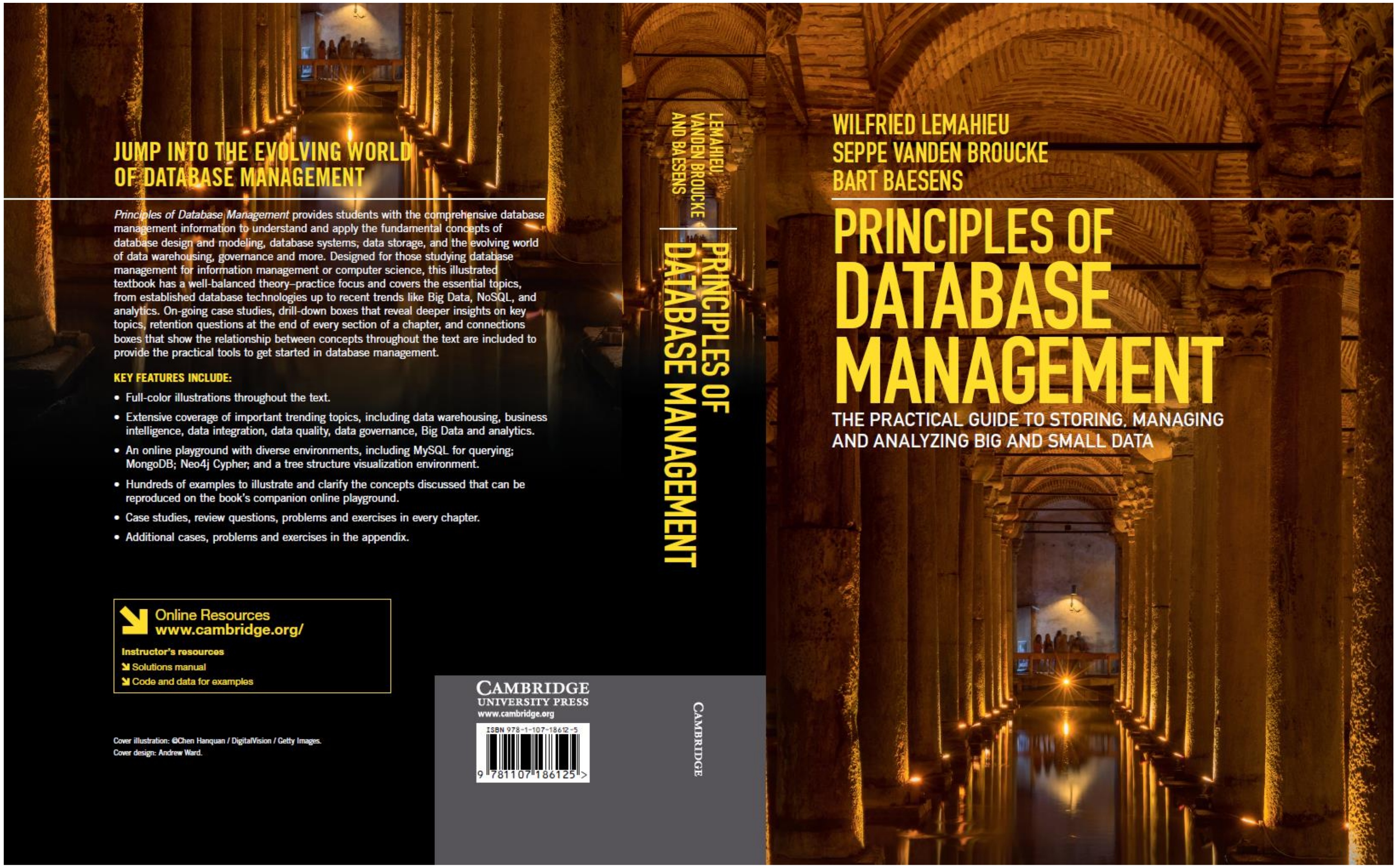
Physical File Organization and Indexing



JUMP INTO THE EVOLVING WORLD OF DATABASE MANAGEMENT

Principles of Database Management provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

KEY FEATURES INCLUDE:

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
www.cambridge.org/

Instructor's resources
- Solutions manual
- Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

ISBN 978-1-107-18612-5

LEMAHIEU
VANDEN BROUCKE
AND BAESENS

PRINCIPLES OF DATABASE MANAGEMENT

CAMBRIDGE

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

PRINCIPLES OF DATABASE MANAGEMENT

THE PRACTICAL GUIDE TO STORING, MANAGING AND ANALYZING BIG AND SMALL DATA

www.pdbmbook.com

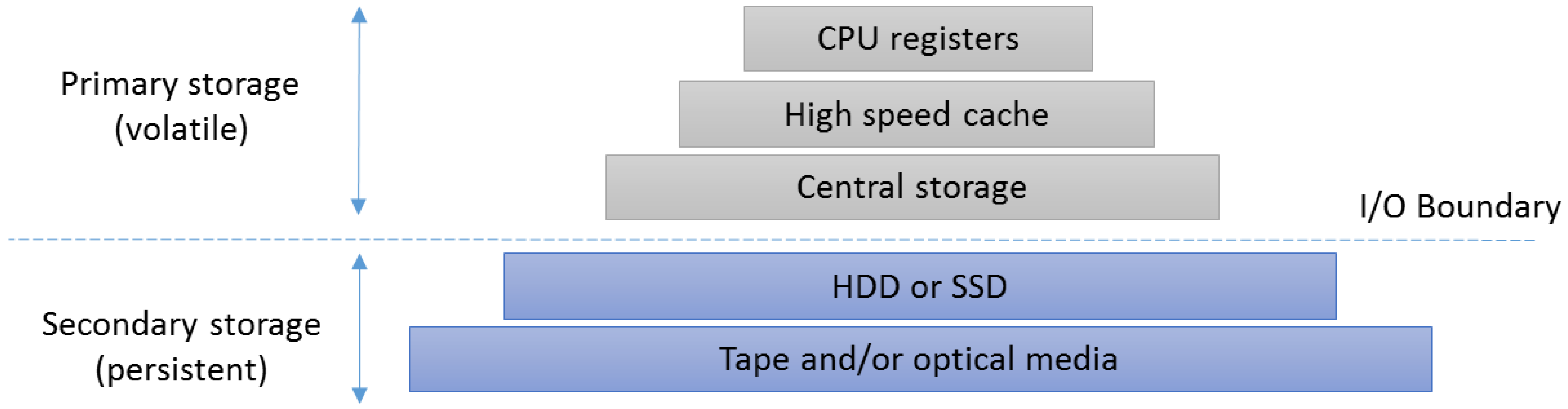
Introduction

- Storage Hardware and Physical Database Design
- Record Organization
- File Organization

Storage Hardware and Physical Database Design

- The Storage Hierarchy
- Internals of Hard Disk Drives
- From Logical Concepts to Physical Constructs

The Storage Hierarchy



- Computer memory hierarchy
  - high speed memory, expensive and limited in capacity at the top
  - slower memory, relatively cheap and larger in size at the bottom

The Storage Hierarchy

- Primary Storage (a.k.a. volatile memory)
  - Central Processing Unit (CPU): executes mathematical and logical processor operations
  - cache memory operates at nearly same speed as CPU
  - central storage (a.k.a. internal memory, main memory): consists of memory chips (also called Random Access Memory, or RAM) of which the performance is expressed in nanoseconds
  - contains database buffer and runtime code of the applications and DBMS

The Storage Hierarchy

- Secondary Storage
  - persistent storage media
  - Hard disk drive (HDD) and solid state drives (SSD) based on flash memory
  - contains physical database files

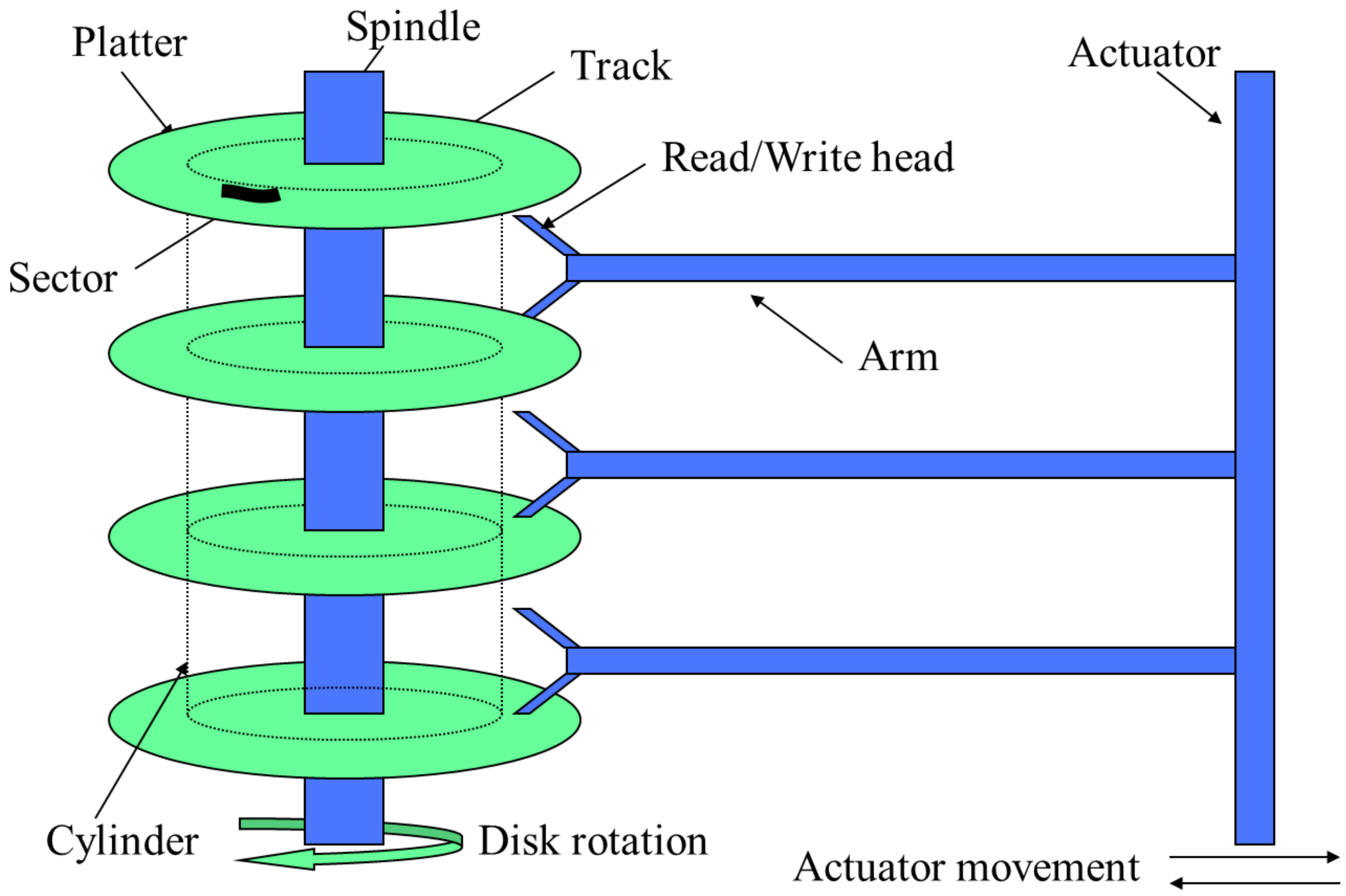
The Storage Hierarchy

- Primary and secondary storage divided by 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
- Still lower in the hierarchy: optical drives (e.g., rewritable DVD, Blu-ray) and tape
- In what follows: hard disk drive as the storage medium!

Internals of Hard Disk Drives

- Hard Disk Drive (HDD) stores data on circular platters, which are covered with magnetic particles
- A HDD also contains a hard disk controller
- HDDs are directly accessible storage devices (DASDs)
- Platters are secured on a spindle, which rotates at a constant speed
- Read/write heads can be positioned on arms, which are fixed to an actuator

Internals of Hard Disk Drives



Internals of Hard Disk Drives

- By combining disk rotation with actuator movement, each individual section of the disk is directly reachable
- 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

Internals of Hard Disk Drives

- 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, latency)
- Transfer time depends on block size, density of magnetic particles and rotation speed of disks
- Response time = service time + queueing time
- Service time = seek time + rotational delay + transfer time

Internals of Hard Disk Drives

- Physical file organization can be optimized to minimize expected seek time and rotational delay
- T_{rba} refers to expected time to retrieve/write disk block independently of previous read/write: T_{rba} = Seek + ROT/2 + BS/TR
- T_{sba} refers to expected time to sequentially retrieve disk block with R/W head already in correct position: T_{sba} = ROT/2 + BS/TR
- Note: block size (BS), rotation time (ROT) and transfer rate (TR)

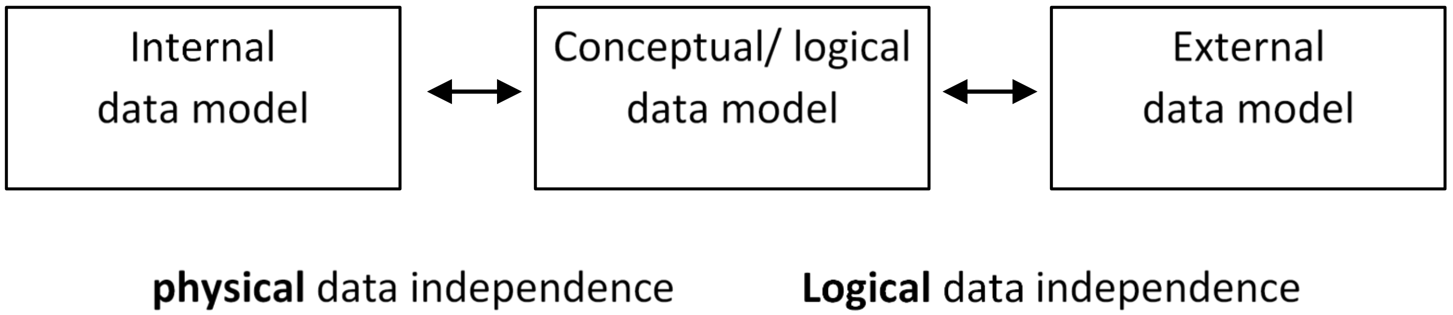
Internals of Hard Disk Drives

Average seek time	8.9 ms
Spindle speed	7200 rpm
Transfer rate	150 MBps
Block size	4096 bytes

- T_{rba} = 8.9 ms + 4.167 ms + 0.026 ms = 13.093 ms
- T_{sba} = 4.167 ms + 0.026 ms = 4.193 ms

From Logical Concepts to Physical Constructs

- Physical database design: translate logical data model into internal data model (a.k.a. physical data model)
- Trade-off between efficient update/retrieval and efficient use of storage space
- Focus on physical organization of structured, relational data!

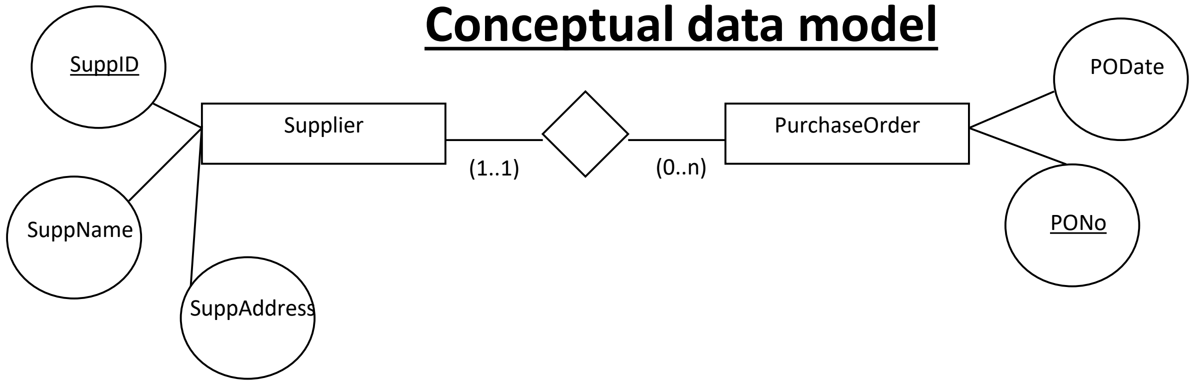

Internal data model
Conceptual/ logical data model
External data model
physical data independence
Logical data independence

From Logical Concepts to Physical Constructs

Logical data model **(general terminology)**	**Logical data model** **(relational setting)**	**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

From Logical Concepts to Physical Constructs

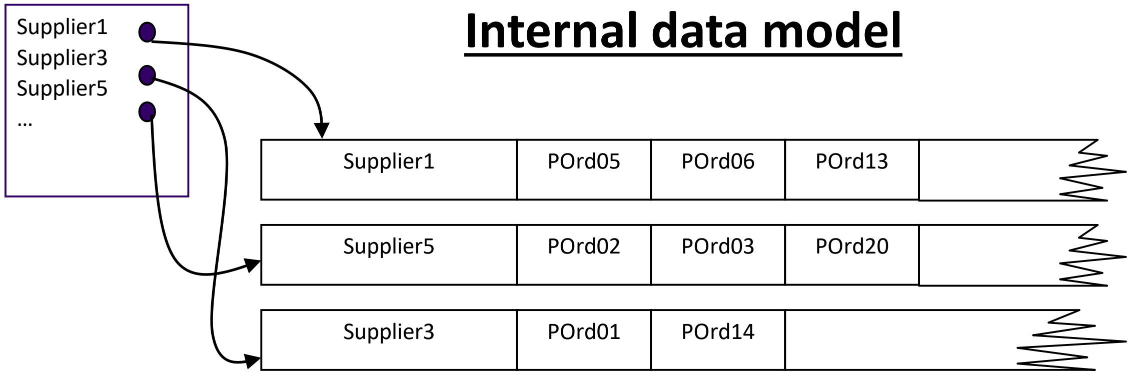
Conceptual data model



Logical data model

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

Internal data model



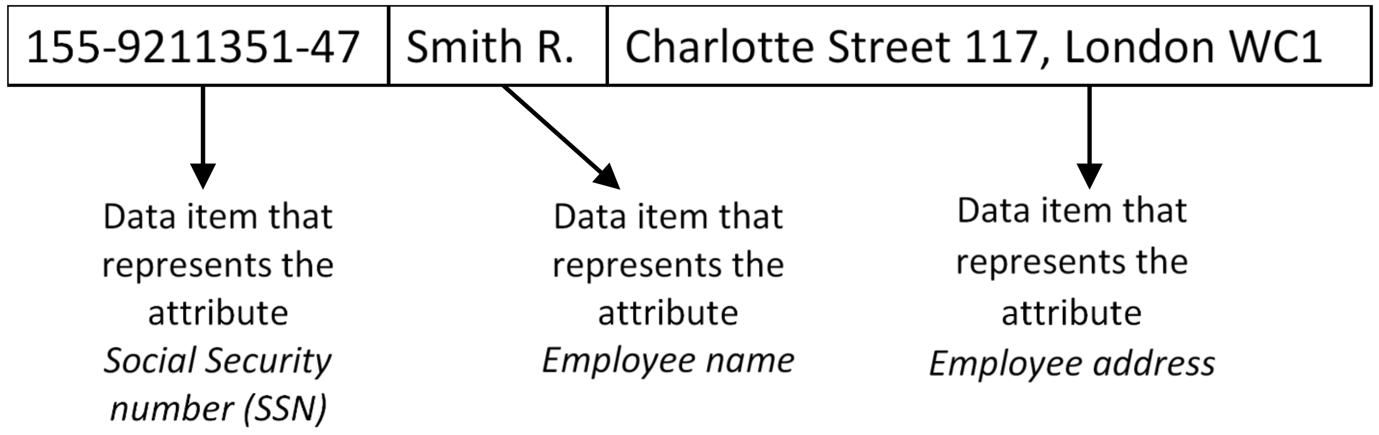
Record Organization

- Record organization refers to organization of data items into stored records
- Physical implementation of data item is a series of bits
- Common techniques
  - relative location
  - embedded identification
  - pointers and lists

Record Organization

- Relative Location
  - simplest and most widespread
  - data items that represent attributes of same entity are stored on physically adjacent addresses
  - attribute types determined by relative ordering

Record Organization


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*

```
CREATE TABLE EMPLOYEE
          (SSN …
           EMPLOYEE NAME …
           EMPLOYEE ADDRESS …
           …
          );
```

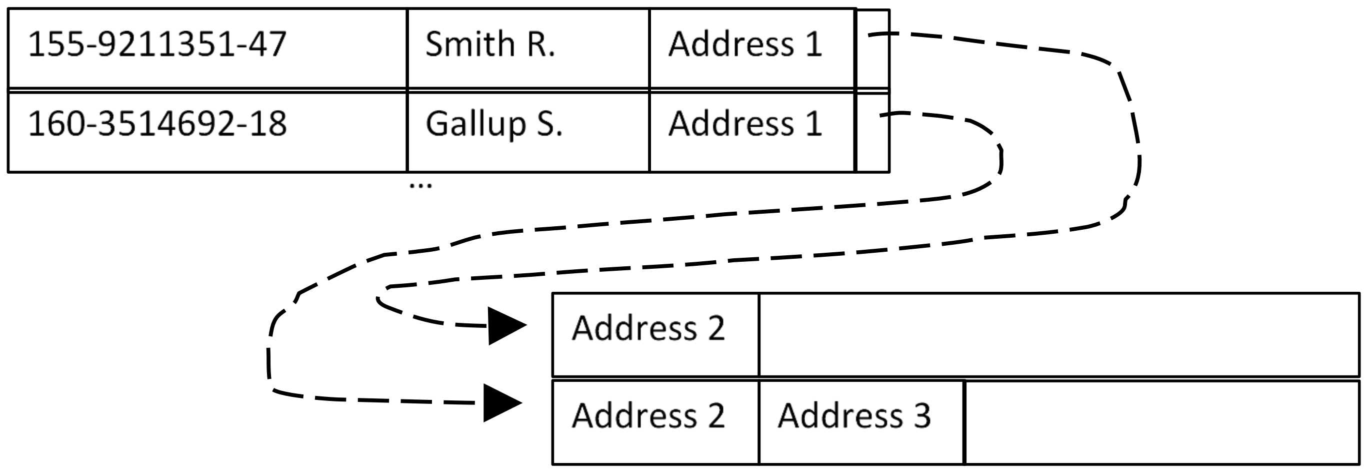
Record Organization

- Embedded Identification
  - data items representing attributes always preceded by attribute type
  - only non-empty attributes of record included
  - missing attributes not a problem and no need to store attributes in fixed order to identify them
  - similar to XML and JSON

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

Record Organization

- Pointers and Lists
  - ideal for dealing with variable length records (due to e.g. variable length data type, multivalued attribute type, optional attribute type, etc.)



Record Organization

- Blocking factor (BF) indicates how many records are stored in single disk block
- For a file with fixed length records, BF is calculated as: $BF = \lfloor BS/RS \rfloor$
- For variable length records, BF denotes the average number of records in a block
- Blocking factor determines how many records are retrieved with a single read operation

File Organization

- Introductory Concepts
- Heap File Organization
- Sequential File Organization
- Random File Organization (Hashing)
- Indexed Sequential File Organization
- List Data Organization
- Secondary Indexes and Inverted Files
- B-trees and B^+-trees

Introductory Concepts

- Search key: single attribute type, or set of attribute types, whose values determine criteria according to which records are retrieved
  - 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

Introductory Concepts

- Primary file organization methods: determine physical positioning of stored records on storage medium
  - E.g., heap files, random file organization, indexed sequential file organization
  - can only be applied once
- Linear search: each record in file is retrieved and assessed against search key
- Hashing and indexing: primary techniques that specify relationship between record’s search key and physical location

Introductory Concepts

- Secondary file organization methods: provide constructs to efficiently retrieve records according to search key that was not used for primary file organization
  – based on secondary index

Heap File Organization

- Basic primary file organization method
- New records inserted at end of file
- No relationship between record's attributes and physical location
- Only option for record retrieval is linear search
- For a file with NBLK blocks, it takes on average NBLK/2 sba to find record according to unique search key
- Searching records according to non-unique search key requires scanning entire file

Sequential File Organization

- Records stored in ascending/descending order of search key
- Efficient to retrieve records in order determined by search key
- Records can still be retrieved by means of linear search, but now a more effective stopping criterion can be used, i.e. once first higher/lower key value than required one is found

Sequential File Organization

- Binary search technique can be used
- For unique search key K, with values K_i, algorithm to retrieve record with key value K_μ
  - `Selection criterion: record with search key value` K_μ
  - `Set l = 1; h = number of blocks in file (suppose records are in ascending order of search key K)`
  - `Repeat until` $h \geq l$
    - `i = (l + h) / 2, rounded to nearest integer`
    - `Retrieve block i and examine key values` K_j `of records in block i`
  - `if any` $K_j = K_\mu \rightarrow$ `record is found!`
  - `else if` $K_\mu >$ `all` $K_j \rightarrow$ `continue with l = i+1`
  - `else if` $K_\mu <$ `all` $K_j \rightarrow$ `continue with h = i-1`
  - `else record is not in file`

Sequential File Organization

- Expected number of block accesses to retrieve record according to primary key by means of
  - linear search: NBLK/2 sba
  - binary search: $\log_2$(NBLK) rba

Sequential File Organization

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

- $BF = \lfloor BS/RS \rfloor = \lfloor 2048/100 \rfloor = 20$
- 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
- If binary search is used, expected number of block accesses is $\log_2(1500) \approx 11$ rba

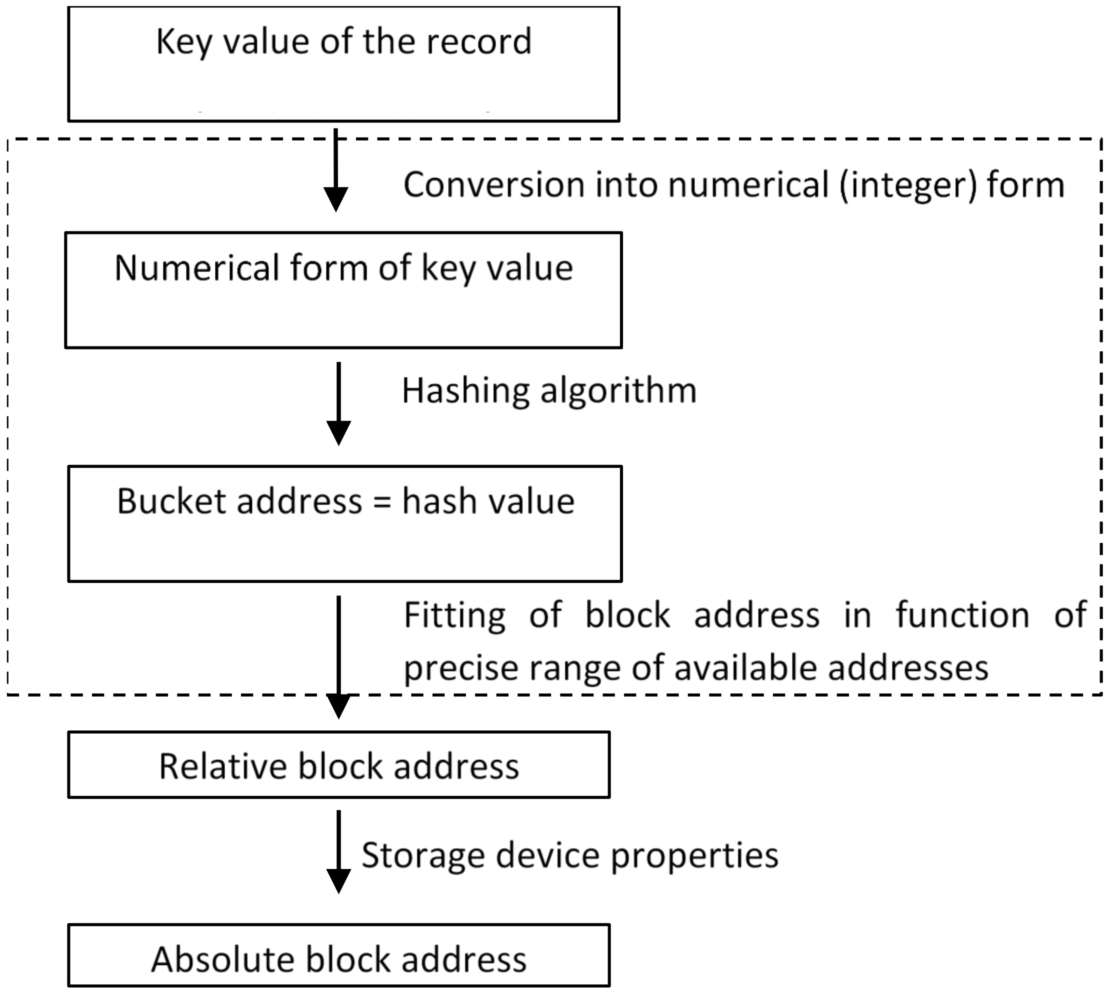
Sequential File Organization

- Updating sequential file is more cumbersome than updating heap file
  - often done in batch
- Sequential files often combined with one or more indexes (see later, indexed sequential file organization)

Random File Organization (Hashing)

- Random file organization (a.k.a. direct file organization, hash file organization) assumes direct relationship between value of search key and physical location
- Hashing algorithm defines key-to-address transformation
  - generated addresses pertain to bucket (contiguous area of record addresses)
- Most effective when using primary key or other candidate key as search key

Random File Organization (Hashing)


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

Random File Organization (Hashing)

- Hashing cannot guarantee that all keys are mapped to different hash values, hence bucket addresses
- Collision occurs when several records are assigned to same bucket (also called synonyms)
- If more synonyms than slots for a bucket, bucket is in overflow
  - additional block accesses needed to retrieve overflow records
- Hashing algorithm should distribute keys as evenly as possible over the respective bucket addresses

Random File Organization (Hashing)

- Popular hashing technique is division: address(key_i) = key_i mod M
  - M is often a prime number (close to, but a bit larger than, the number of available addresses)

Random File Organization (Hashing)

	Series 1			Series 2	
Key value	Division by 20	Division by 23	Key value	Division by 20	Division by 23
3000	00	10	3000	00	10
3001	01	11	3025	05	12
3002	02	12	3050	10	14
3003	03	13	3075	15	16
3004	04	14	3100	00	18
3005	05	15	3125	05	20
3006	06	16	3150	10	22
3007	07	17	3175	15	01
3008	08	18	3200	00	03
3009	09	19	3225	05	05
3010	10	20	3250	10	07
3011	11	21	3275	15	09
3012	12	22	3300	00	11

	Series 1			Series 2	
Key value	Division by 20	Division by 23	Key value	Division by 20	Division by 23
3013	13	00	3325	05	13
3014	14	01	3350	10	15
3015	15	02	3375	15	17
3016	16	03	3400	00	19
3017	17	04	3425	05	21
3018	18	05	3450	10	00
3019	19	06	3475	15	02

Random File Organization (Hashing)

- Efficiency of hashing algorithm measured by expected number of rba and sba
- Retrieving non-overflow record:
  - 1 rba to first block of bucket denoted by hashing algorithm, possibly followed by 1 or more sba
- Additional block accesses needed for overflow record depending on percentage of overflow records and overflow handling technique

Random File Organization (Hashing)

- Percentage of overflow records depends on hashing algorithm and key set
- Aim to achieve uniform distribution, spreading the set of records evenly over the set of available buckets
- Required number of buckets NB: $NB = \lceil NR / (BS \times LF) \rceil$ with NR number of records, BS bucket size and LF loading factor
- Trade-off: larger bucket size implies smaller chance of overflow, but more additional overhead to retrieve non-overflow records

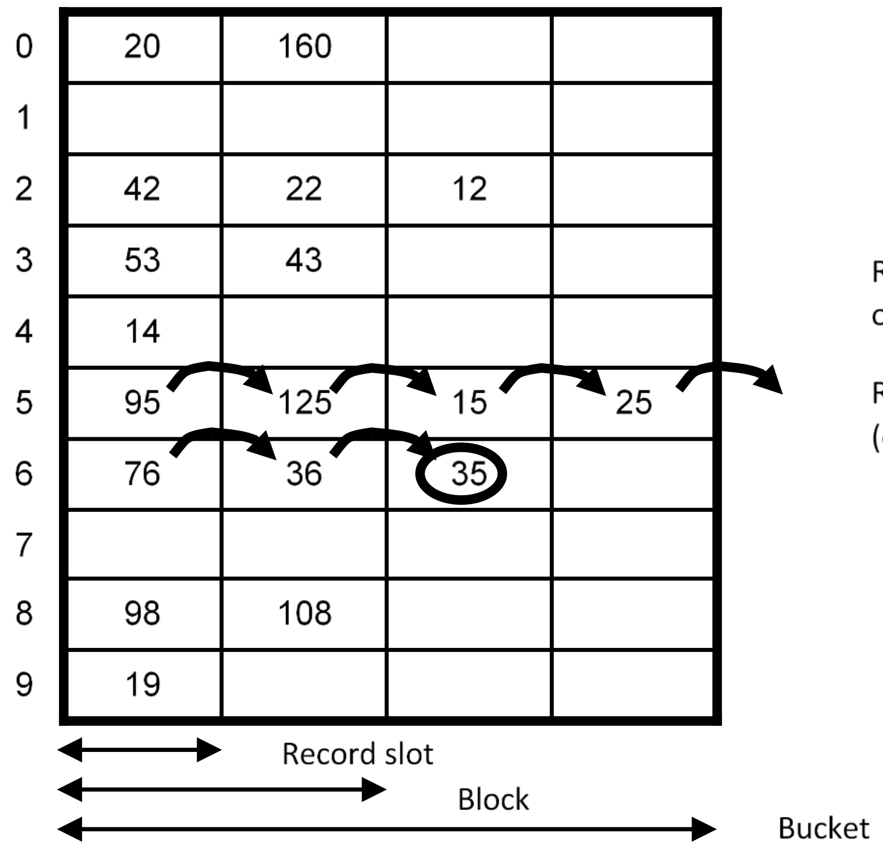
Random File Organization (Hashing)

- Loading factor (LF) represents average number of records in bucket divided by bucket size
  - indicates how ‘full’ every bucket is on average
  - embodies trade-off between efficient use of storage capacity and retrieval performance
  - often set between 0.7 and 0.9
- Different overflow handling techniques
  - overflow records stored either in primary area or in separate overflow area

Random File Organization (Hashing)

- Open addressing
  - overflow records stored in next free slot after full bucket where record would normally have been stored

Hash = key mod 10
LF = 0,45
Bucket size = 4
BF = 2
Open addressing



Required block accesses for retrieving non-overflow records: 1 rba or (1 rba + 1 sba)

Required block accesses for retrieving (overflow) record with ID 35: (1 rba + 3 sba)

Random File Organization (Hashing)

- Chaining
  - overflow records stored in separate overflow area, with subsequent records that overflow from same bucket being chained together by pointers (linked list)
  - Pro: no cluttering of primary area, no additional overflow
  - Con: results in additional rba
- Note: dynamic hashing techniques allow for file to shrink or grow without need for rearranging

Indexed Sequential File Organization

- Random file organization is efficient to retrieve individual records by search key value
- Sequential File Organization is efficient if many records are to be retrieved in certain order
- Indexed Sequential File organization method reconciles both concerns
- Indexed Sequential File organization combines sequential file organization with one or more indexes

Indexed Sequential File Organization

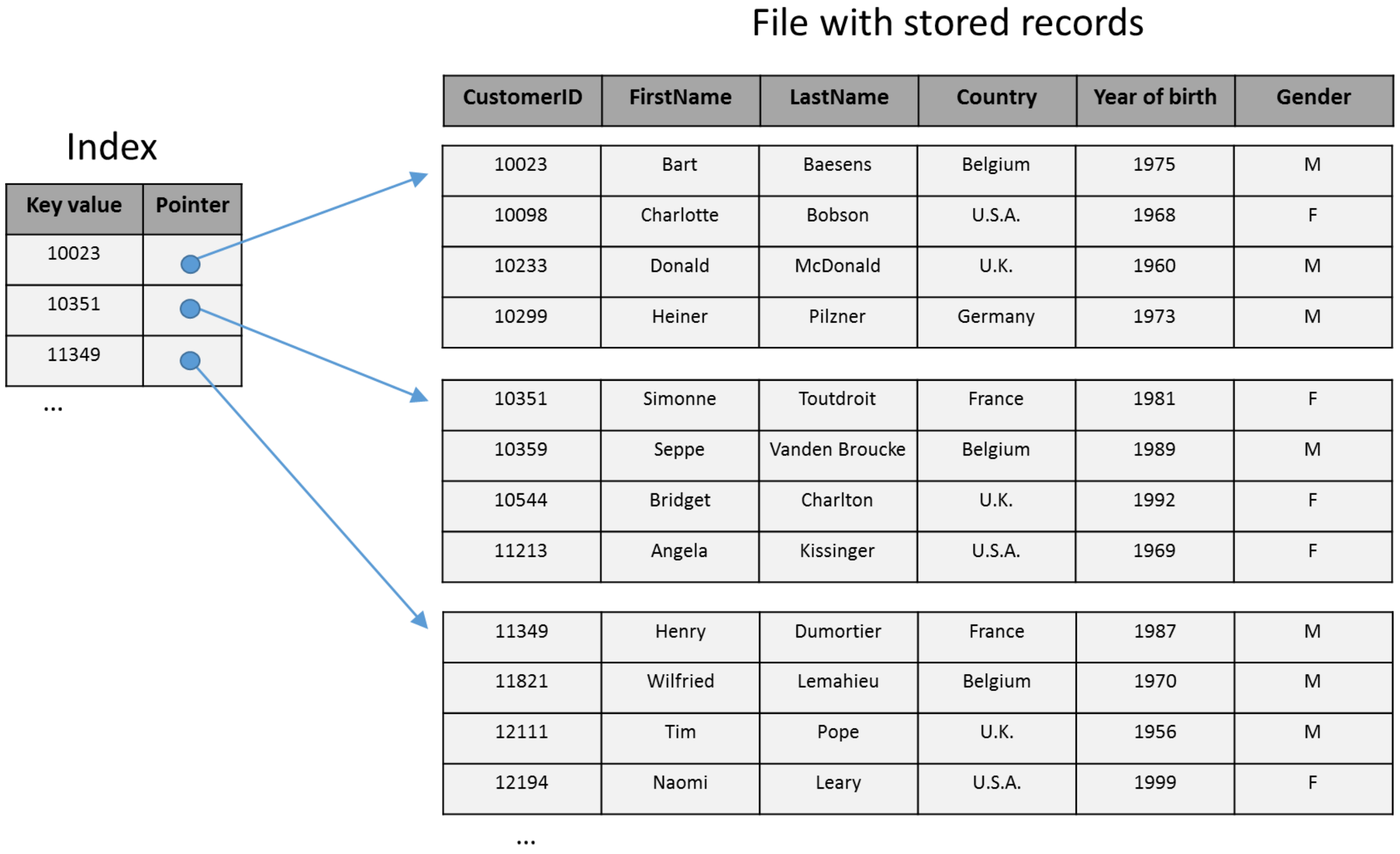
- File is divided into intervals or partitions
- Each interval is represented by index entry containing search key value of first record in interval and pointer to physical position of first record in interval
- Pointer can be block pointer (referring to physical block address) or record pointer (consisting of combination of block address and record id or offset within block)
- Index is sequential file, ordered according to search key values with entries: <search key value, block pointer or record pointer>
- Search key can be atomic (e.g., a CustomerID) or composite (e.g. Year of Birth and Gender)

Indexed Sequential File Organization

- Dense index has index entry for every possible value of search key
- Sparse index has index entry for only some of search key values
- Dense indexes are generally faster, but require more storage space and are more complex to maintain than sparse indexes
- Note: index file occupies fewer disk blocks than data file and can be searched much quicker

Indexed Sequential File Organization

- With primary index file organization, data file is ordered on unique key and index is defined over this unique search key



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	U.S.A.	1968	F
10233	Donald	McDonald	U.K.	1960	M
10299	Heiner	Pilzner	Germany	1973	M
10351	Simonne	Toutdroit	France	1981	F
10359	Seppe	Vanden Broucke	Belgium	1989	M
10544	Bridget	Charlton	U.K.	1992	F
11213	Angela	Kissinger	U.S.A.	1969	F
11349	Henry	Dumortier	France	1987	M
11821	Wilfried	Lemahieu	Belgium	1970	M
12111	Tim	Pope	U.K.	1956	M
12194	Naomi	Leary	U.S.A.	1999	F

...

Indexed Sequential File Organization

Linear search	NBLK sba
Binary search	$\log_2$(NBLK) rba
Index based search	$\log_2$(NBLKI) + 1 rba, with NBLKI << NBLK

Note: NBLKI represents number of blocks in index!

Indexed Sequential File Organization

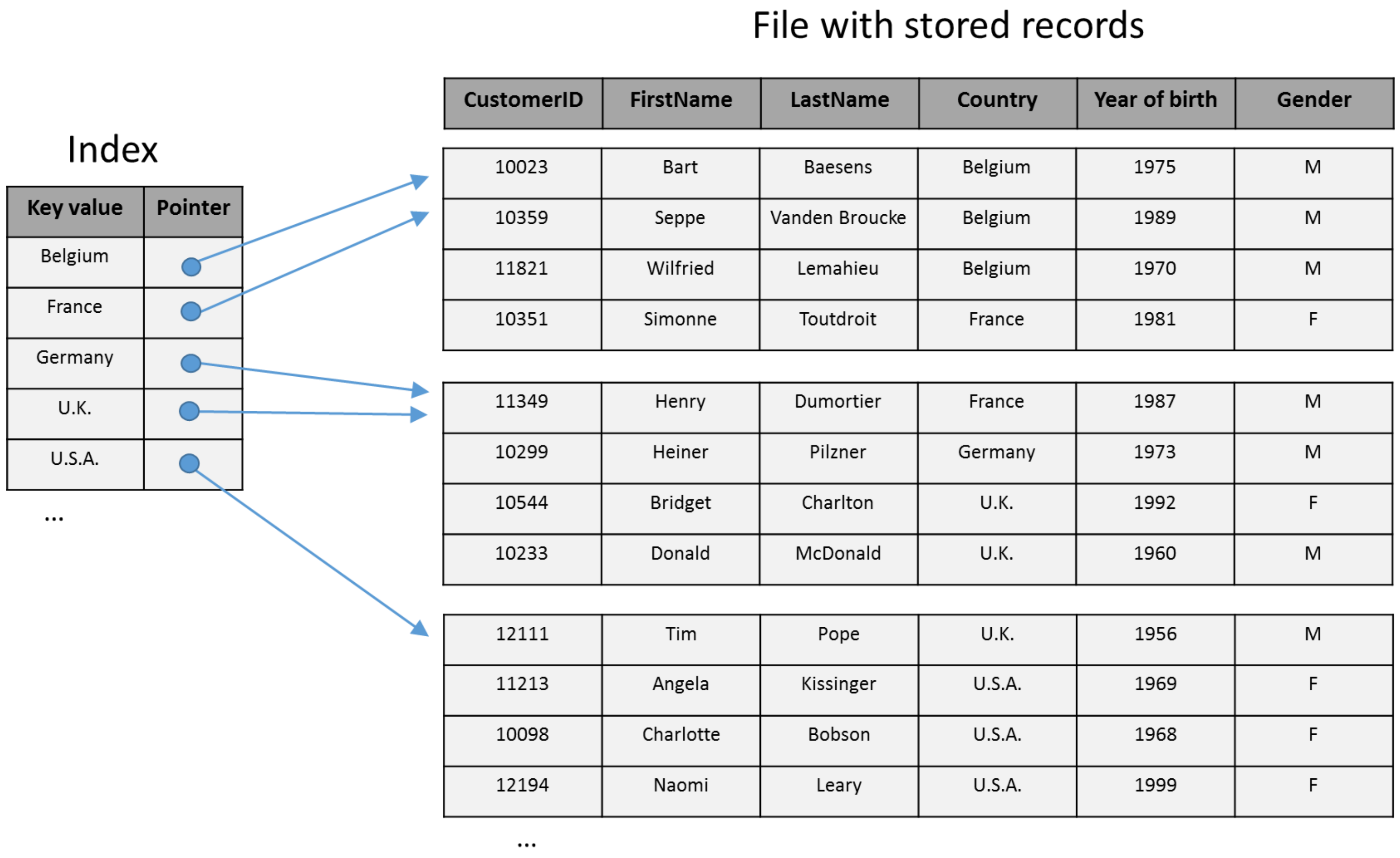
Number of records (NR)	30000
Block size (BS)	2048 bytes
Records size (RS)	100 bytes
Index entry	15 bytes

- Blocking factor of index = $\lfloor 2048/15 \rfloor$=136
- NBLKI = $\lceil 1500/136 \rceil$ = 12 blocks
- Binary search on index requires $\log_2(12) + 1 \approx 5$ rba (compare to 750 sba and 11 rba!)

Indexed Sequential File Organization

- Clustered index is similar to primary index, but ordering criterion and search key, is non-key attribute type or set of attribute types
- Can be dense or sparse
- Search process is same as with primary index, except additional sba may be required after first rba to data file, to retrieve all subsequent records with same search key value

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

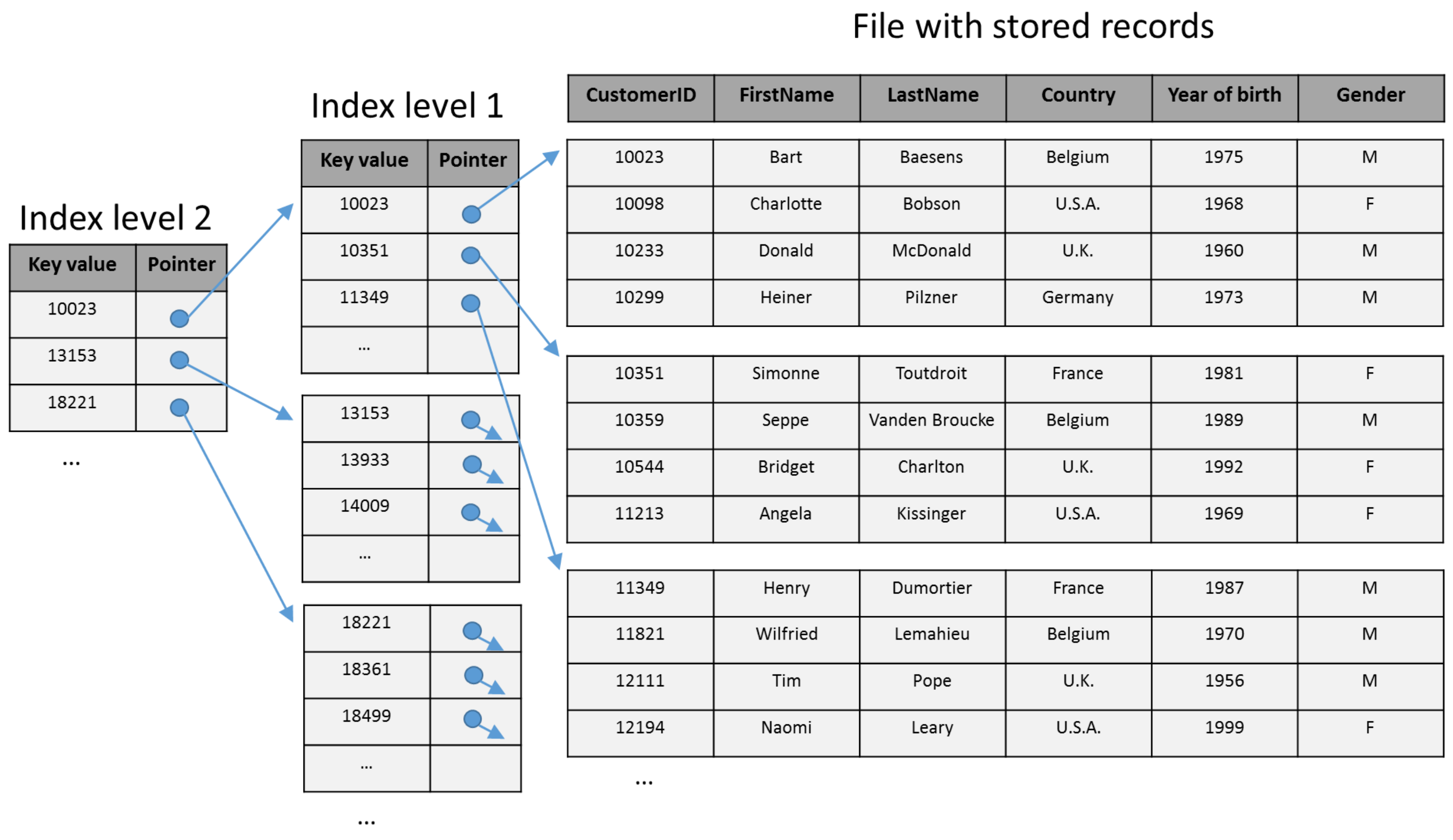
...

Indexed Sequential File Organization

- Similar to primary indexes, clustered indexes assume keeping index up to date if records are inserted or deleted or if search key value is updated
- Options:
  - start new block for every new value of search key
  - provide separate overflow section for records that cannot be stored in appropriate position in regular sequential file

Indexed Sequential File Organization

- Creating index-to-an-index results in multilevel indexes



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	U.S.A.	1968	F
10233	Donald	McDonald	U.K.	1960	M
10299	Heiner	Pilzner	Germany	1973	M
10351	Simonne	Toutdroit	France	1981	F
10359	Seppe	Vanden Broucke	Belgium	1989	M
10544	Bridget	Charlton	U.K.	1992	F
11213	Angela	Kissinger	U.S.A.	1969	F
11349	Henry	Dumortier	France	1987	M
11821	Wilfried	Lemahieu	Belgium	1970	M
12111	Tim	Pope	U.K.	1956	M
12194	Naomi	Leary	U.S.A.	1999	F

...

Indexed Sequential File Organization

- Primary index and clustered index can never occur together since there is only one way to physically order a file
- Secondary indexes can be defined over other search keys
  - no impact on the physical ordering of records

List Data Organization

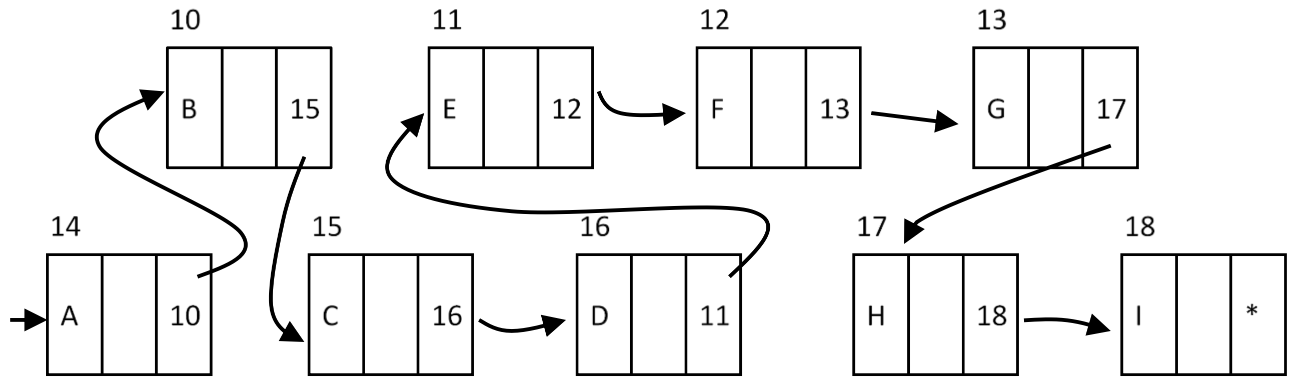
- A list can be defined as an ordered set of elements
- If each element has exactly one successor, except for the last element, we call it a linear list
  - all other types of lists are called nonlinear lists

List Data Organization

- Linear list embodies sequential data structure and can be represented in 2 ways:
  - sequential file organization method: logical ordering represented by means of physical contiguity
  - linked list: logical ordering represented using pointers

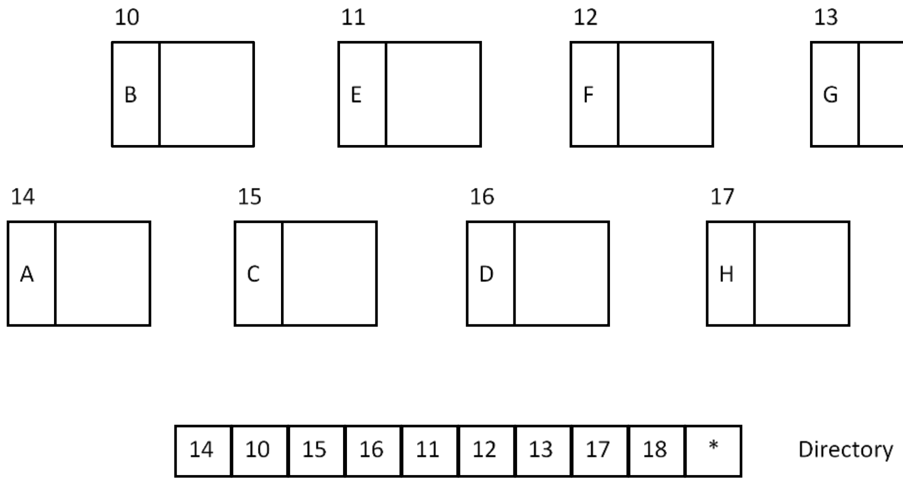
List Data Organization

- One-Way Linked List
  - records physically stored in arbitrary order, or sorted according to another search key
  - logical sequential ordering represented by means of pointers
  - often used to chain overflow records



List Data Organization

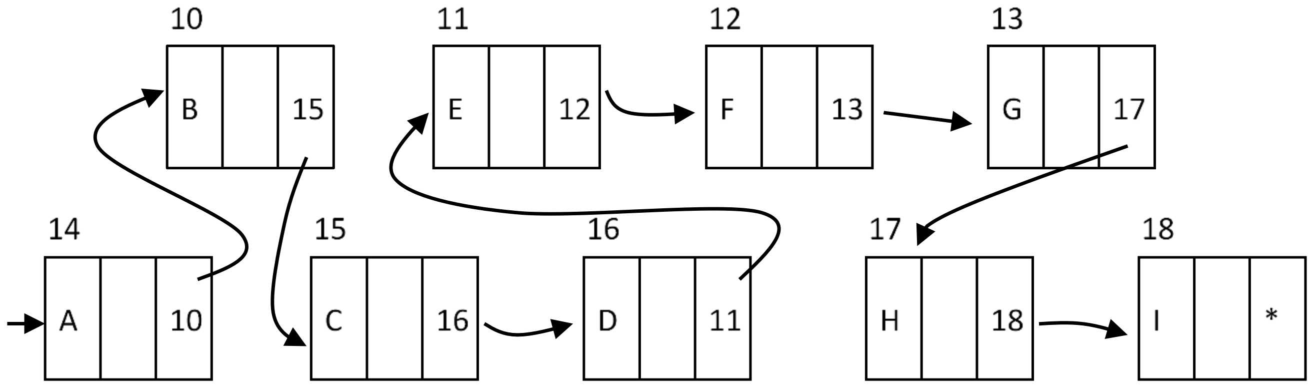
- One-Way Linked List (contd.)
  - to avoid that all records must be retrieved (even if only part of list needs to be processed), a directory can be used
  - a directory is a file that defines relationships between records in another file



List Data Organization

- One-Way Linked List (contd.)
  - indexed addressing: records distributed into intervals, with each interval represented by an index entry

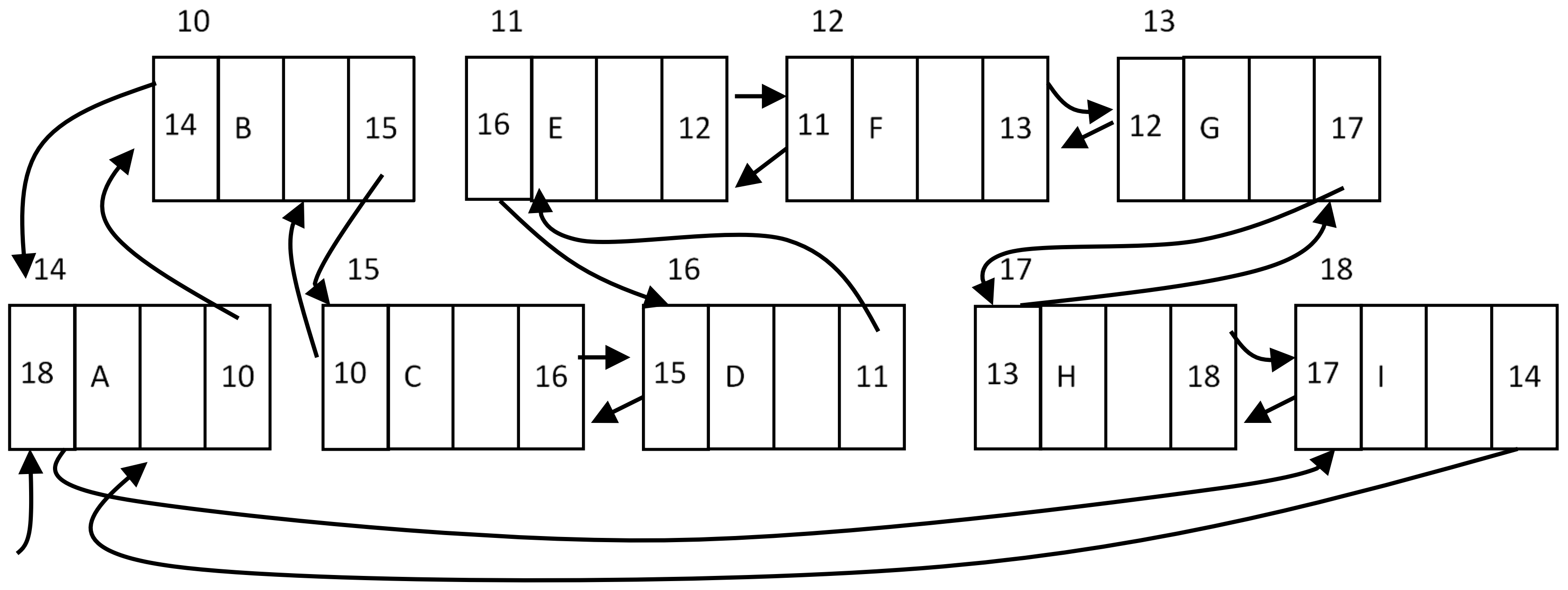
Key	Address
C	14
F	16
I	13



List Data Organization

- One-Way Linked List (contd.)
  - expected retrieval time similar to sequential file organization except all block accesses are rba
  - limited impact of blocking
  - cannot easily retrieve predecessor of record
  - list cannot be reconstructed in case pointer gets lost or damaged
- Two-Way Linked List
  - each record contains ‘prior’ pointer as well as ‘next’ pointer
  - list can be processed efficiently in both directions

List Data Organization



List Data Organization

- A tree consists of nodes and edges with following properties:
  - one *root* node
  - every node, except for root, has exactly one *parent node*
  - every node has 0, 1 or more *children* or *child nodes*
  - nodes with same parent node are called *siblings*
  - all children, children-of-children, etc. of a node are the node's *descendants*
  - a node without children is a *leaf node*
  - tree structure consisting of non-root node and all its descendants is a *subtree*
  - nodes are distributed in *levels*, representing distance from root
  - tree where all leaf nodes are at the same level is called *balanced,* otherwise *unbalanced*

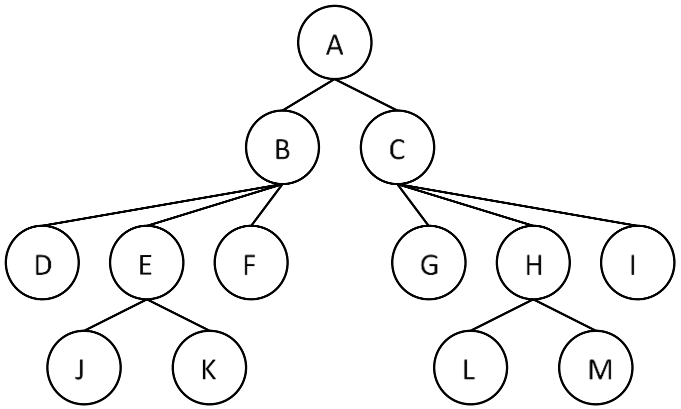
List Data Organization

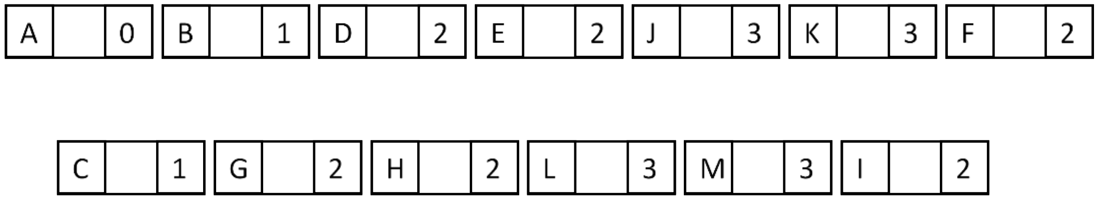
- Tree data structures can be used to provide
  - physical representation of logical hierarchy or tree structure (e.g. hierarchy of employees)
  - purely physical index structure to speed up search and retrieval of records by navigating interconnected nodes of tree (search tree)
    - B-trees and B^+-trees

List Data Organization

- Trees can be implemented by means of physical contiguity
  - nodes stored in ‘top-down-left-right’ sequence: first root, then root’s first child, then child’s first child etc.
  - if node has no more children, its next sibling (from left to right) is stored
  - if node has no more siblings, its parent’s next sibling is stored
  - each nodes’ level needs to be included explicitly

List Data Organization



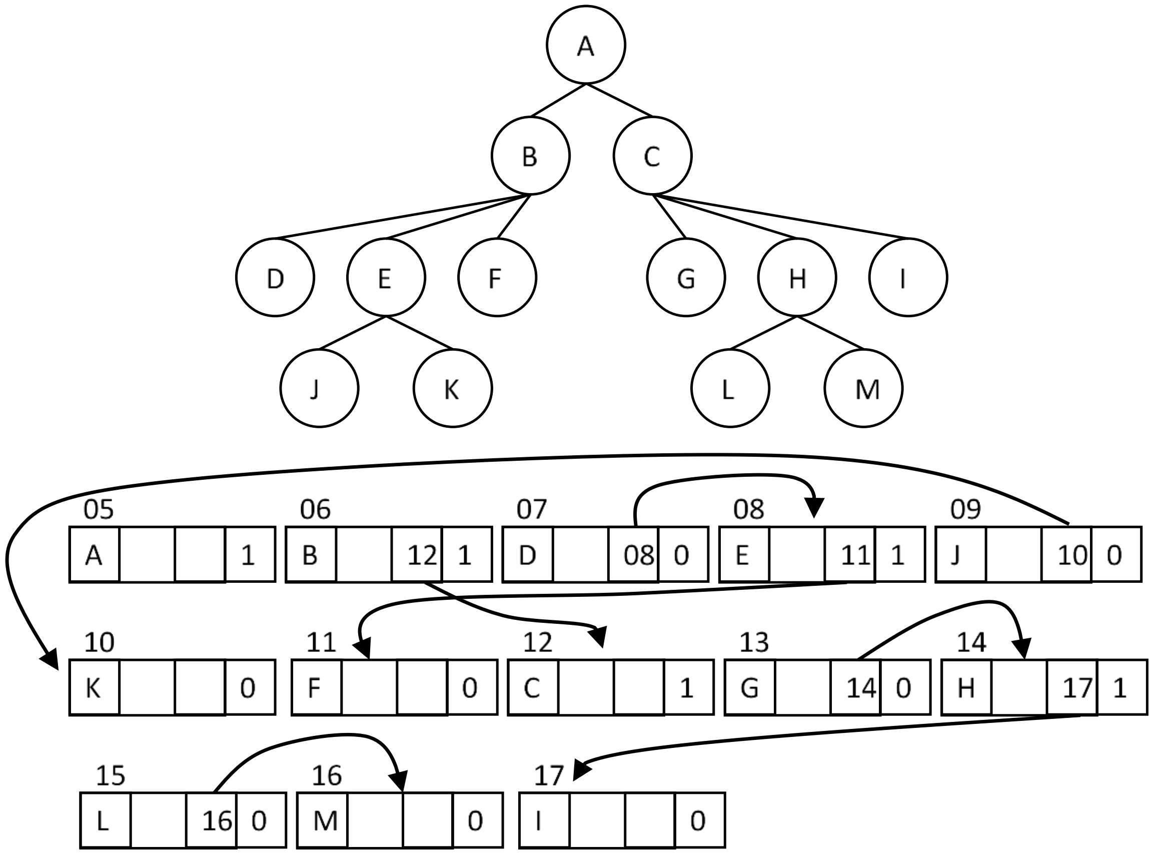


- Letters represent record keys, numbers denote level in tree for physical records
- Can only navigate in sequential way

List Data Organization

- Trees can be implemented by means of Linked Lists
  - physical contiguity complemented with pointers
  - each node has pointer to its next sibling, if it exists
  - both parent-child and sibling-sibling navigation supported, respectively by accessing physically subsequent record and following pointer
  - single bit often added indicating whether node is a leaf node (bit = 0) or not (bit = 1)

List Data Organization



- Many variations possible!

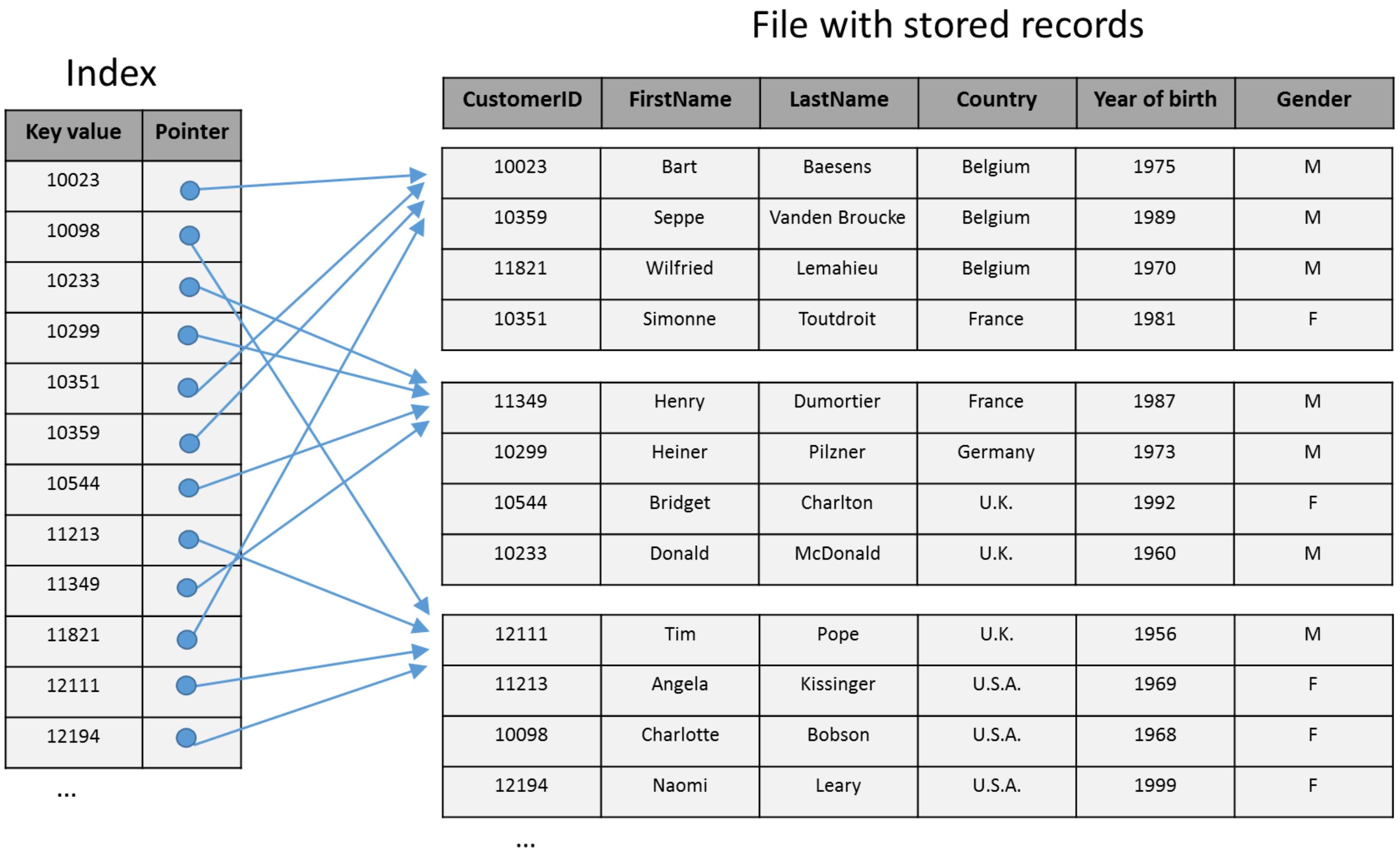
Secondary Indexes and Inverted files

- Characteristics of Secondary Indexes
- Inverted Files
- Multicolumn Indexes
- Other Index Types

Characteristics of Secondary Indexes

- Secondary index is based on attribute type or set of attribute types that is/are not used as ordering criteria of actual data file
- Secondary index's search key can be
  - atomic or composite
  - primary key, other candidate key, or non-key attribute type or combination of attribute types
- Index is again sequential file which can be searched by means of binary search

Characteristics of Secondary Indexes



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	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

...

Unique search key!

Characteristics of Secondary Indexes

- If search key is non-unique
  - dense index, with index entry for each record and multiple entries with same key value
  - add a level of indirection, with each index entry referring to separate block that contains all pointers to records with corresponding search key value (inverted file)

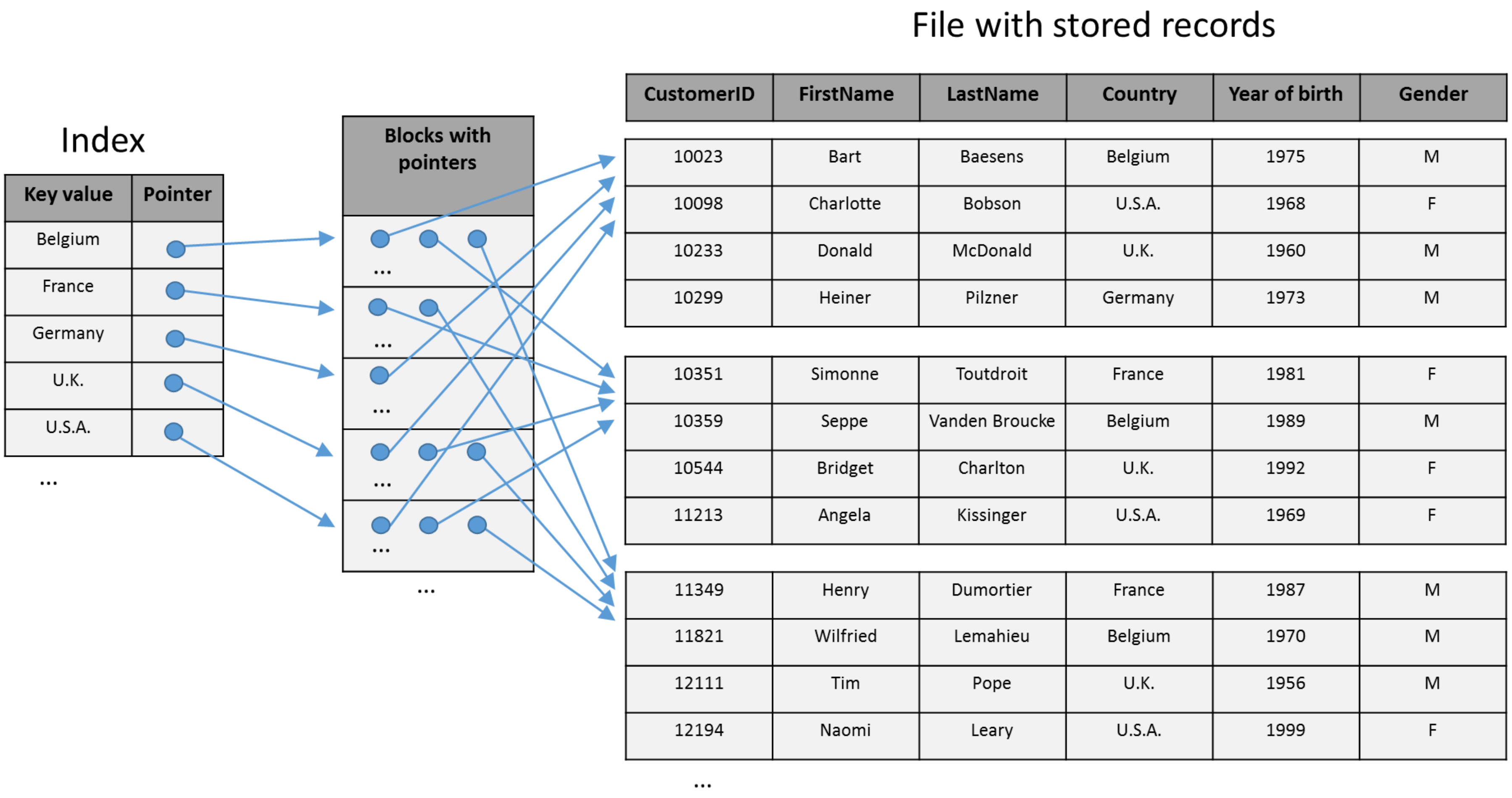
Characteristics of Secondary Indexes

- Sample file with 30000 records and 1500 blocks
- Secondary index defined over unique search key
- Index contains 30000 index entries (one for each search key value)
- Index entry size = 15 bytes, Blocking Factor of index = 136
- NBLKI = $\lceil 30000/136 \rceil$ = 221 blocks
- Expected number of block accesses to retrieve record by means of the secondary index $\text{Log}_2(221) + 1 \approx 9$ rba
- Without secondary index, it would take a full file scan, hence on average 1500/2 = 750 sba

Inverted files

- Inverted file defines index over non-unique, non-ordering search key of data set
- Index entries: <key value, block address>
- Block address refers to block containing record pointers or block pointers to all records with that particular key value
- Requires additional rba to block with pointers to records
- Queries that involve multiple attribute types can be executed efficiently by taking the intersection of blocks with pointers

Inverted files



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

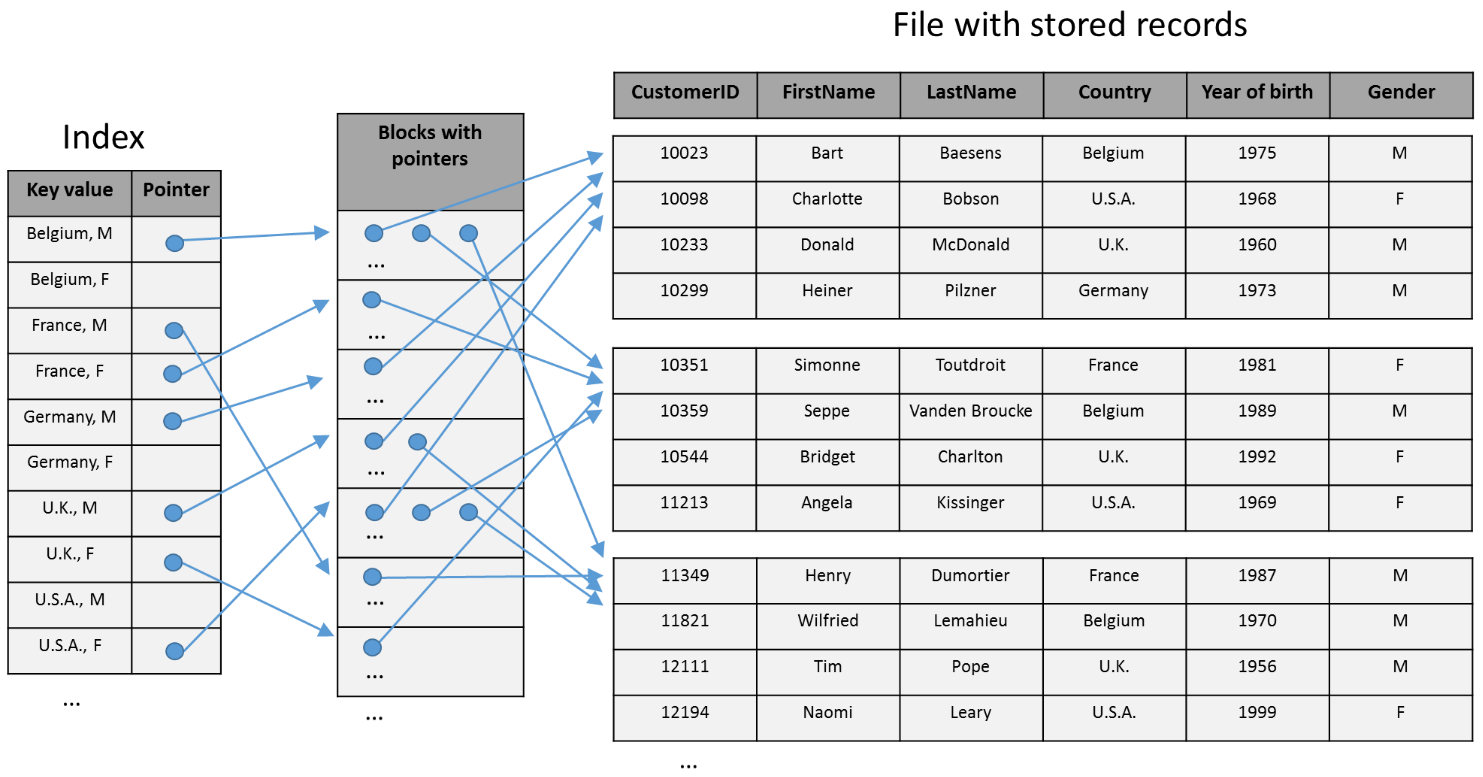
...

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

...

Multicolumn Indexes

- Multicolumn index is index over composite search key
  - can be implemented using inverted files



Index

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

...

File with stored records

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

...

- Efficient to retrieve all records with the desired (Country, Gender) values or all people living in a certain country, regardless of their gender!
- Not efficient to retrieve all males regardless of country!

Other Indexes

- Hash indexes provide secondary file organization method that combines hashing with indexed retrieval
  - index entries: <key value, pointer>
  - index is organized as hash file
  - applying hash function to search key, yields index block where corresponding index entry can be found

Other Indexes

- Bitmap index
  - for attribute types with only limited set of values
  - contain a row ID and a series of bits—one bit for each possible value of indexed attribute type
  - bit position that corresponds to the actual value for the row at hand is set to ‘1’
  - row ID’s can be mapped to record pointers

Other Indexes

RowID	Belgium	U.S.A.	U.K.	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

Other Indexes

- Join index
  - multicolumn index that combines attribute types from 2 or more tables in such a way that it contains the precalculated result of a join between these tables

B-Trees and B$^+$-Trees

- Multilevel Indexes Revisited
- Binary Search Trees
- B-trees
- B$^+$-Trees

Multilevel Indexes Revisited

- Multilevel indexes useful for speeding up data access if lowest level index becomes too large
- Index can be considered as a sequential file and building an index-to-the-index improves access
- Higher-level index is, again, a sequential file to which index can be built and so on
- Lowest level index entries may contain pointers to disk blocks or records
- Higher-level index contains as many entries as there are blocks in the immediately lower level index
- Index entry consists of search key value and reference to corresponding block in lower level index
- Index levels can be added until highest-level index fits within single disk block
- First-level index, second-level index, third-level index etc.

Multilevel Indexes Revisited

- With binary search on single index, search interval, consisting of disk blocks, is reduced by 2 with every iteration
- Approximately $\log_2$(NBLKI) rba to search index consisting of NBLKI blocks
  - one additional rba needed to actual data file
- With multilevel index, search interval is reduced by BFI with every index level (BFI = blocking factor of index)
  - BFI denotes how many index entries fit within single disk block
  - also called fan-out of index

Multilevel Indexes Revisited

- Searching data file according to multilevel index requires $\lceil \log_{BFI}(NBLKI)+2 \rceil$ rba (NBLKI = number of blocks in first-level index):
  - need to add index levels until highest-level index fits within single disk block
  - number of required blocks for index level i: $NBLKI_i = \lceil NBLKI_{i-1}/BFI \rceil$ for i = 2,3,…
  - by applying the previous formula (i-1) times, $NBLKI_i = \lceil NBLKI/(BFI^{i-1}) \rceil$ for i = 2,3,… with NBLKI number of blocks in lowest level index
  - for highest-level index, consisting of only one block, it holds that $1 = \lceil NBLKI/(BFI^{h-1}) \rceil$, with h denoting highest index level
  - therefore $h-1 = \lceil \log_{BFI}(NBLKI) \rceil$ and hence $h = \lceil \log_{BFI}(NBLKI)+1 \rceil$
  - number of block accesses to retrieve record by means of multilevel index then corresponds to a rba for each index level, plus a rba to the data file, which thus equals to $\lceil \log_{BFI}(NBLKI)+2 \rceil$
- BFI is typically >> 2, so using multilevel index is more efficient than binary search on single level index

Multilevel Indexes Revisited

- 30000 records sample file
- Retain lowest level index from secondary index example
- Index entries are 15 bytes and BFI = 136
- Number of blocks in first-level index (NBLKI) = 221
- Second-level index then contains 221 entries and consumes $\lceil 221/136 \rceil = 2$ disk blocks
- If third index level is introduced, it contains 2 index entries and fits within single disk block
- Searching record by means of multilevel index requires 4 rba; 3 to respective index levels and 1 to actual data file
  - can also be calculated as: $\lceil \log_{136}(221)+2 \rceil = 4$

Multilevel Indexes Revisited

- Multilevel index can be considered as search tree, with each index level representing level in tree, each index block representing a node and each access to the index resulting in navigation towards a subtree in the tree
- Multilevel indexes may speed up data retrieval, but large multilevel indexes require a lot of maintenance in case of updates

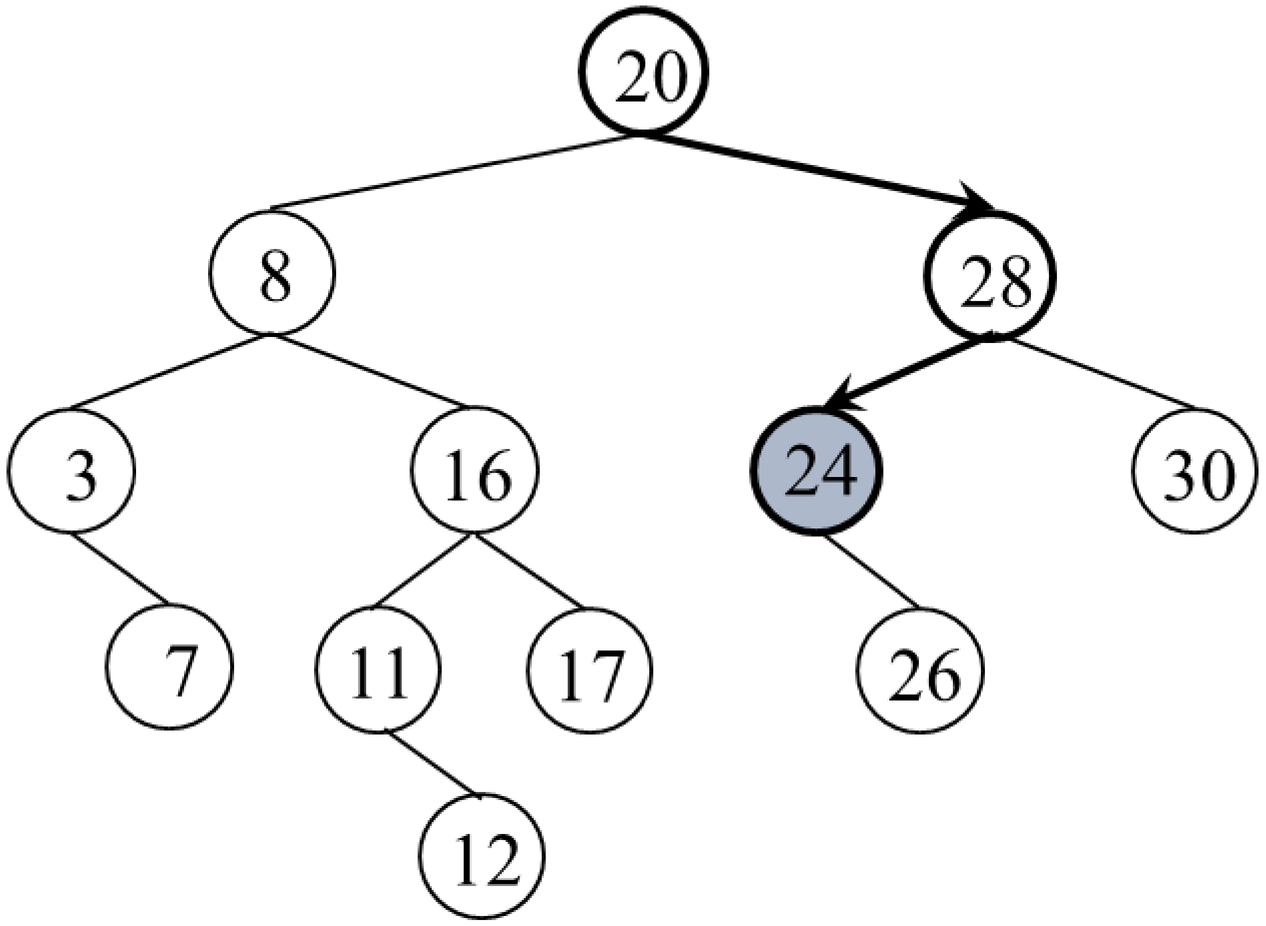
Binary Search Trees

- Binary search tree is a physical tree structure, where each node has at most 2 children
- Each tree node contains a search key value and maximum 2 pointers to children
- Both children are root nodes of subtrees, with one subtree only containing key values that are lower than the key value in the original node, and the other subtree only containing key values that are higher

Binary Search Trees

- Search efficiency improved by ‘skipping’ half of the search key values with every step (~ binary search)
- Suppose search key K is used with values K_i
  - to find node with search key value K_{μ}, key value K_i in root node is compared to K_{μ}
  - if $K_i = K_{\mu}$, search key is found
  - if $K_i > K_{\mu}$, pointer to root of the ‘left’ subtree is followed
  - if $K_i < K_{\mu}$, pointer to root of the ‘right’ subtree is followed
- Apply recursively

Binary Search Trees



B-trees

- A B-tree is a tree-structured index
  - variation of search tree
  - each node corresponds to a disk block and nodes are kept between half full and full to cater for a certain dynamism
- Every node contains a set of search key values, a set of tree pointers that refer to child nodes and a set of data pointers that refer to data records, or blocks with data records, that correspond to search key values
- Data records stored separately and no part of B-tree

B-trees

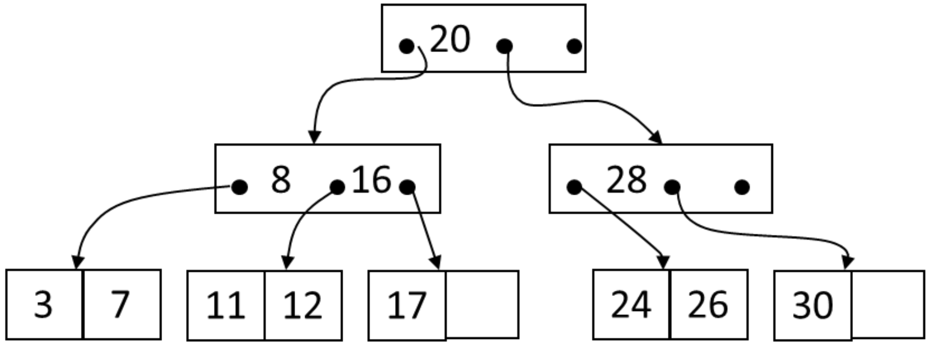
- A B-tree of order k holds the following properties:
  - non-leaf node has format: <P_0, <K_1, Pd_1>, P_1, <K_2, Pd_2>, … <K_q, Pd_q>, P_q>, with $q \leq 2k$. P_i is tree pointer: points to another node in the tree. This node is the root of the subtree that P_i refers to. Every Pd_i is a data pointer: it points to record with key value K_i, or to disk block that contains it.
  - B-tree is a *balanced* tree. Every path from root of to any leaf has same length (*height* of B-tree). Leaf nodes have same structure as non-leaf nodes, except that all their tree pointers P_i are null.
  - within a node: $K_1 < K_2 < … < K_q$
  - for every key value X in subtree referred to by P_i:
    - $K_i < X < K_{i+1}$ for $0 < i < q$
    - $X < K_{i+1}$ for $i = 0$
    - $K_i < X$ for $i = q$

B-trees

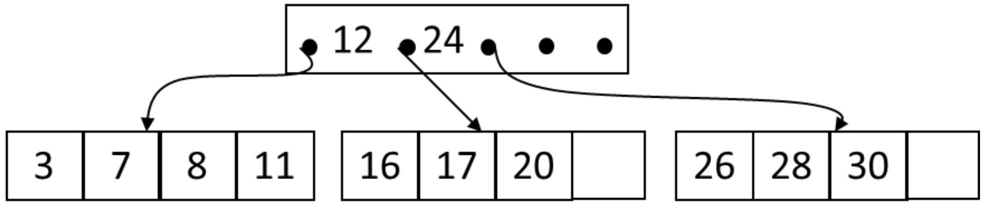
- A B-tree of order k holds the following properties (contd.):
  - B-tree’s root node has a number of key values, and equal number of data pointers between 1 and 2k. Number of tree pointers and child nodes varies between 2 and 2k+1.
  - all internal nodes have number of key values and data pointers between k and 2k. Number of tree pointers and child nodes varies between k+1 and 2k+1.
  - every leaf has a number of key values and data pointers between k and 2k and no tree pointers
- If indexed search key is non-unique, a level of indirection is introduced (e.g., inverted file approach)
  - data pointers Pd_i then point to block containing pointers to all records satisfying search key value K_i

B-trees

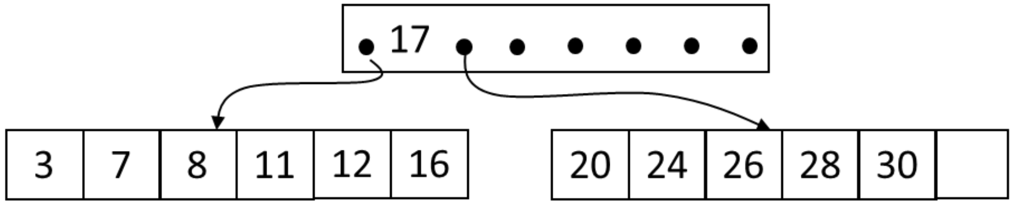
Order 1 (height = 3):



Order 2 (height = 2):



Order 3 (height = 2):



B-trees

- B-tree is searched recursively starting from the root
  - if desired key value X is found in a node (say $K_i = X$), then corresponding data record(s) can be accessed by following Pd_i
  - if desired value is not found in node, the subtree pointer P_i to be followed is the one corresponding to the smallest value of i for which $X < K_{i+1}$. If $X >$ all K_i then the tree pointer P_{i+1} is followed
- Note: fan-out and search efficiency is much higher than with binary search!

B-trees

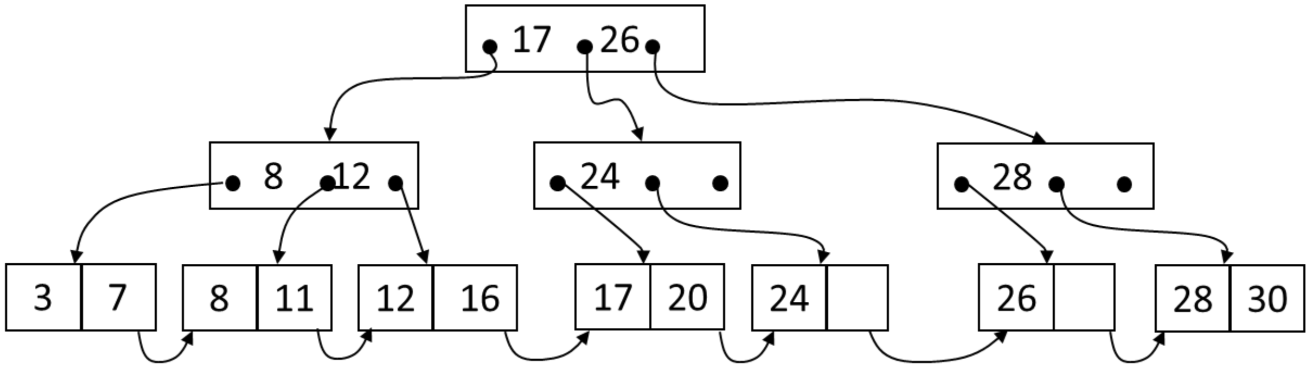
- Capacity of node equals the size of a disk bock
- All nodes, except for the root, are filled for at least 50%
  - impact on additions and removals
- Complex to make exact predictions about required number of block accesses when searching B-tree
- B-trees can also be used as primary file organization technique
  - instead of data pointers, nodes contain actual data fields of records that correspond to search key values

B$^+$-trees

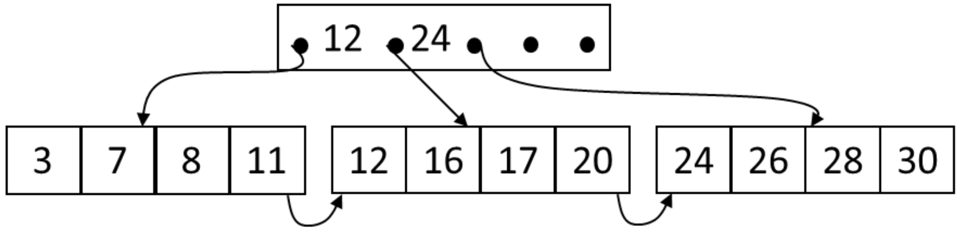
- In a B$^+$-tree
  - only leaf nodes contain data pointers
  - all key values that exist in non-leaf nodes are repeated in leaf nodes, such that every key value occurs in a leaf node, along with corresponding data pointer
  - higher-level nodes only contain subset of key values present in leaf nodes
  - every leaf node of a B$^+$-tree also has one tree pointer, pointing to its next sibling

B⁺-trees

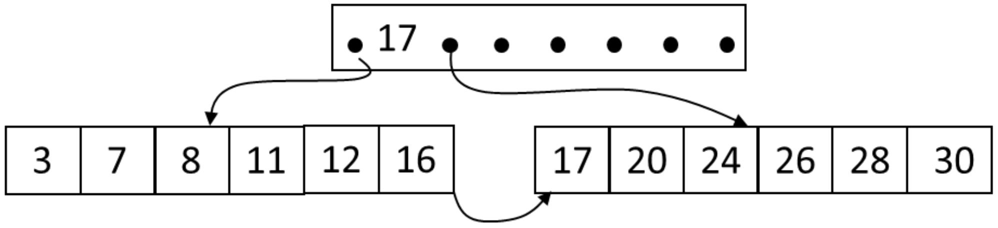
Order 1 (height = 3):



Order 2 (height = 2):



Order 3 (height = 2):



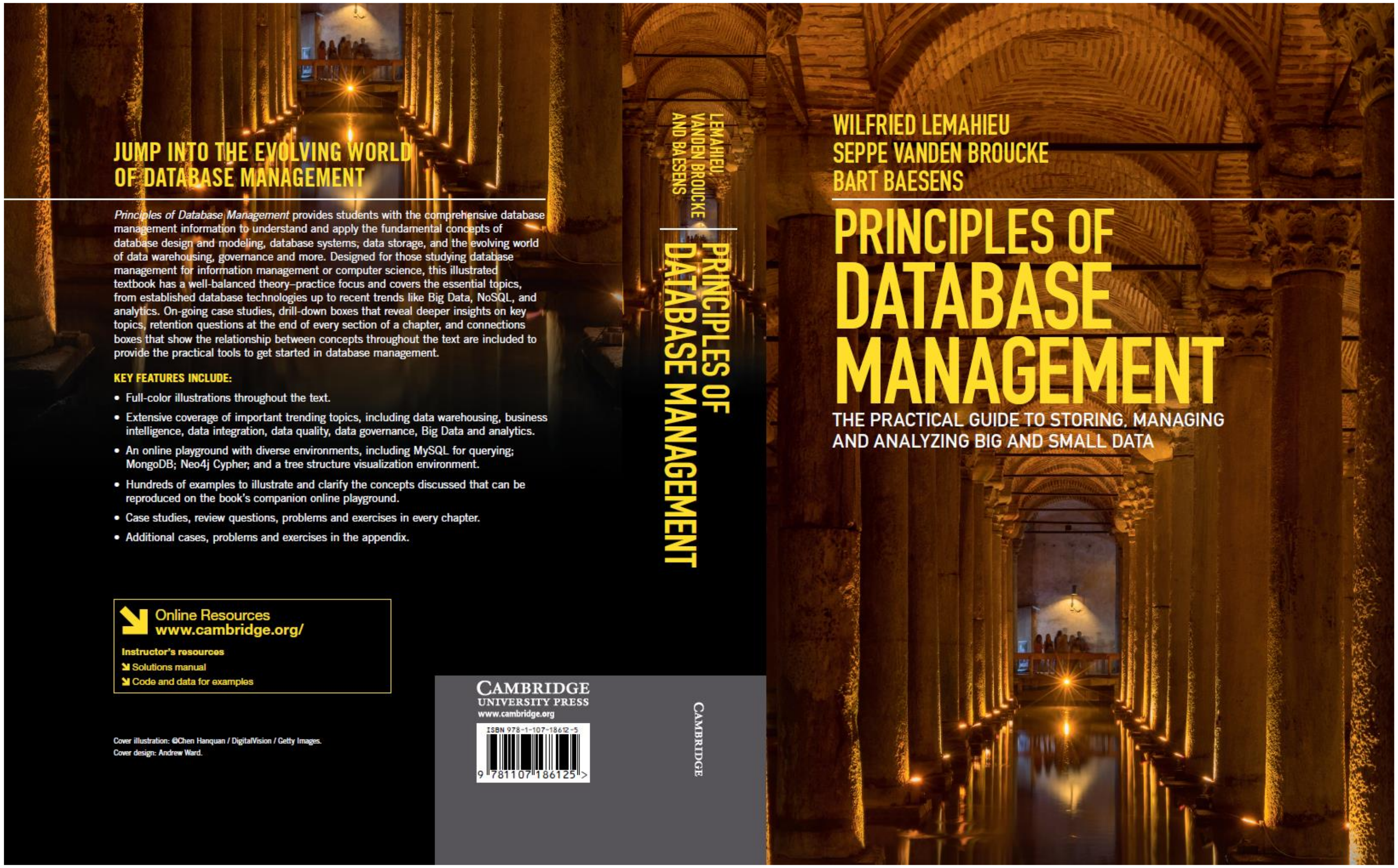
B$^+$-trees

- Searching and updating B$^+$-tree is similar as B-tree
- B$^+$-trees are often more efficient, because non-leaf nodes do not contain data pointers
  - height of B$^+$-tree is often smaller, resulting in less block accesses to search
- Variations on fill factor which is 50% for standard B-trees and B$^+$-trees
  - E.g., a B-tree with fill factor of 2/3 is called a B*-tree

Conclusions

- Storage Hardware and Physical Database Design
- Record Organization
- File Organization

More information?



JUMP INTO THE EVOLVING WORLD OF DATABASE MANAGEMENT

Principles of Database Management provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

KEY FEATURES INCLUDE:

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
www.cambridge.org/

Instructor's resources
Solutions manual
Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

ISBN 978-1-107-18612-5

LEMAHIEU
VANDEN BROUCKE
AND BAESENS
PRINCIPLES OF DATABASE MANAGEMENT
CAMBRIDGE

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

PRINCIPLES OF DATABASE MANAGEMENT

THE PRACTICAL GUIDE TO STORING, MANAGING AND ANALYZING BIG AND SMALL DATA

www.pdbmbook.com