More Indexes and Database Applications

Databases

Ira Assent

ira@cs.au.dk

Data-Intensive Systems group, Department of Computer Science, Aarhus University, DK

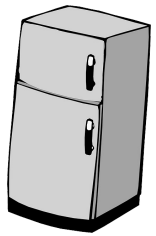
Intended learning outcomes

- Be able to
  - Describe the core principles in B-tree family indexes
  - Use a database from a program in Java or Python

ira@cs.au.dk

Recap: Triggers and Indexes

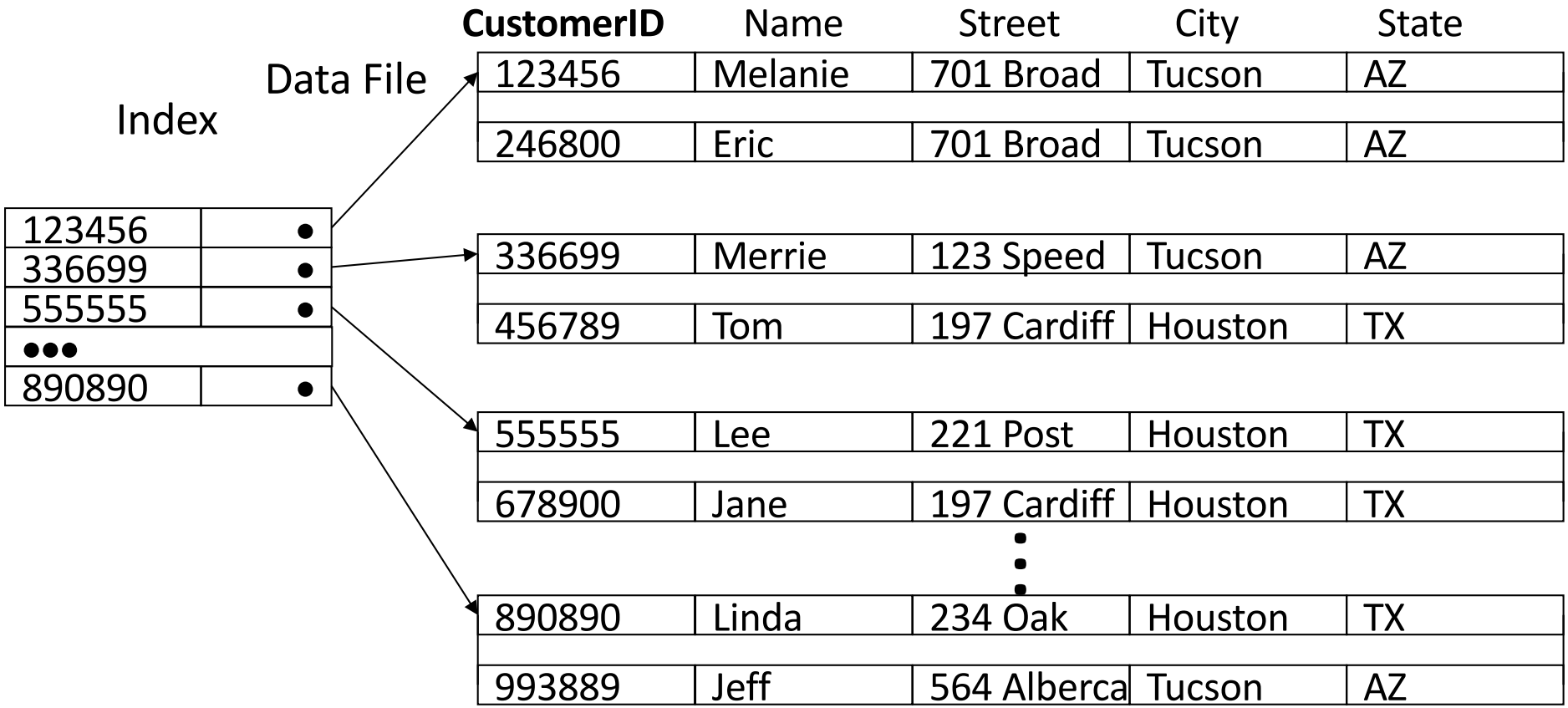
- Triggers enable general reactions to data changes
  - `CREATE TRIGGER PreserveDependencies`
  - event-condition-action rules
    - event: `AFTER, BEFORE;  INSERT, DELETE, UPDATE`
    - condition: any SQL Boolean expression
    - action: any sequence of SQL modifications
  - Use also in combination with views, named queries
- Indexing structures: `CREATE INDEX Run ON Student(name);`
  - For efficient data access on disk
    - "virtual sorting" of Student wrt to search key: attribute name
    - Lookup search key in index file (often in main memory), then load data from disk
- Contiguous block I/O faster than random read (avoids disk seek)
  - One disk access = 1,000,000 RAM accesses! (cache even faster)
- Justifies "count only I/Os" model of complexity
  - Means: "measure" runtime as how many blocks you read from disk



ira@cs.au.dk

Primary Index

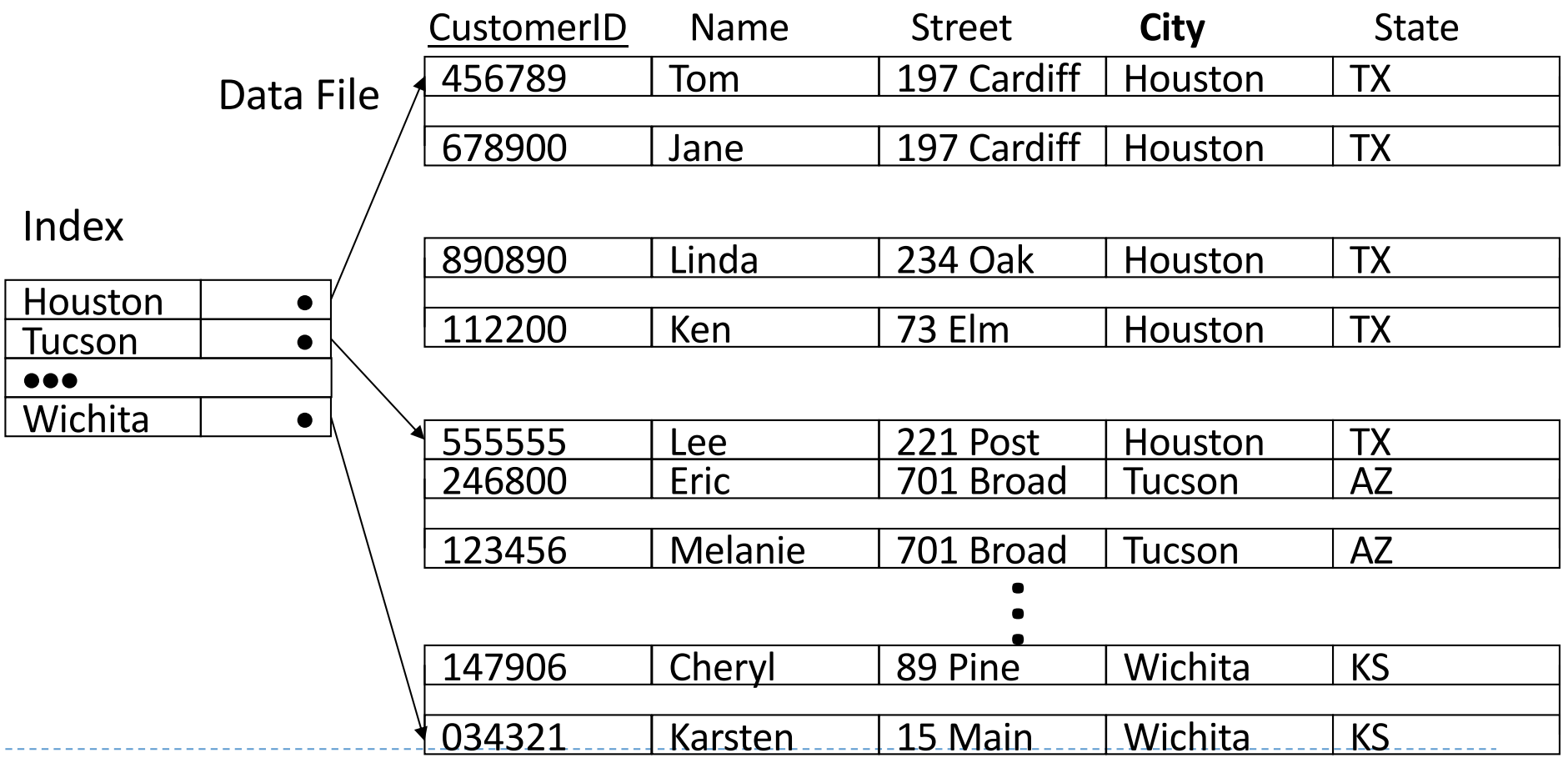
- **Primary Index**: defined on a data file ordered on the primary key
  - **Dense index** has one entry for each search key value
  - **Sparse index** - fewer index entries than search key values



ira@cs.au.dk

Clustering (Dense) Index

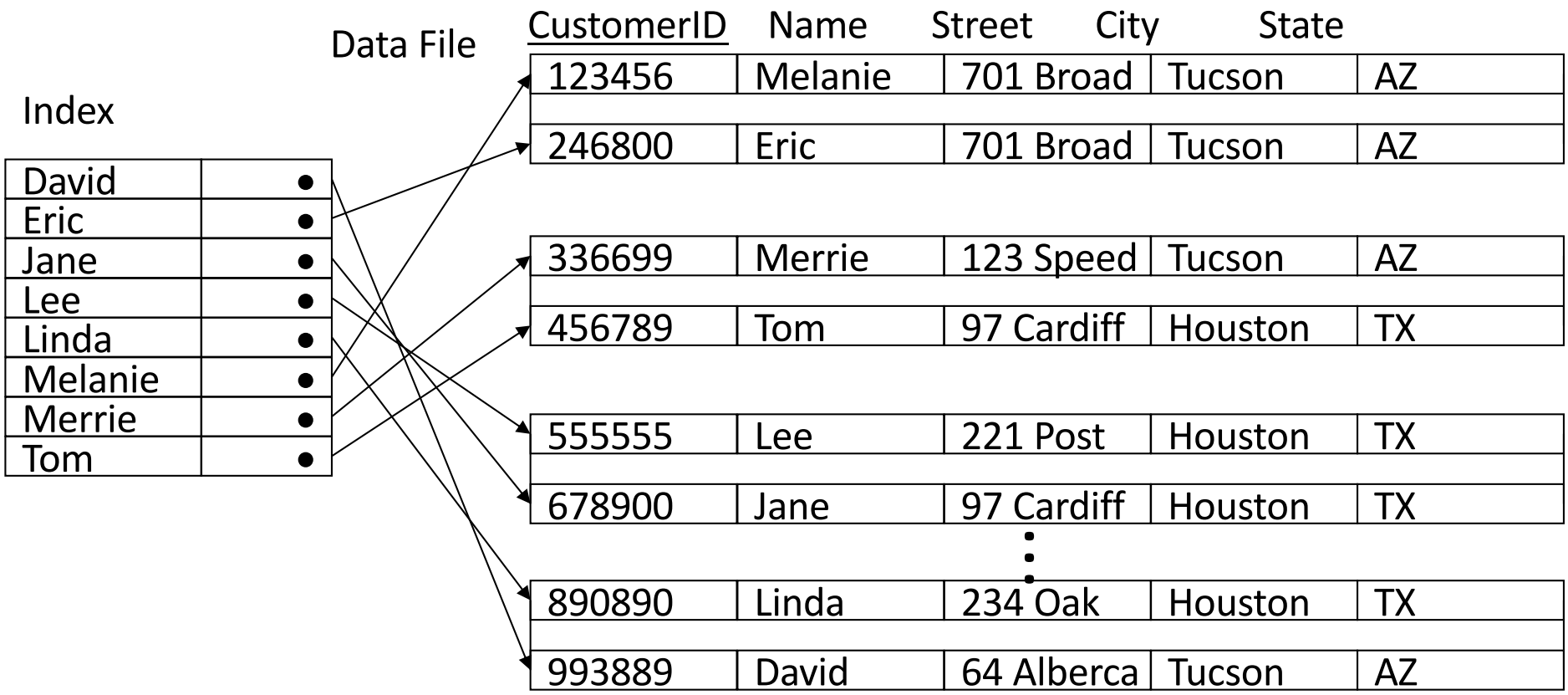
- **Clustering Index**: defined on a data file ordered on a non-key field
  - One index entry for each distinct value of the field, points to first data block of records for search key



ira@cs.au.dk

Secondary (Dense) Index

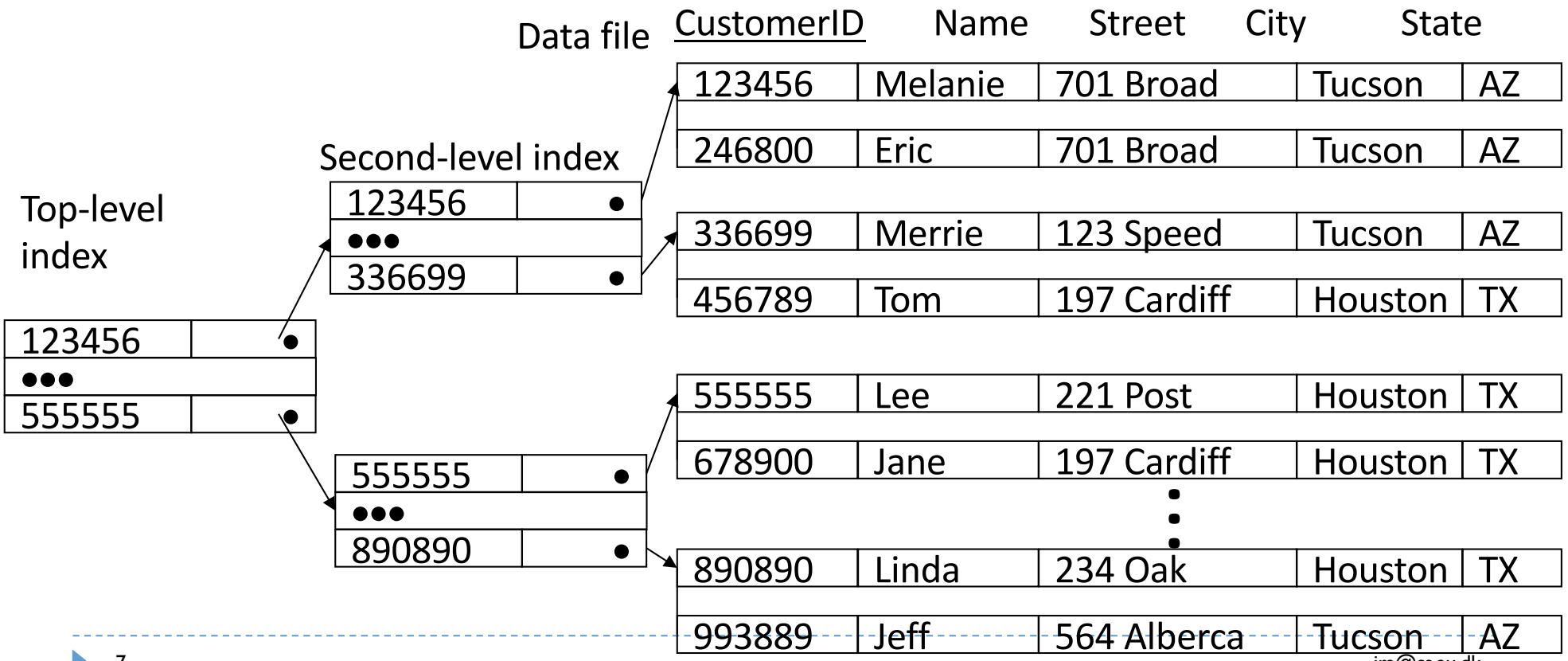
- **Secondary Index**: defined on a data file not ordered on the index's search key



ira@cs.au.dk

Multi-level Index Example

- **Multi-level index**: index on index, until all entries of the top level fit in one disk block
  - Every level of the index is an ordered file
  - Pin top-level index in main memory (RAM)



ira@cs.au.dk

Multilevel index

What is true for multilevel indexes (top level is the one in main memory)?

A. All levels of the index must be sparse
B. All levels of the index except for the bottom-most-level index must be sparse
C. All levels of the index must be dense
D. All levels of the index must be dense except for the bottom-most-level index

ira@cs.au.dk

Using indexes

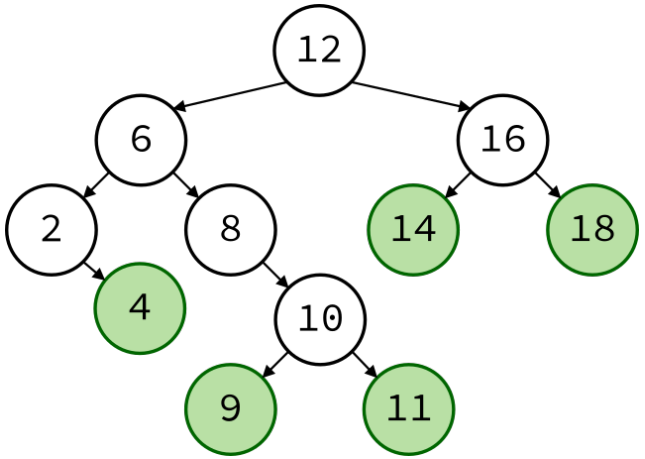
```
SELECT *
  FROM R
  WHERE x=42 AND y>87;
```

- We have one index for `x` and another for `y`
  - Use index to find row pointers for `x=42`
  - Use index to find row pointers for `y>87`
  - Compute the intersection of the two pointer set
- Similarly, `OR` corresponds to disjunction, so the union of the pointer set

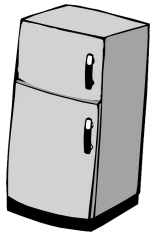
ira@cs.au.dk

B-Trees

- A variation of search trees
  - Supports
    - Insert a row
    - Delete a row
    - Search for a row given the index attribute(s)

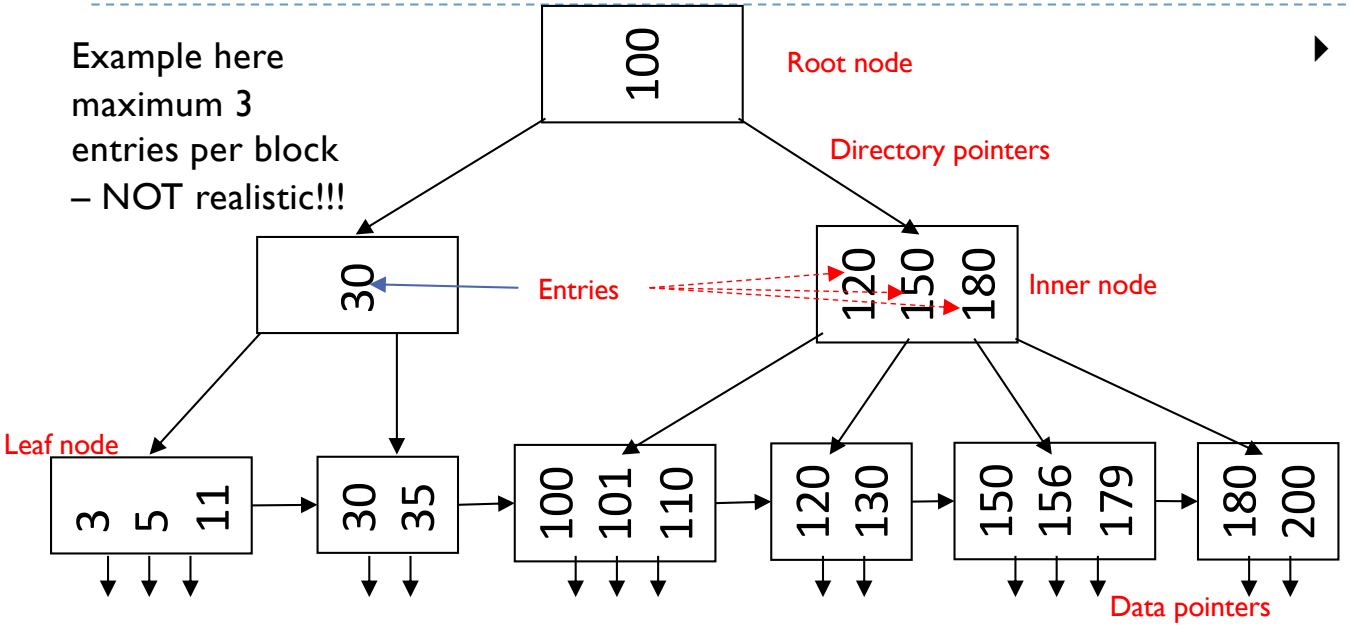


- "Perfect" for disk storage
  - **High fanout**
    - Each node has many children
    - Not binary as in this search tree example
  - **Block based**
    - Each node is the size of one block
  - Very robust to data changes, data volumes, etc.
  - Used by all relational DBMSes



ira@cs.au.dk

B+-Tree

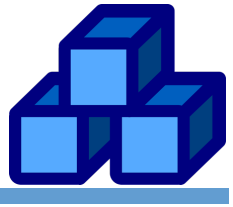


- Values in node odered on search key
  - E.g. search key is `price` for Table `Product(id, name, price)`
  - Left pointer to subtree with smaller values than current
  - Right pointer to subtree with larger or equal values than current

- In practice, use a variant of B-trees known as B+-trees
  - All non-leaf nodes (inner nodes) only contain directory entries
    - i.e., guide the search to lower levels, but do not store any data
    - i.e., all search keys present at the leaf level, and some search keys are "repeated" higher up in the tree
  - Data pointers only at the leaf level → higher "fanout", i.e., more search keys in inner nodes → lower trees → less I/O
- Each row is pointed to by a leaf node

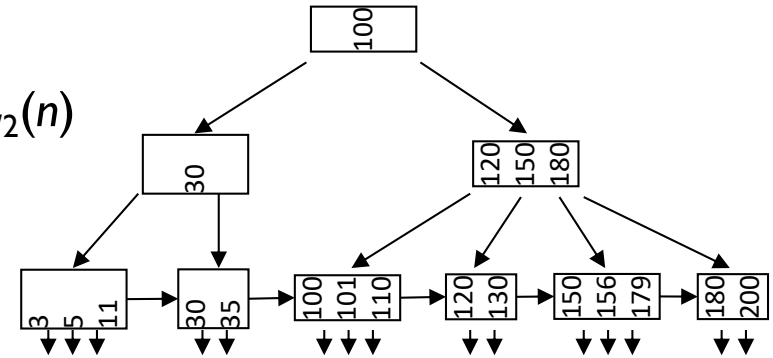
ira@cs.au.dk

B+-Tree Invariants



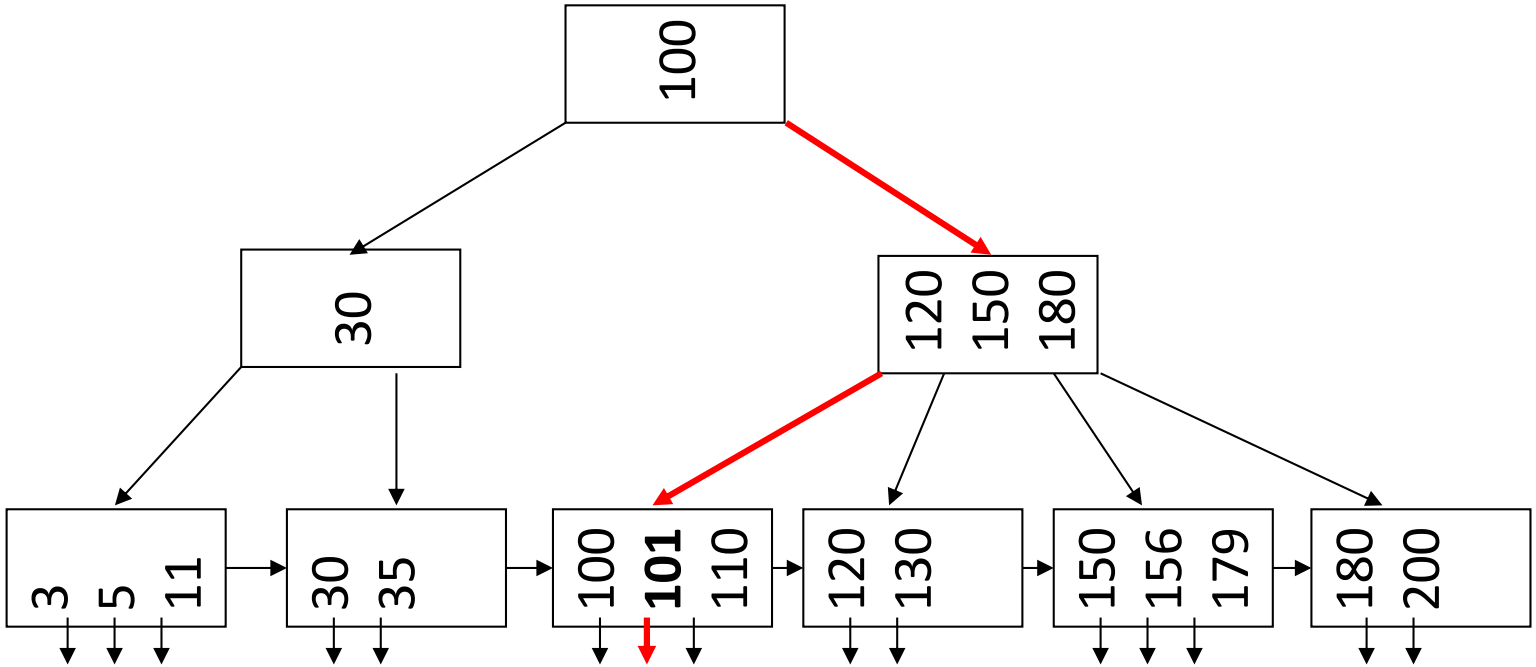
- **Invariants**: properties that will hold for any B+-tree

1. Each node (block) holds at most p-1 keys, compute from OS block size and size of search key
   - Called a B+-tree of order p
     - Also possible to have B+-trees where inner nodes and leaf nodes have different order
   - Typically, p is several hundreds
2. Each node must hold at least $\lceil p/2 \rceil$ pointers
   - One pointer to left of each search key in inner node, plus one to the right of last key
   - One pointer to data of each search key in leaf plus one to next sibling node
   - Except for root node which has at least 2 pointers, unless it is the only node
3. All leaves must be at the same level
   - Thus, tree remains balanced:
     - Its height with *n* rows is at most $1+\log_{p/2}(n)$
     - In practice, the height is 3 or 4
       - 1-2 top levels often in RAM



ira@cs.au.dk

B+-Tree Point Query



- Search path for key 101
  - Want to find all information on products that cost 101

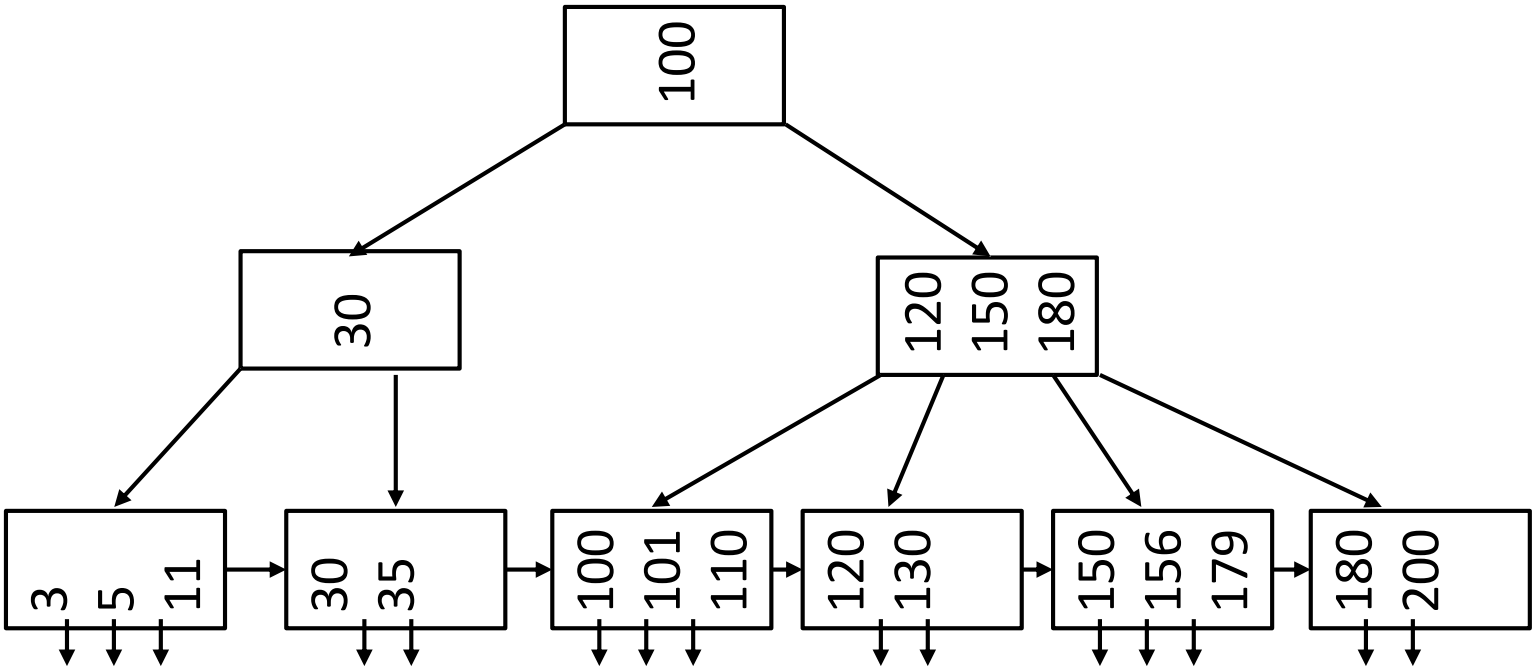
```
SELECT * FROM Product WHERE id = '101';
```

- Time proportional to the height of the tree

ira@cs.au.dk

I/O

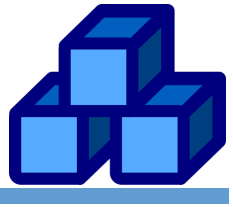
Assuming that the root is held in main memory, and that each node is one page, how many I/Os for a range query from 101-166?



A. 2
B. 3
C. 4
D. 5
E. 6
F. 9

ira@cs.au.dk

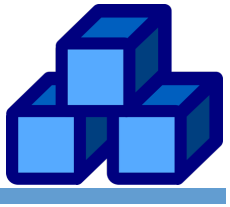
Maintaining B+-trees

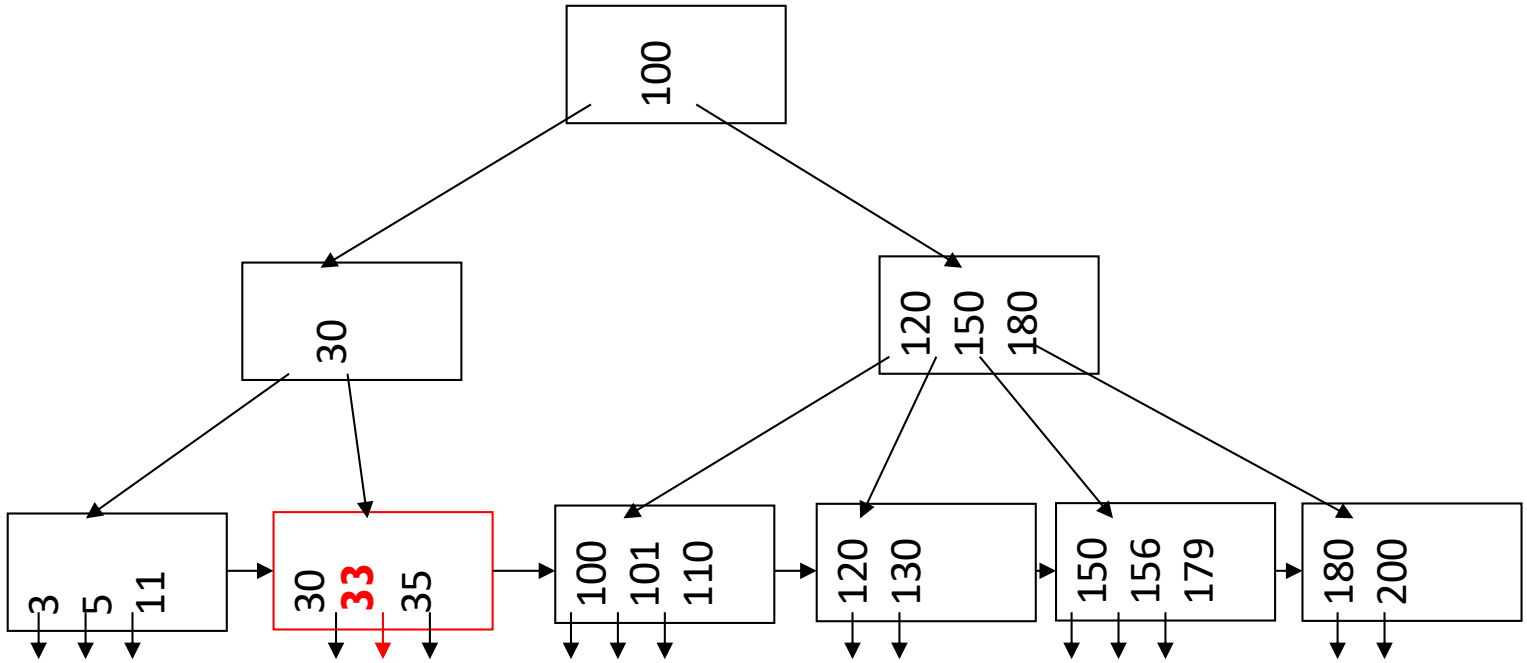


- When the database is modified, the index needs to be modified as well
  - Insert, update, delete
- Index needs to always be ordered
  - some extra maintenance needed when no free space in index file where search key should be
- B+-trees can typically be modified locally, i.e., only few blocks need changes
  - Means low I/O cost → efficient maintenance

- We start with insert examples
- Briefly see deletes
- Updates handled as delete plus insert

B+-Tree Insertion (1/3)

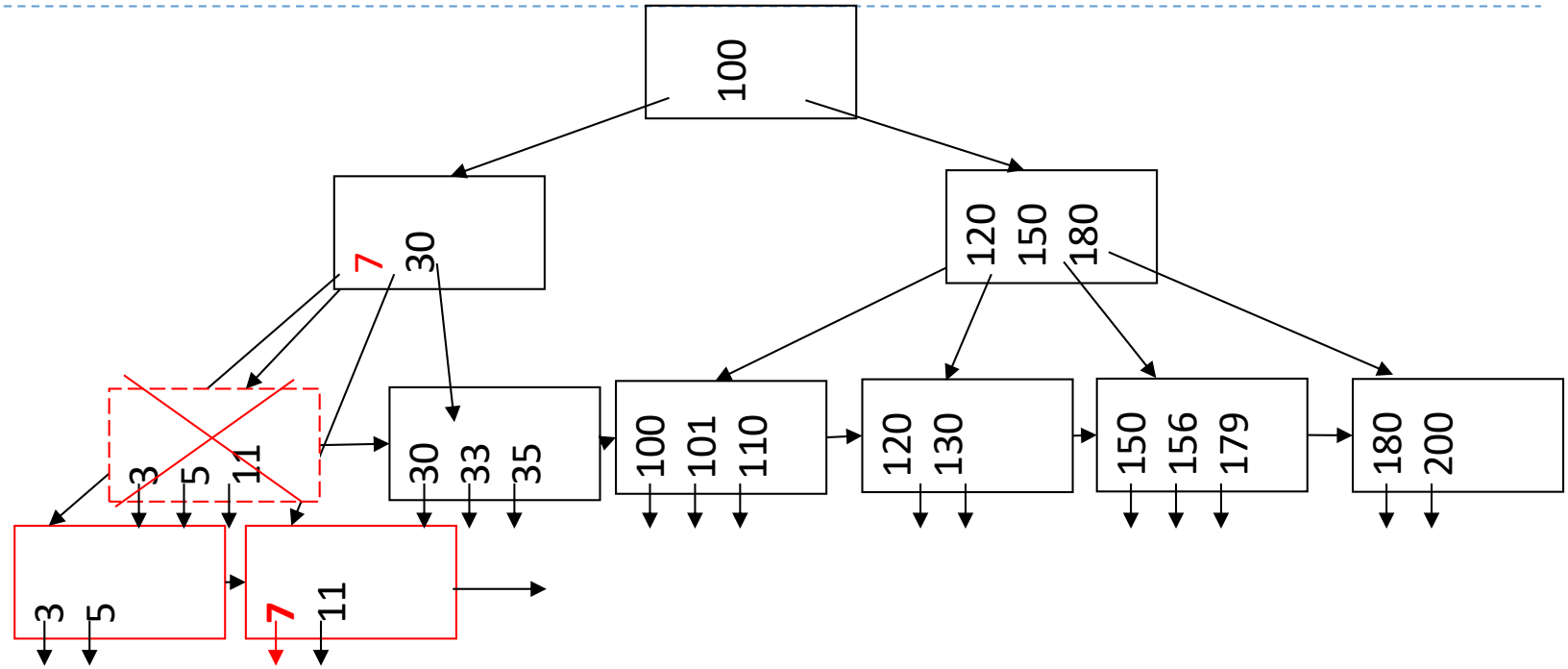




- Inserting 33 (simple case)
  - As in query, find where it should be and simply add if there is space

ira@cs.au.dk

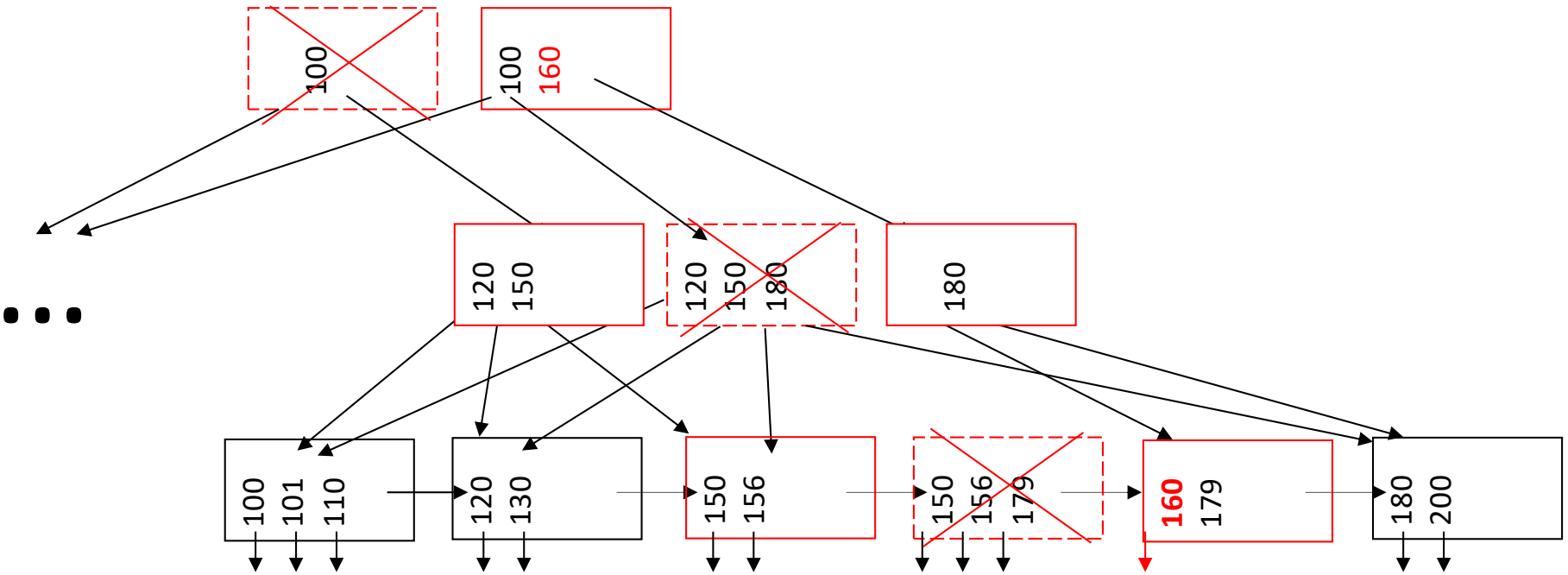
B+-Tree Insertion (2/3)



- Inserting 7 (split leaf)
  - Find, where it should be
  - Leaf node with (3,5,11) is full
    - Need to split the node into two new nodes which replace the old one
  - Need a corresponding entry also higher up in the index so the two new nodes are correctly found (7 entered before 30)

ira@cs.au.dk

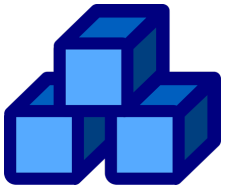
B+-Tree Insertion (3/3)



- Inserting 160
  - 150, 159, 179 full
- Split and send to parent
  - 120, 150, 180 also full
- Split and send to parent root

ira@cs.au.dk

B+-Tree Deletion

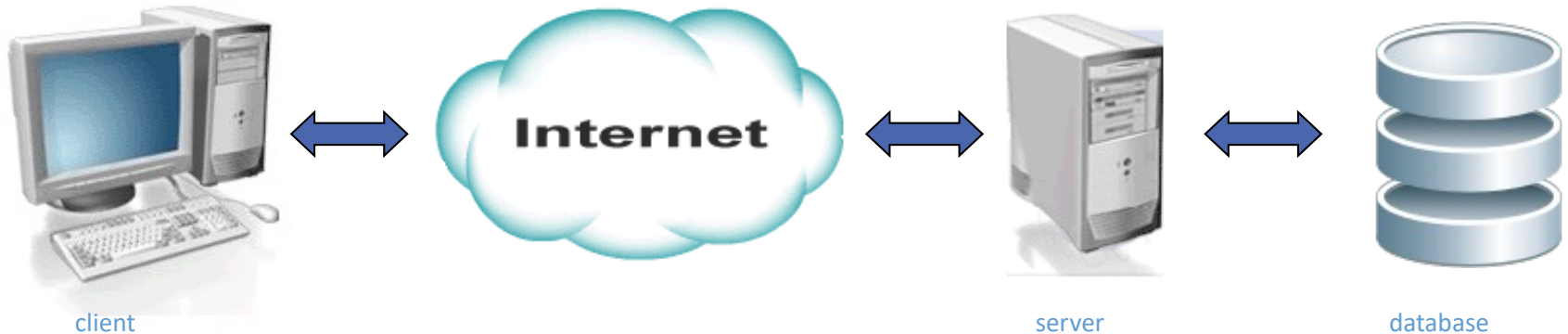


- Balanced deletion is also possible
  - There are similar case-based algorithms
- Generally, deleted rows are left as *tombstones*
  - The overhead of deletion is too large
- Most tables tend to grow with time
  - The tombstones quickly get reused
- Otherwise, periodically rebuild the index
  - Or perform online *reorg* of the index

ira@cs.au.dk

Embedded SQL

- SQL is rarely written as ad-hoc queries using the generic SQL interface
- The typical scenario:



- SQL is *embedded* in the server application code

ira@cs.au.dk

Static vs. Dynamic SQL

- Static SQL
  - syntactic extension of host language
  - predefined and stored in the database
  - typical use: monthly accounting statements
  - checked in advance, efficient
- Dynamic SQL
  - API in host language
  - dynamically interpreted by the database
  - typical use: web applications
  - highly flexible

- We consider Java and Python here to match commonly used programming languages for most study programs in this lecture
  - Connect from program to a database
  - Query database or execute statements on the connection
  - Close connection

ira@cs.au.dk

Calling all Data Science Students

- **SQLite** is a very lightweight DBMS storing a database in a single file, without a separate database server
  - As the name suggests, supports SQL queries
  - Does not have all the features of MySQL, but sufficient for smaller applications

https://www.sqlite.org/docs.html

- SQLite is included in both iOS and Android mobil phones

- Let's say we want to create a table like the one below in SQLite using the programming language Python

country			
name	pop	area	capital
'Denmark'	5748769	42931	'Copenhagen'
'Germany'	82800000	357168	'Berlin'
'USA'	325719178	9833520	'Washington, D.C.'

ira@cs.au.dk

Python and SQLite


```
import sqlite3
connection = sqlite3.connect('example.sqlite')  # creates file if
necessary
c = connection.cursor()
c.executescript('''DROP TABLE IF EXISTS country;
                   DROP TABLE IF EXISTS city''')
countries = [('Denmark', 5748769, 42931, 'Copenhagen'),
             ('Germany', 82800000, 357168, 'Berlin'),
             ('USA', 325719178, 9833520, 'Washington, D.C.')]
cities = [('Copenhagen', 'Denmark', 775033, 800),
          ('Aarhus', 'Denmark', 273077, 750),
          ('Berlin', 'Germany', 3711930, 1237),
          ('Washington, D.C.', 'USA', 693972, 1790)]
c.execute('CREATE TABLE country (name, pop, area, capital)')
c.execute('CREATE TABLE city (name, country, pop, founded)')
c.executemany('INSERT INTO country VALUES (?,?,?,?)', countries)
c.executemany('INSERT INTO city VALUES (?,?,?,?)', cities)
connection.commit()  # save data to database before closing
connection.close()
```

Make a database connection – the program and the database can "talk"

Make a cursor to work on the connection

The cursor allows execution of SQL!

When done, make sure to commit (= save your changes) and to close the connection again

ira@cs.au.dk

SQLite query examples


```
for row in c.execute('SELECT * FROM country'): # returns iterator
    print(row)          # row is by default a Python tuple
```

('Denmark', 5748769, 42931, 'Copenhagen')
('Germany', 82800000, 357168, 'Berlin') …

```
for row in c.execute('''SELECT * FROM city, country WHERE
city.name = country.capital AND city.pop < 700000'''):
    print(row)
```

('Washington, D.C.', 'USA', 693972, 1790, 'USA', 325719178, 9833520, 'Washington, D.C.') …

```
print(*c.execute('''SELECT country.name, COUNT(city.name)
   AS cities, 100 * SUM(city.pop) / country.pop
   FROM city JOIN country ON city.country = country.name
       WHERE city.pop > 500000
       GROUP BY city.country'''))
```

('Germany', 2, 6) ('USA', 2, 0) ('Denmark', 1, 13) …

ira@cs.au.dk

MySQL and Python

- Connecting to MySQL Using Connector/Python
- The `connect()` constructor creates a connection to the MySQL server and returns a MySQLConnection object
- example how to connect to the MySQL server:

```
import mysql.connector
cnx = mysql.connector.connect(user='MeMyselfAndI',
password='VerySecret123', host='127.0.0.1',
database='NGO')
cnx.close()
```

- Allows software program and database to communicate through the connection

https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html

ira@cs.au.dk

Querying MySQL from Python program

- query using a cursor created using the connection's cursor() method
  - data returned is formatted and printed on console

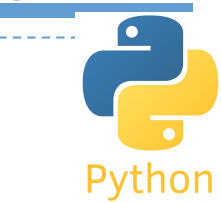

```
import datetime
import mysql.connector
cnx = mysql.connector.connect(user='Me',
database='NGO')
cursor = cnx.cursor()
query = ???
start= datetime.date(2023, 1, 1)
end = datetime.date(2023, 12, 31)
cursor.execute(query, (start, end))
for (name, when) in cursor:
   print("{}, {} was hired on {:%d %b
%Y}".format(name, when))
cursor.close()
cnx.close()
```

???

1. ("SELECT start, end FROM Supporter " "WHERE name IS NOT NULL")
2. ("SELECT name, when FROM Supporter " "WHERE name IS NOT NULL")
3. ("SELECT name, when FROM Supporter " "WHERE when BETWEEN %s AND %s")
4. ("SELECT start, end FROM Supporter " "WHERE name BETWEEN %s AND %s")

ira@cs.au.dk

Querying MySQL from Python program



- query using a cursor created using the connection's cursor() method
  - data returned is formatted and printed on console

```
import datetime
import mysql.connector
cnx = mysql.connector.connect(user='Me',
database='NGO')
cursor = cnx.cursor()
query = ("SELECT name, when FROM
Supporter" "WHERE when BETWEEN %s AND %s")
start= datetime.date(2023, 1, 1)
end = datetime.date(2023, 12, 31)
cursor.execute(query, (start, end))
for (name, when) in cursor:
   print("{}, {} was hired on {:%d %b
%Y}".format(name, when))
cursor.close()
cnx.close()
```

Debbie was hired on 22 Feb 2023

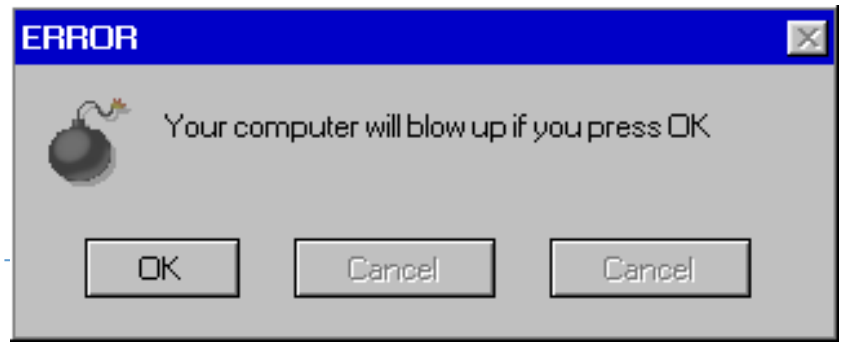
open a connection to MySQL server
store connection object in variable cnx
create a new cursor, a MySQLCursor object, using the connection's cursor() method
SELECT statement in variable query, using unquoted %s-markers for dates
start and end converted from Python types to MySQL type, adds required quotes
replaces the first %s with '2023-01-01', second with '2023-12-31'
Execute query using execute() method
data used to replace the %s-markers in the query is passed as a tuple: (start, end)
MySQL server produces result set
use the cursor object as an iterator
first column in row in variable name, second in when
print result, formatting output using Python's built-in format() function
when converted automatically to Python datetime.date object

https://dev.mysql.com/doc/connector-python/en/connector-python-example-cursor-select.html
ira@cs.au.dk

Connection error handling

- To handle connection errors: use try statement, catch all errors using the `errors.Error` exception

```
import mysql.connector
from mysql.connector import errorcode
try:
   cnx = mysql.connector.connect(user='YouAndYOU',
   database='NGO')
   except mysql.connector.Error as err:
      if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("Incorrect user name or password")
      elif err.errno == errorcode.ER_BAD_DB_ERROR:
         print("Database does not exist")
      else:
         print(err)
   else:
      cnx.close()
```



ira@cs.au.dk

JDBC – Java Database Connectivity

- A common Java framework for SQL databases

1. Import JDBC library of classes: `import java.sql.*`
2. Load driver class `Class.forName("com.mysql.jdbc.Driver");`
   - Also available for other DBMS
3. Creating a connection:
   - `Connection con;`
   - `con = DriverManager.getConnection(url, userid, password);`
4. Create a connection object: `DriverManager.getConnection(url, userid, password);`
5. Create and execute statement

- SQL statements are built as string expressions
- URL structure (for MySQL)
  - `jdbc:mysql://server:port/database`
  - the name of your own machine is localhost
  - the standard port number is 50000
  - the name of the database is, e.g., NGO

ira@cs.au.dk

SQL Statements

- Create a statement object:

```
Statement stmt = con.createStatement();
```

- The statement object is used many times 

  - `stmt.executeQuery("…");`
    - Read data (SELECT queries)
      - `stmt.executeQuery("SELECT * FROM Rooms");`
  - `stmt.executeUpdate("…");`
    - Manipulate data (Modifications of the data)
      - `stmt.executeUpdate("DELETE * FROM Rooms");`

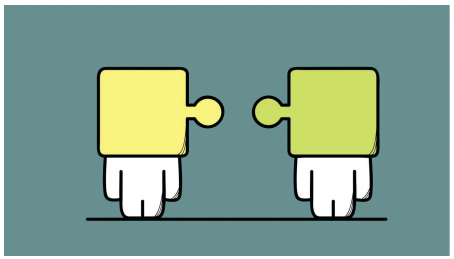
- And is finally closed 

  - `stmt.close();`

ira@cs.au.dk

Impedance Mismatch

- **Impedance mismatch**: relational model vs objects or types in programming languages
  - When exchanging data, needs to be handled exceptionally
  - E.g. Java uses native types
    - `int, char[], String, ...`
    - collection classes
  - Whereas SQL uses tables
    - `CHAR(7), VARCHAR(20), FLOAT, DATE, ...`
    - possibly huge amounts of data
- Not obvious how to translate tables into Java objects
  - Results are instead accessed using cursors



ira@cs.au.dk

Using Result Sets

- A ResultSet object manages a cursor on rows

```
ResultSet rs = stmt.executeQuery("...");
while (rs.next()) {
    ...
    }
rs.close();
```

- A cursor can by default only move forward
  - `rs.next();`
- A Boolean result tells if the move was possible
  - looks like an iterator object
- `ORDER BY` determines the order

rs

room	capacity
Turing-216	4
Ada-333	26
Aud-E	286

ira@cs.au.dk

Reading With Cursors

- Column index or column name
  - `String room = rs.getString(1);`
  - `int capacity = rs.getInt("capacity");`
- Different result types
  - `getString(...)`
  - `getInt(...)`
  - `java.sql.Time time = getTime(...)`
- Check for NULL
  - `wasNull()`

rs

room	capacity
Turing-216	4
Ada-333	26
Aud-E	286

ira@cs.au.dk

Modifications with Cursors

- A result set can then be updated

```
rs.updateString("room","ADA-333");
```

- Updates can be pushed to the database

```
rs.updateRow();
```

- Rows can be deleted both places

```
rs.deleteRow();
```

	room	capacity
	Turing-216	4
rs	Ada-333	26
	Aud-E	286

- A special virtual *insert row* exists

```
rs.moveToInsertRow();
rs.updateString("room”,"Turing-310");
rs.updateInt("capacity",4);
rs.insertRow();
rs.moveToCurrentRow();
```

	room	capacity
	Turing-216	4
rs	Ada-333	26
	Aud-E	286
	Turing-310	4

ira@cs.au.dk

MySQL and Java example

```
import java.sql.*;
public class Test {
   public static void main(String args[]) {
      Connection con;
      try {
         String server = "localhost";
         String port = "50000";
         String url = "jdbc:mysql://"+server+":"+port+"/sample";
         String userid = "Mimmi";
         String password = "YourGuess";
         Class.forName("com.mysql.jdbc.Driver").newInstance();
         con = DriverManager.getConnection(url, userid, password);
         Statement stmt = con.createStatement();
         ResultSet rs = stmt.executeQuery("SELECT * FROM Rooms");
         while (rs.next())
           System.out.println(rs.getString(1)+" "+rs.getString(2));
         stmt.close();
         con.close();
       } catch(Exception e) { e.printStackTrace(); }
   }
}
```

Store-Aud 152
Lille-Aud 70
Nyg-357 1
...

Prepared Statements

- SQL statements may be prepared
  - checked and compiled once
  - executed multiple times


```
PreparedStatement pstmt =
    con.prepareStatement("SELECT * FROM Rooms");
ResultSet rs = pstmt.executeQuery();
```


ira@cs.au.dk

Arguments to Prepared Statements

- Use ? symbols for variables
- Insert values using absolute position

```
PreparedStatement pstmt =
    con.prepareStatement(
        "INSERT INTO Meetings VALUES(?,?,?,'dDB',?)"
    );
pstmt.setInt(1,34716);
pstmt.setDate(2,new java.sql.Date(2014,10,6));
pstmt.setInt(3,14);
pstmt.setString(4,"ira");
pstmt.executeUpdate();
```


ira@cs.au.dk

Summary

- Intended learning outcomes
  - Be able to
    - Describe the core principles in B-tree family indexes
    - Use a database from a program in Java or Python

ira@cs.au.dk

Where to go from here?

- Now we can build real database programs!

- We will look at how to authorize access to a database or parts of it
  - In reality, not everyone will have full access to all data

  - We will look at extremely large databases and how they deviate from the relational approach
    - NoSQL

ira@cs.au.dk

What was this all about?

Guidelines for your own review of today's session

- A B+-Tree is…
  - To query a B+-tree…
  - The B+-Tree is kept balanced by…
- In order to use a database as part of a program we connect…
  - This is done by…
    - For SQlite and Python…
    - For MySQL and Python….
    - For MySQL and Java….
  - Impedance mismatch means that…
    - A cursor is used as follows…

ira@cs.au.dk