Databases in Python

MySQL, SQLite

Accessing persistent storage (Relational databases) from Python code

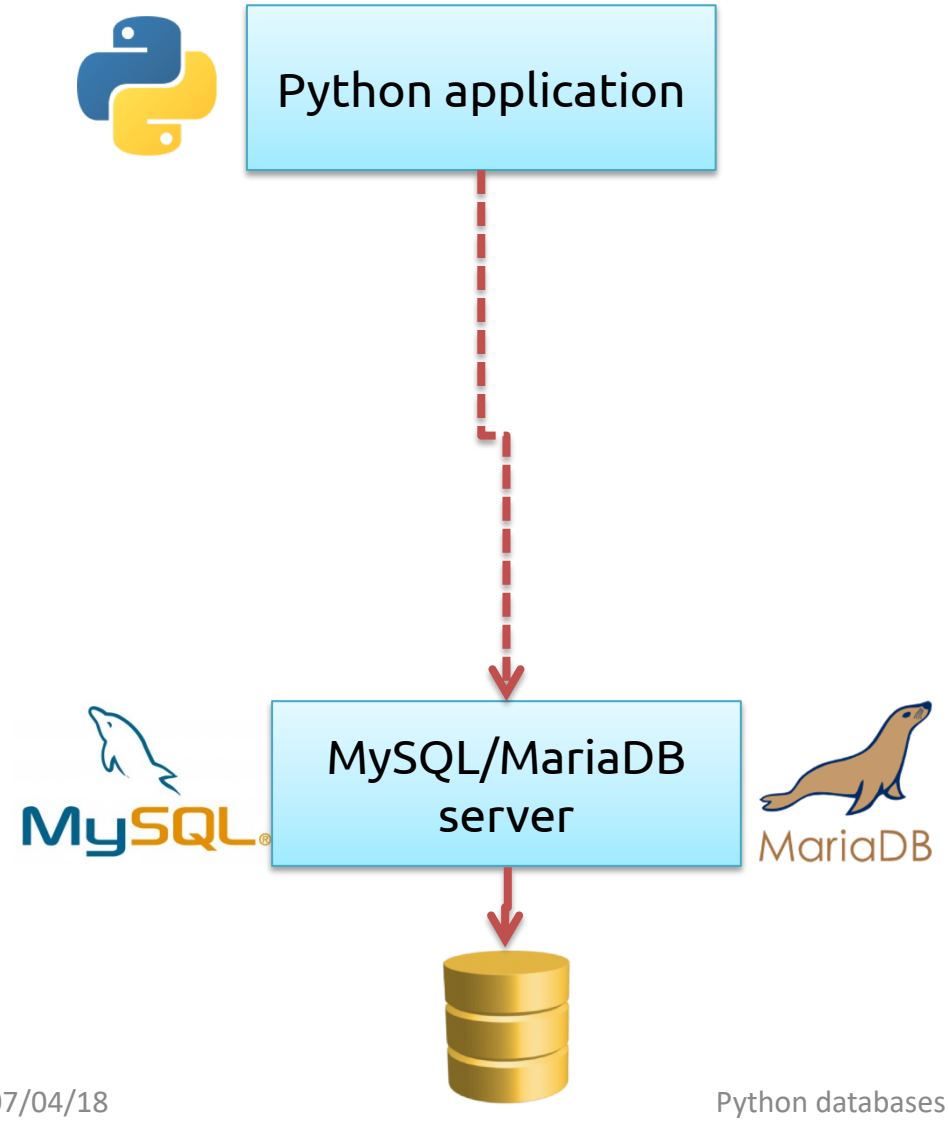
POLITECNICO
DI TORINO

e-Lite

Goal

- Making some data 'persistent'
  - When application restarts
  - When computer restarts
- Manage big amounts of data
  - Not all in-memory
- Exploit the power of SQL
  - Complex data
  - Complex queries

General Architecture



Analyzed databases

MySQL

- Open source database server (from Oracle)
- Full featured
- Runs as a separate process (may be on a different computer)
- Allows concurrent access
- http://dev.mysql.com

Analyzed databases

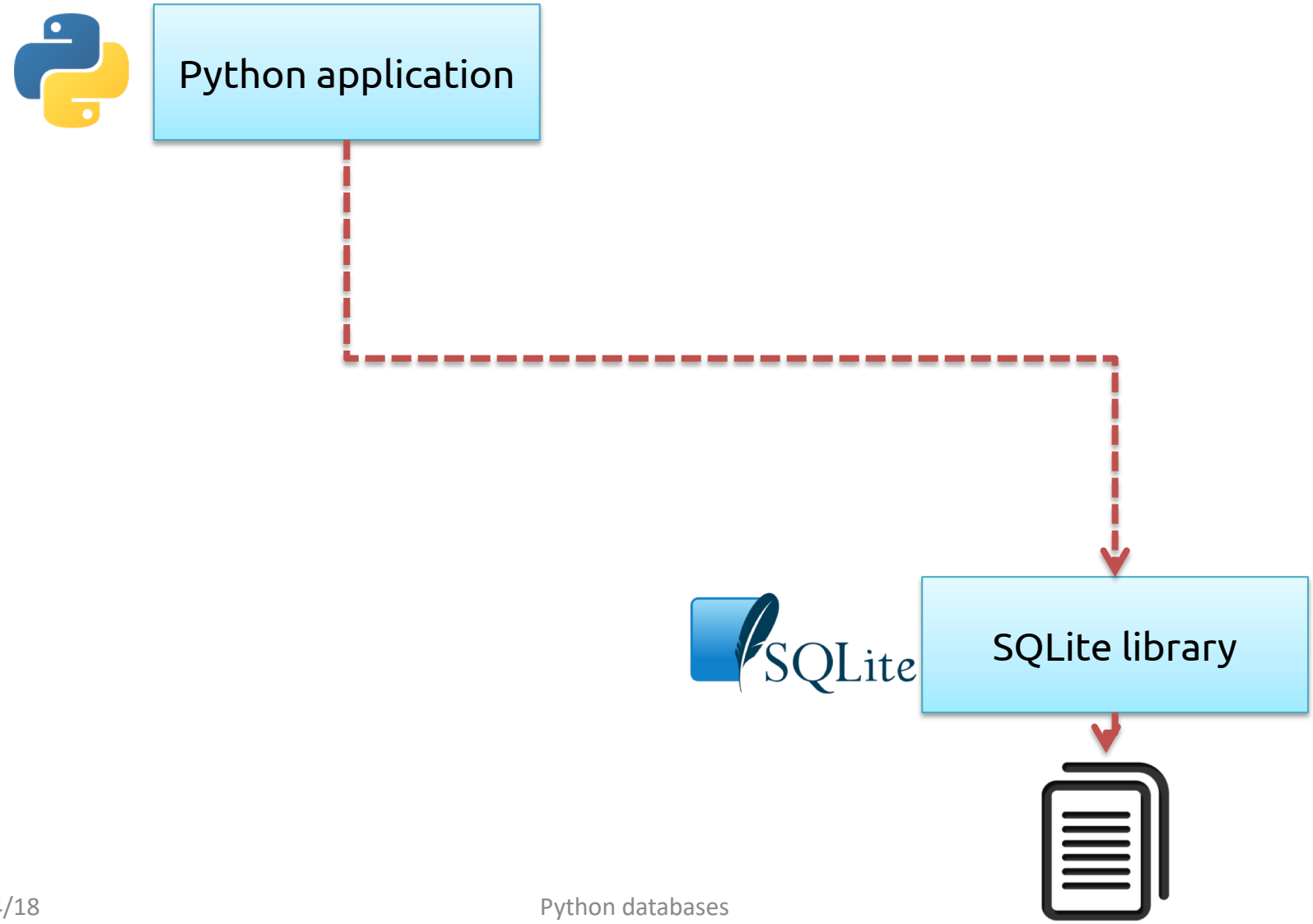
MySQL

- Open source database server (from Oracle)
- Full featured
- Runs as a separate process (may be on a different computer)
- Allows concurrent access
- http://dev.mysql.com

MariaDB

- Open source fork of MySQL server
- Community-driven
- 99% compatible
- In some cases, faster
- On most Linux distributions
- http://mariadb.org/

General Architecture



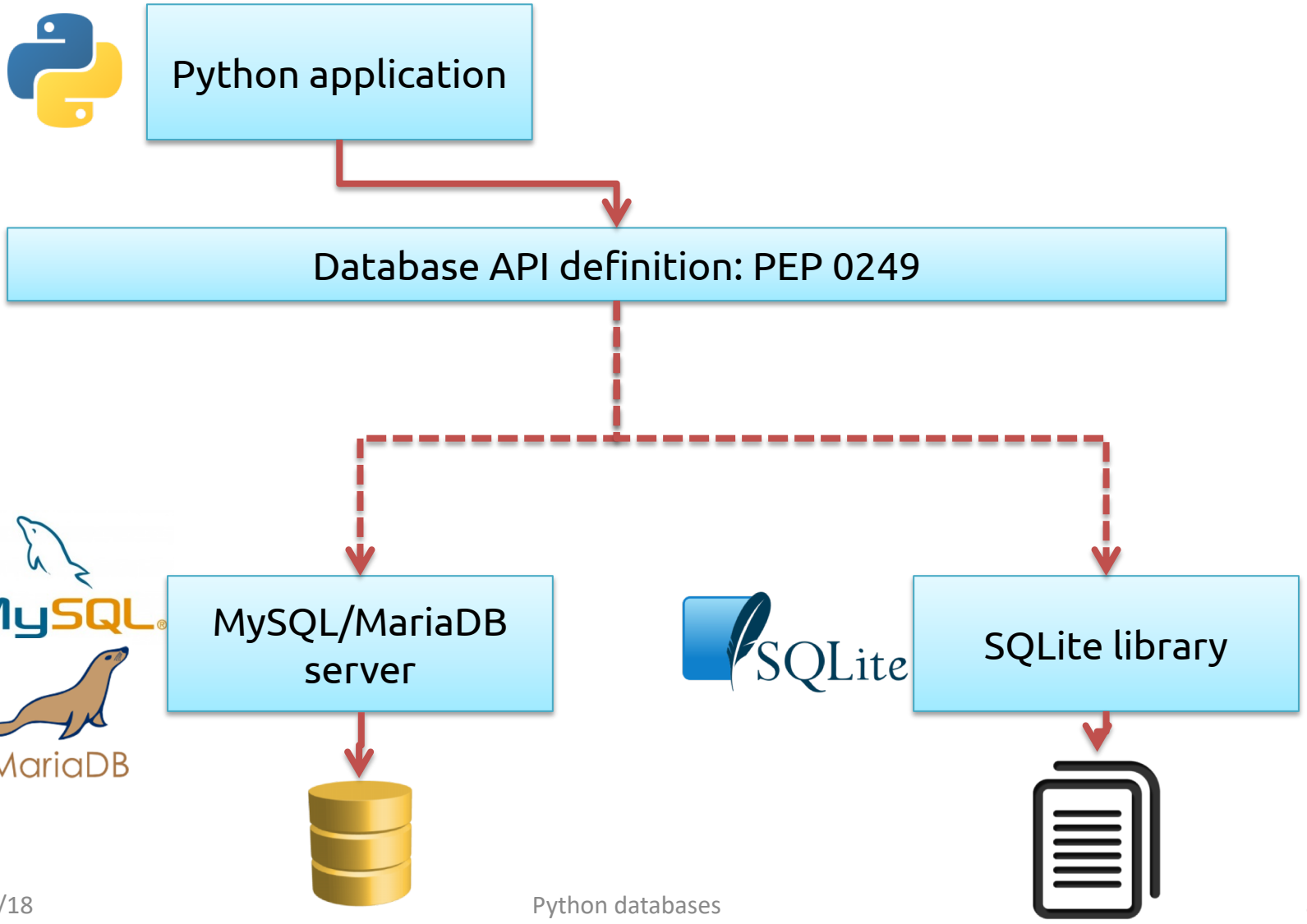
Analyzed databases

MySQL / MariaDB

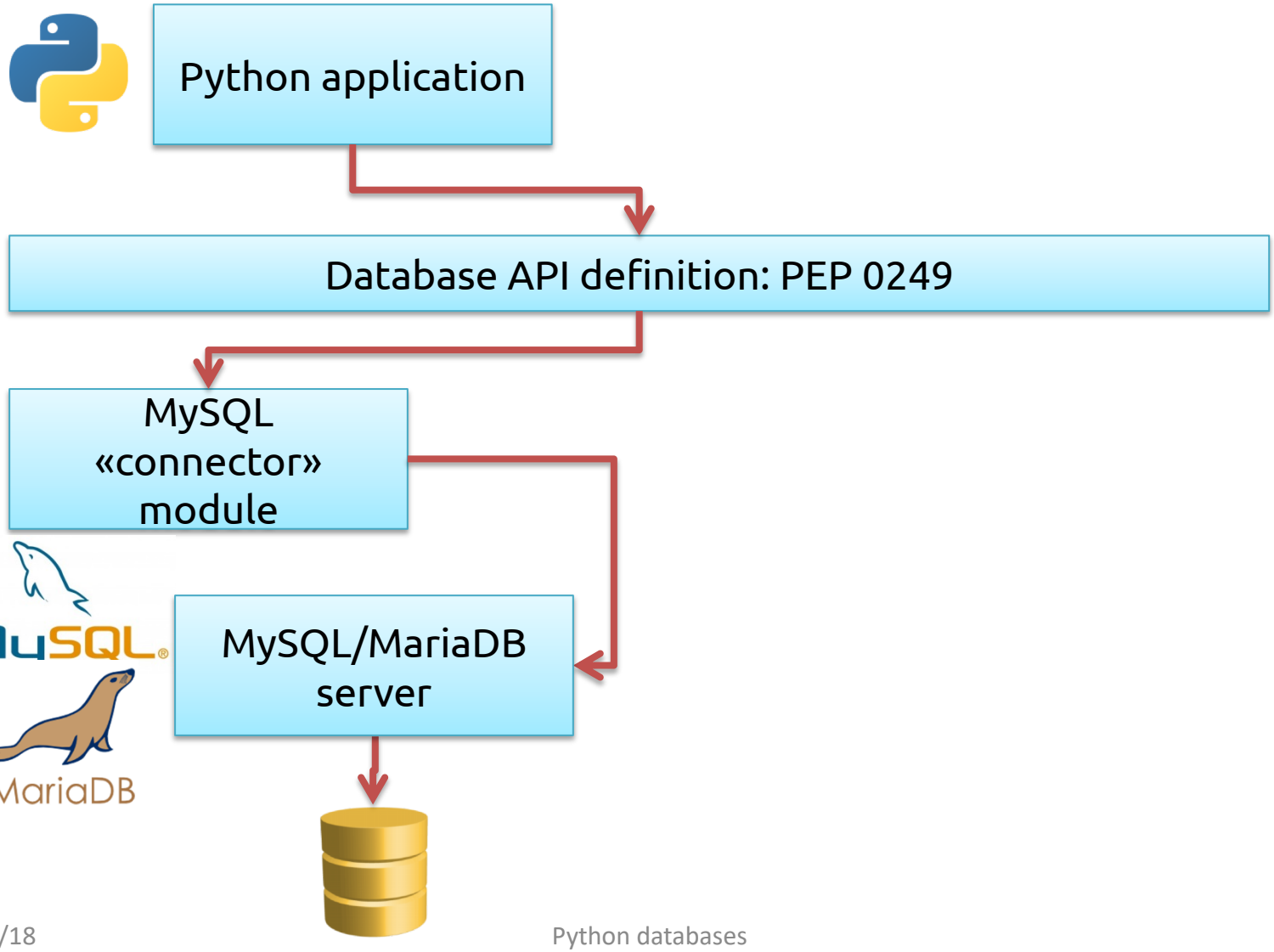
SQLite

- Open source file-based storage
- Software library integrated in your program (serverless)
- Self-contained
- https://www.sqlite.org/

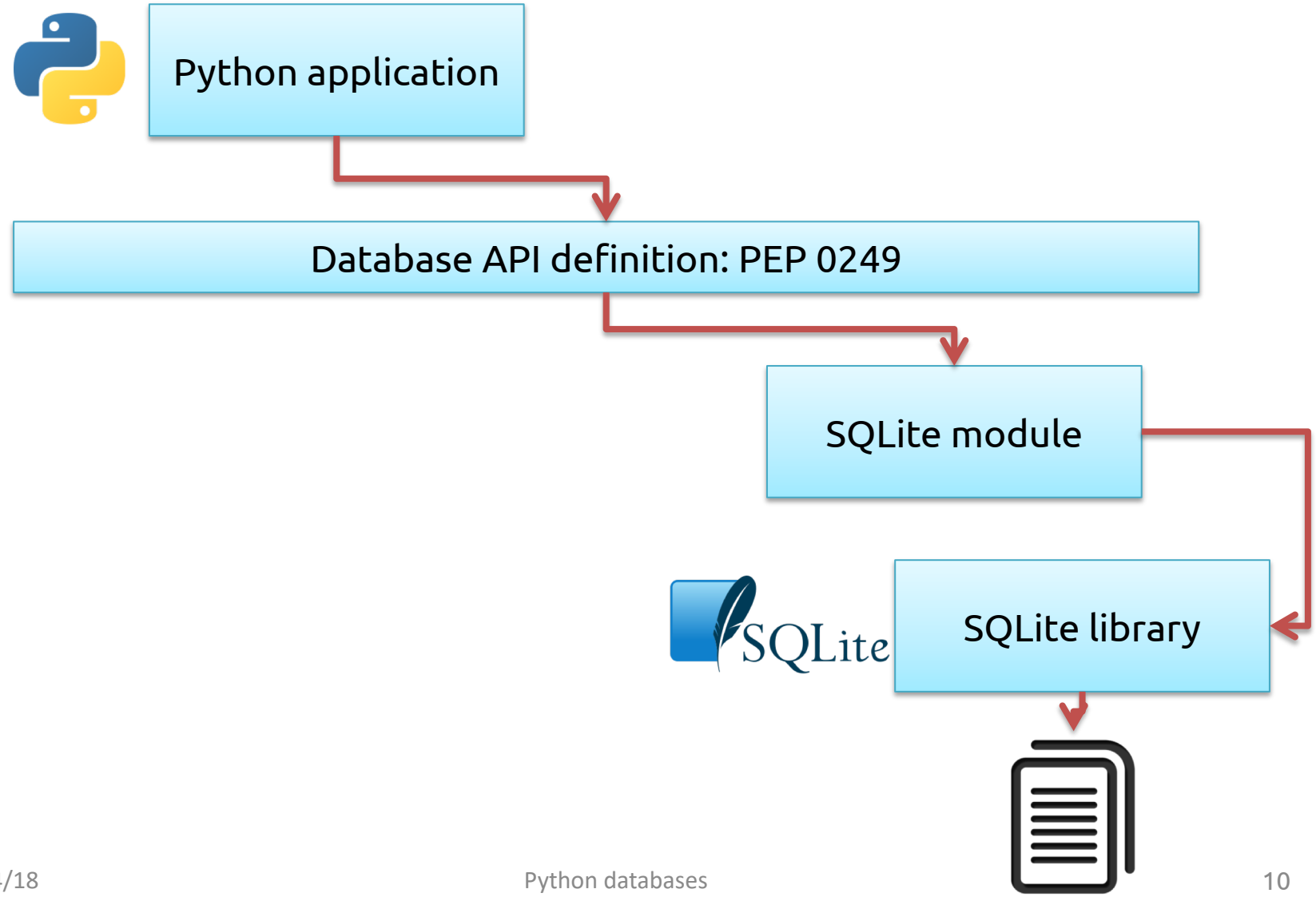
General Architecture



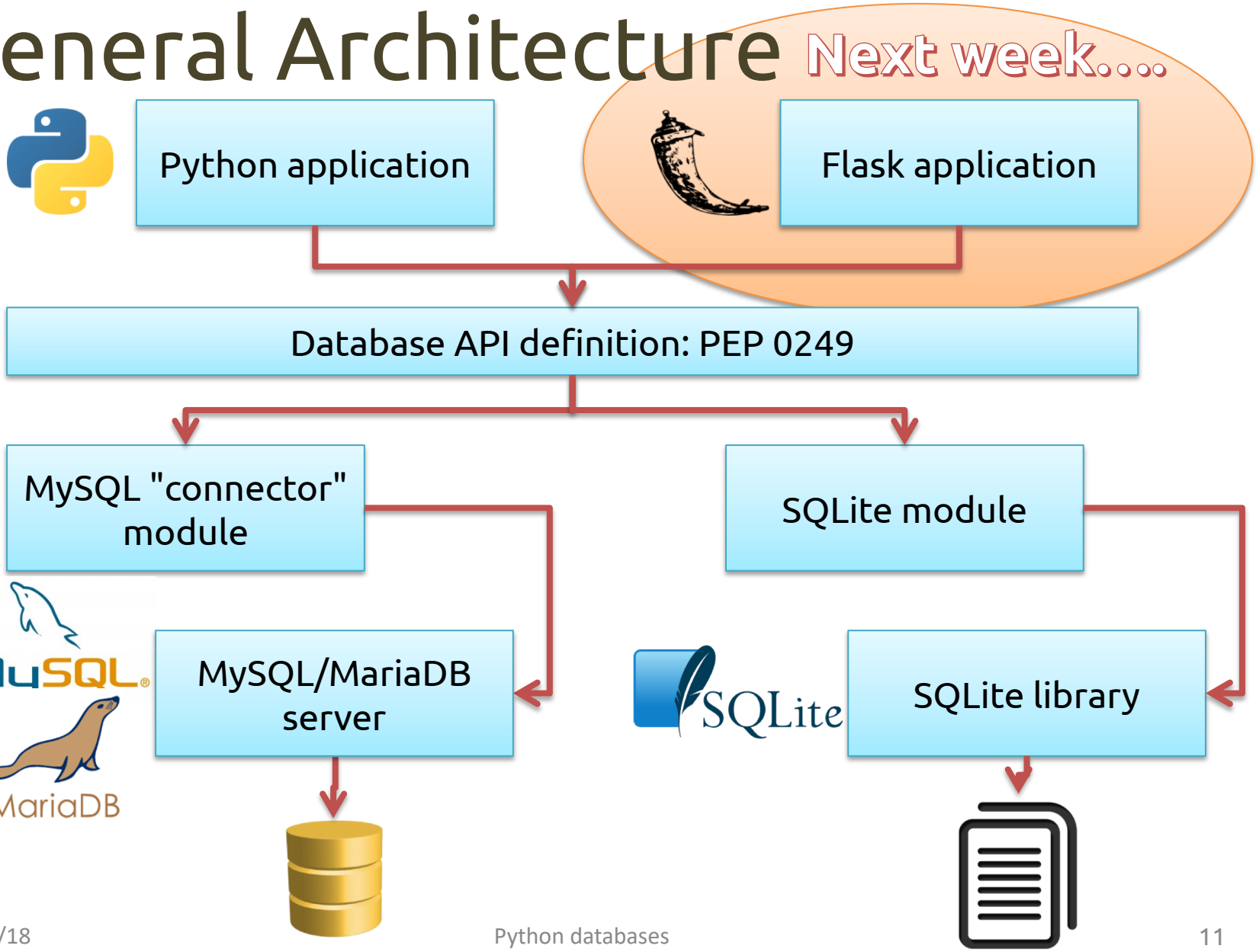
General Architecture



General Architecture



General Architecture



Other options

- PostgreSQL – more complex, but more complete than MySQL/MariaDB
- Non-relational databases (‘NoSQL’)
  - won’t be considered here

PEP 0249

- Python Database API Specification v2.0
  - https://www.python.org/dev/peps/pep-0249/
- Specifies a standard API that Python modules that are used to access databases should implement
- Does not provide a library nor a module
- Third party modules may adhere to these specifications

Main concepts in PEP 249

- Access to database is provided through a **connect** method, that returns a **Connection** object
- For executing queries, you need a **Cursor** object, that can be obtained by the Connection
- A **cursor** may **execute**() a SQL query, with parameters
- A **cursor** may **fetch** the **results** of the query

Minimal example

1. `sql = "SELECT id, original, modified FROM translation"`
2. `conn = mysql.connector.connect(user='root', password='', host='localhost', database='funnyecho')`
3.
```
cursor = conn.cursor()
cursor.execute(sql)
```
4. `translations = cursor.fetchall()`
5.
```
cursor.close()
conn.close()
```
6. `return translations`

Minimal example

```
1  sql = "SELECT id, original, modified FROM translation"

2  conn = mysql.connector.connect(user='root', password='',
   host='localhost', database='funnyecho')

3  cursor = conn.cursor()
   cursor.execute(sql)

4  translations = cursor.fetchall()

5  cursor.close()
   conn.close()

6  return translations
```

The **only** step that depends on the type of database

Step 1: defining the query

- Write a correct SQL statement, stored as a Python string
  - `sql = "SELECT id, original, modified FROM translation"`
- Variable arguments may be specified with ‘%s’ or '?' placeholders
  - `sql = "INSERT INTO translation (original, modified) VALUES (%s, %s)"`
  - `sql = "INSERT INTO translation (original, modified) VALUES (?, ?)"`

Placeholders

- Never use string concatenation over SQL statements. N-E-V-E-R! Huge security problems (SQL Injection)
- SQL statement "templates" that include placeholders
- Actual values passed in the .execute call
- Different libraries use different types of placeholder

Placeholder syntax

MySQL/MariaDB

- C-like format string
- `...WHERE name=%s`
- Beware: always use %s, even for numeric data – not %d or %f

SQLite

- Question mark
- `...WHERE name=?`

Step 2: Connecting to the database

- Depending on the library, use the provided ‘connect’ method
- The method parameters are dependent on the module implementation (non-standard)
  - ```
    conn = mysql.connector.connect(user='root',
    password='', host='localhost',
    database='funnyecho')
    ```
    ```

Step 5 (b): Clean up

- Don’t forget to close the connection, thus freeing up resources on the database server
  - `conn.close()`
- Write the close statement immediately, otherwise you’ll forget it
- Remember not to ‘return’ the function before cleaning up

Step 3: execute the query

- First, obtain a cursor from the connection
  - `cursor = conn.cursor()`
- Then, execute the query
  - `cursor.execute(sql)`
- Query parameters (%s/? placeholders) are specified as a ‘tuple’ argument
  - `cursor.execute(sql, (txtbefore, txtafter) )`
  - `cursor.execute(sql, (txtid,) )`
  - Beware: one-element tuples require trailing `,`

Step 4 (SELECT): Analyze the result

- Only if the query was a SELECT
- Use various methods of **cursor**:
  - `cursor.fetchone()` # next result
  - `cursor.fetchall()` # all remaining results
  - They return tuples, corresponding to the SELECT'ed columns
  - https://www.python.org/dev/peps/pep-0249/#cursor-methods

Step 4 (UPDATE): Commit the change

- For INSERT, UPDATE and DELETE there is no result
- The change is not applied immediately to the database, but needs to be «committed»
- `conn.commit()`
  - Will commit all pending executed queries in the connection
- Must be called before `conn.close()`
- Don't forget, or you'll lose your data

Step 5 (a): Clean up

- When the cursor is no longer needed
- `cursor.close()`

Step 6: Use the results

- Analyze the returned data, and do what the application requires for them.
- If further queries are needed, go back to step 3 (re-use the same Connection, creating new Cursors)

Using MySQL

- Pre-requisite: a working installation of the mysql server
  - `sudo apt-get install mysql-server`
  - or download from http://dev.mysql.com/downloads/mysql/
- Pre-requisite: a working installation of the mariadb server
  - `sudo apt-get install mariadb-server`

MySQL connectors

Official connector (Oracle)

- Download and install the "MySQL Connector for Python"
  - http://dev.mysql.com/downloads/connector/python/
  - Provides the package "mysql.connector"

Alternative (from pip)

- Pure Python implementation
  - https://github.com/PyMySQL/PyMySQL/
  - pip install PyMySQL
  - Provides the package "pymysql"
- Nearly drop-in replacement
- Easier to install

MySQL Python Connector

- To use: import mysql.connector
- Well-done documentation at
  - http://dev.mysql.com/doc/connector-python/en/index.html

Connecting with mysql (Oracle)

- Basic form
  - `import mysql.connector`
  - `cnx = mysql.connector.connect (`
    - `user='joe',`
    - `password='xxx',`
    - `database='test',`
    - `host='localhost' )`

- Additional parameters
  - http://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html

Connecting with mysql (Oracle)

- Alternate form
  - `import mysql.connector`
  - `params = {`
    - `'user': 'joe',`
    - `'password': 'xxx',`
    - `'host': 'localhost',`
    - `'database': 'test',`
    - `'use_unicode': True }`
  - `cnx = mysql.connector.connect(**params)`

Connecting with PyMySQL

```
– import pymysql
– cnx = pymysql.connect ( ... )
– cursor = cnx.cursor()
```

- ... Same connection parameters
- … Same placeholder (%s)
- … When in doubt, check the Oracle documentation

SQLite and Python

- SQLite is a simple file-based storage library
- Since Python 2.5, it is included by default, in the "sqlite3" package
  - https://docs.python.org/3/library/sqlite3.html
  - Developed at https://github.com/ghaering/pysqlite
- The «connection» just means specifying the file name
  - `import sqlite3`
  - `conn = sqlite3.connect('example.db')`
- Remember: placeholder = ?

References and Links

- MySQL: http://dev.mysql.com/
- MariaDB: http://mariadb.org/
- SQLite (C library): https://www.sqlite.org/
- SQLite for Python (installed by default):
  - documentation: https://docs.python.org/3/library/sqlite3.html
  - developer: https://github.com/ghaering/pysqlite
- PEP 249 "Python Database API Specification v2.0": https://www.python.org/dev/peps/pep-0249/
- PyMySQL "pure python" connector
  - https://github.com/PyMySQL/PyMySQL

Questions?

01QZP AMBIENT INTELLIGENCE

Fulvio Corno
Luigi De Russis
{name.surname}@polito.it

License

- This work is licensed under the Creative Commons “Attribution-NonCommercial-ShareAlike Unported (CC BY-NC-SA 4.0)” License.
- You are free:
  - to **Share** - to copy, distribute and transmit the work
  - to **Remix** - to adapt the work
- Under the following conditions:
  - **Attribution** - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
  - **Noncommercial** - You may not use this work for commercial purposes.
  - **Share Alike** - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-sa/4.0/