Models, ORMs, and SQLAlchemy

Models

- Often backed by a relational database.
- NoSQL databases are becoming more popular for some applications, but SQL databases are still the norm.
- Generally want to avoid writing raw SQL
  - Object Relational Models help here

Object-Relational Mappers (ORMs)

- Map rows of tables in database to objects in your application
- Modifications to the object automatically generate SQL to manipulate the underlying data

ORM advantages

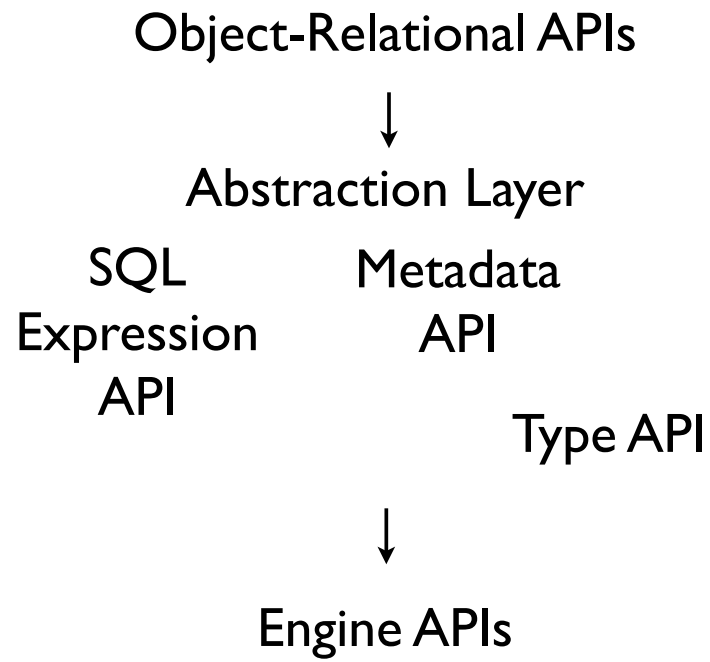
- Easier to work with underlying data
- More portable across different database engines
- Handle things like connection pools and multithreading
- Abstract away the SQL

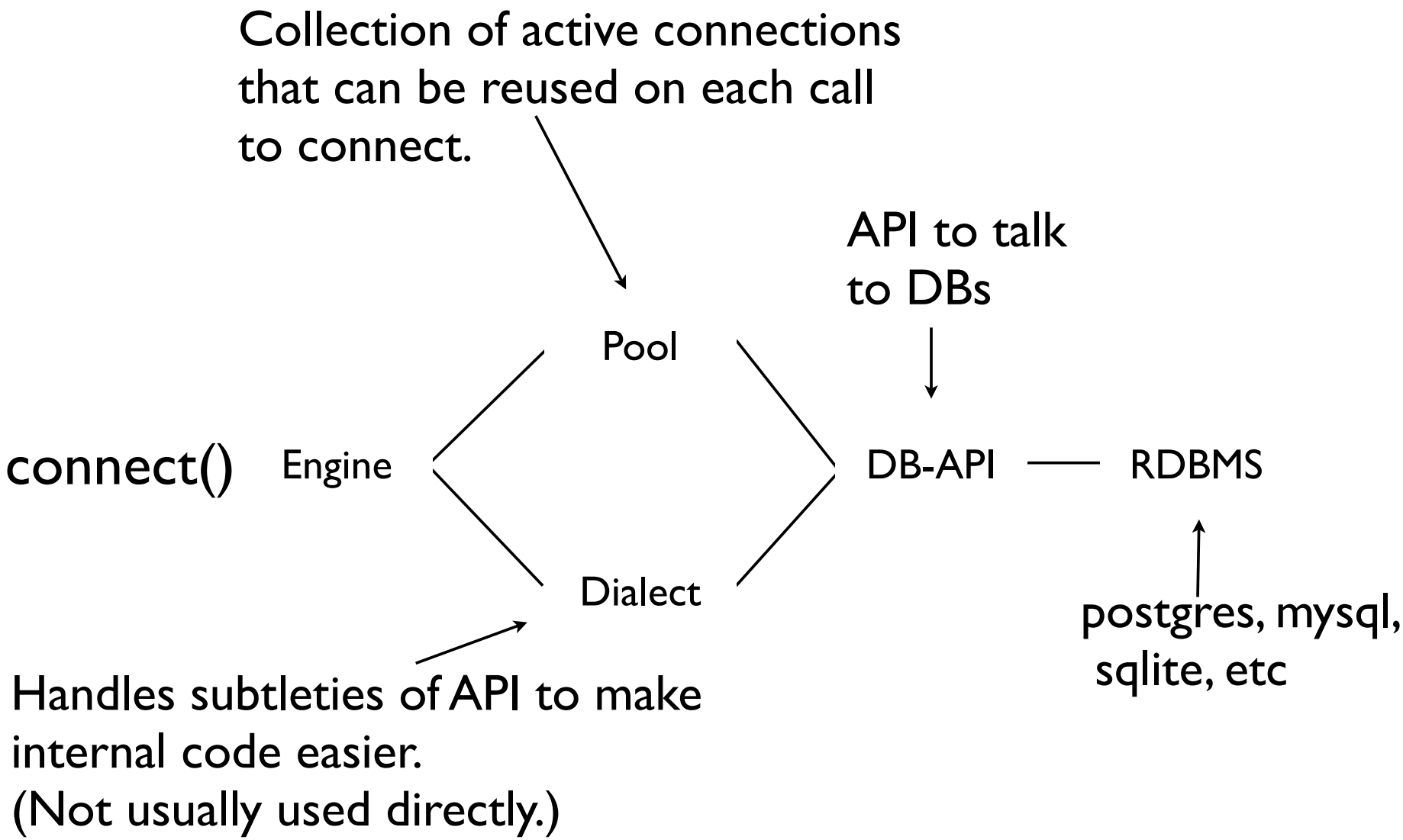
ORM disadvantages

- Abstracting away SQL means less control
  - especially for performance
- Easy to write inefficient code
- Can be hard to track down problems in the ORM.

SQLAlchemy

- Currently most popular Python ORM




Collection of active connections
that can be reused on each call
to connect.
API to talk
to DBs
Pool
connect()
Engine
DB-API
RDBMS
Dialect
postgres, mysql,
sqlite, etc
Handles subtleties of API to make
internal code easier.
(Not usually used directly.)

DB Review and Intro to SQLAlchemy

1. Connect to database
2. Define and create a table
3. Define a Python class to map to table
4. Set up mapping
5. Create a session
6. Add new objects
7. Querying

Identity Map

- All operation on a row in A Session operate on the same set of data.
- An sql query with a particular primary key in a session always maps to the same Python object.