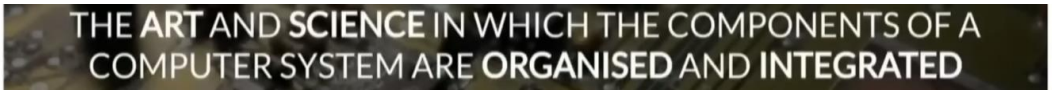
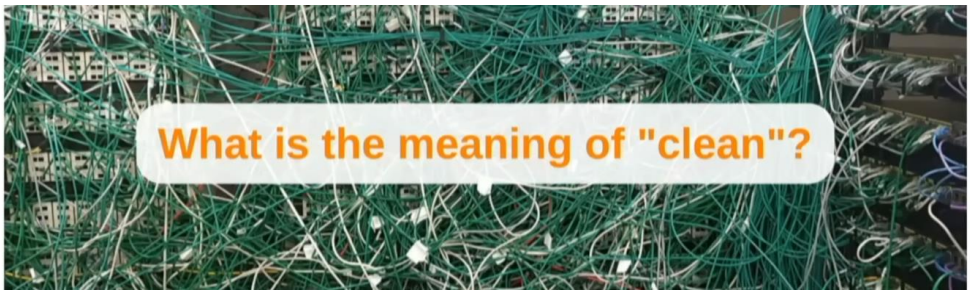
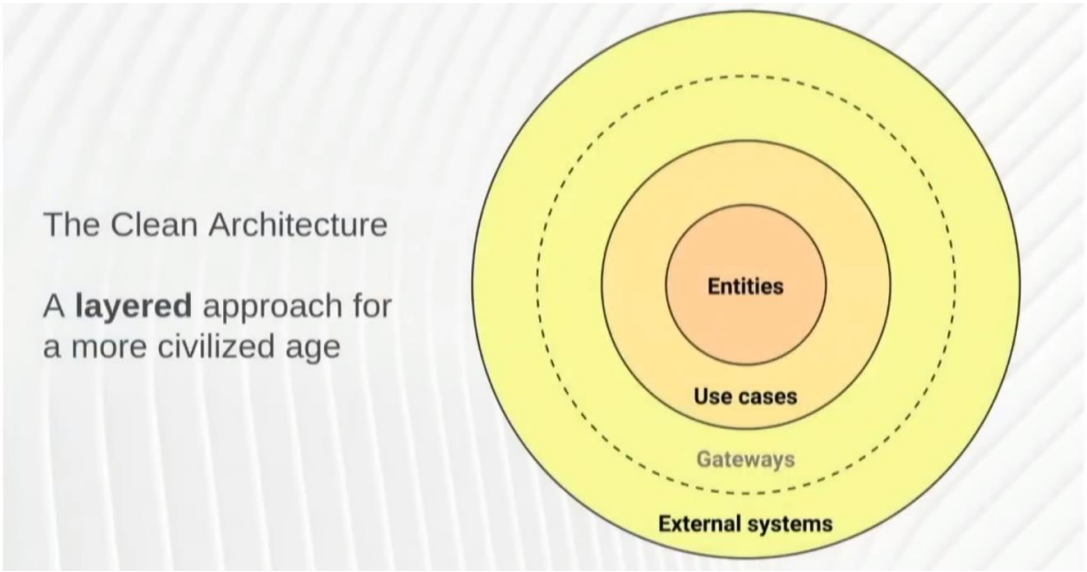
Architectural considerations are often overlooked by developers or completely delegated to a framework. We should start once again discussing how applications are structured, how components are connected and how to lower coupling between different parts of a system, to avoid creating software that cannot easily be maintained or changed. The ""clean architecture"" model predates Robert Martin, who recently brought it back to the attention of the community, and is a way of structuring applications that leverages layers separation and internal APIs to achieve a very tidy, fully-tested, and loosely coupled system.

The talk introduces the main ideas of the architecture, showing how the layers can be implemented in Python, following the content of the book ""Clean Architectures in Python"" edited by Leanpub. The book recently reached 25,000 downloads and many readers found it useful to start learning how to test software and how to structure an application without relying entirely on the framework.



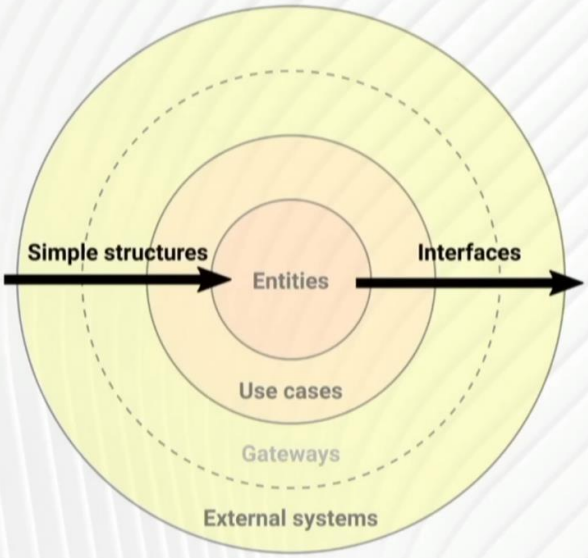




Components will belong to only one layer and can see everything defined within that layer only, it clarifies dependencies

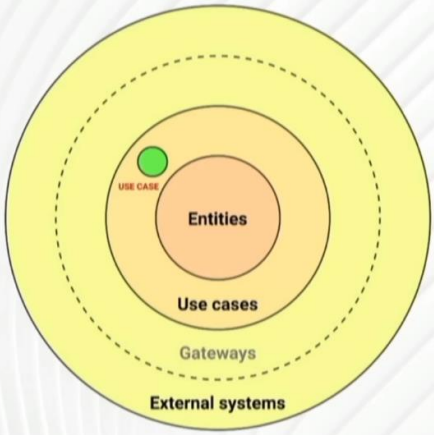
The golden rule

Talk inward with **simple structures**, talk outwards through **interfaces**.



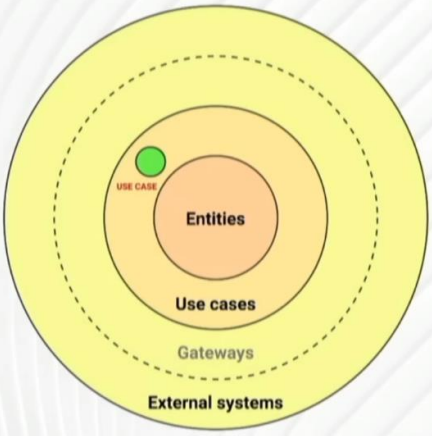
Use case: retrieve a list of items

```
def item_list_use_case():
    pass
```



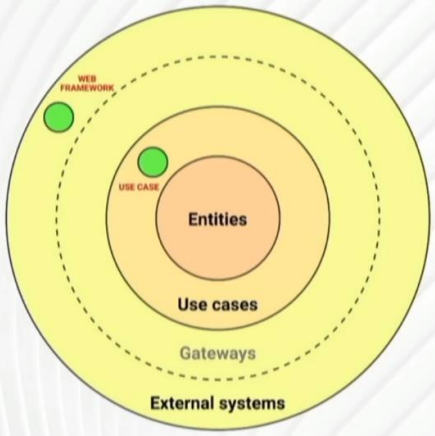
Entities: simple models

```
class Item:
    def __init__(self, code, price):
        self.code = code
        self.price = price
```



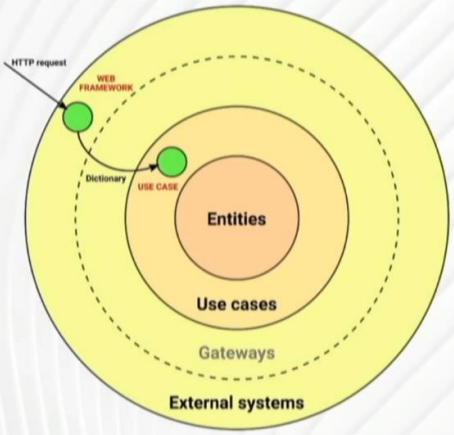
We want to build a **web application**

```
@blueprint.route('/items', methods=['GET'])
def items():
    pass
```



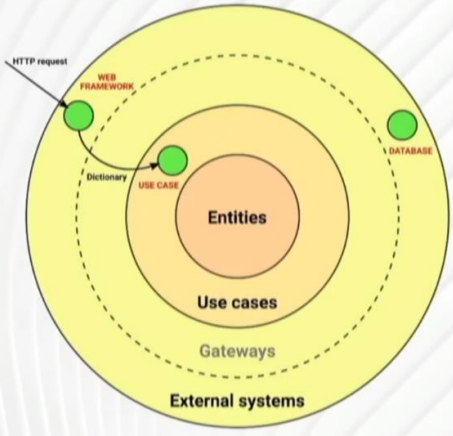
Incoming HTTP requests become
a **call** and **simple structures**

```
@blueprint.route('/items', methods=['GET'])
def items():
    item_list_use_case(request.args)
```



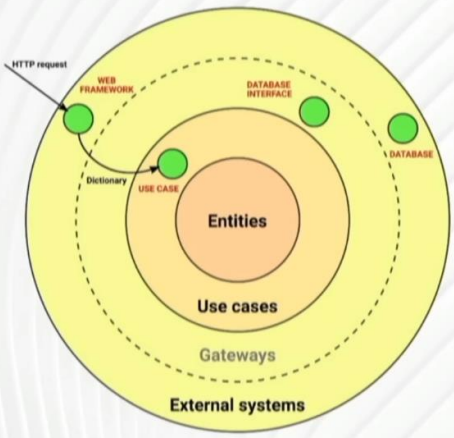
The data is stored in a **repository**

```
@blueprint.route('/items', methods=['GET'])
def items():
    item_list_use_case(request.args)
```



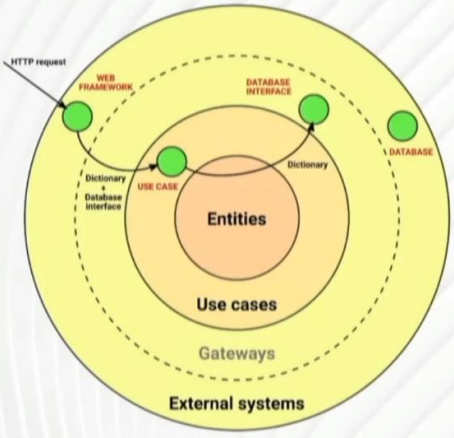
The repository can be accessed only through an **interface**

```python
@blueprint.route('/items', methods=['GET'])
def items():
    item_list_use_case(request.args)
```



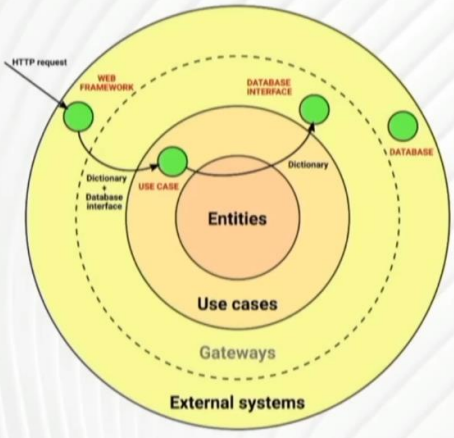
The use case **receives** the repository interface as an argument of the call

```python
@blueprint.route('/items', methods=['GET'])
def items():
    repo = PostgresRepo(CONNECTION_STRING)
    item_list_use_case(repo, request.args)
```



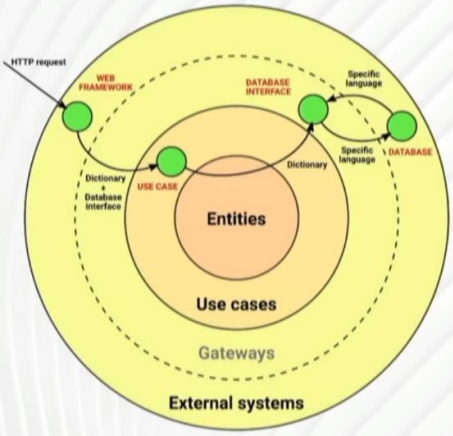
The use case queries the repository interface with **simple structures**

```python
def item_list_use_case(repo, params):
    # BUSINESS LOGIC
    # e.g. prepare parameters

    repo.list(params)
```



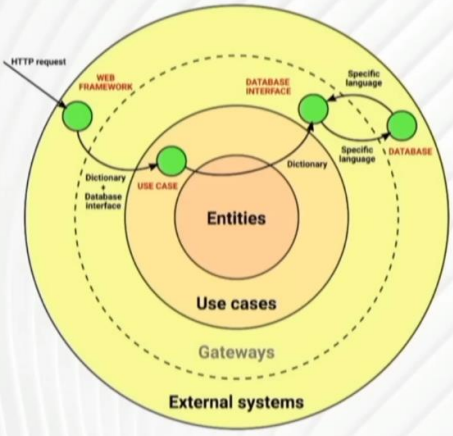
The repository interface and the repository exchange data in a **specific language**

```
class PostgresRepo:
    def __init__(self, CONNECTION_STRING):
        self.ng = create_engine(CONNECTION_STRING)
        Base.metadata.bind = self.ng

    def list(self, filters):
        DBSession = sessionmaker(bind=self.ng)
        session = DBSession()
        query = ...
```



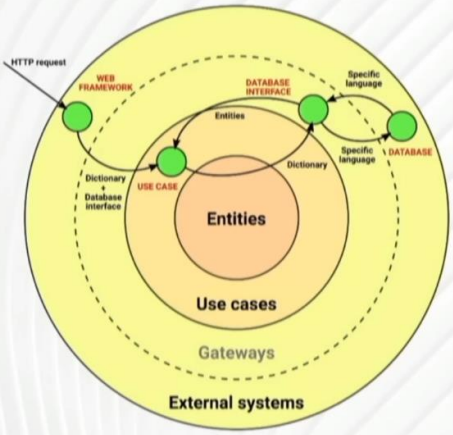
The repository interface translates the specific language into **simple structures** and **entities**

```
class PostgresRepo:
    def __init__(self, CONNECTION_STRING):
        self.ng = create_engine(CONNECTION_STRING)
        Base.metadata.bind = self.ng

    def _create_items(self, results):
        return [
            Item(code=q.code, price=q.price)
            for q in results
        ]

    def list(self, filters):
        DBSession = sessionmaker(bind=self.ng)
        session = DBSession()
        query = ...

        return self._create_items(query.all())
```



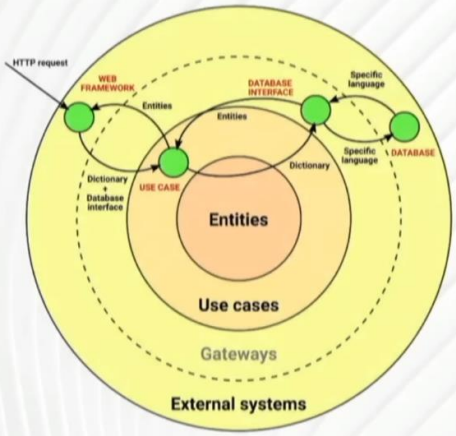
The use case queries the repository interface with **simple structures**

```
def item_list_use_case(repo, params):
    # BUSINESS LOGIC
    # e.g. prepare parameters

    result = repo.list(params)

    # BUSINESS LOGIC
    # e.g. further filtering

    return result
```



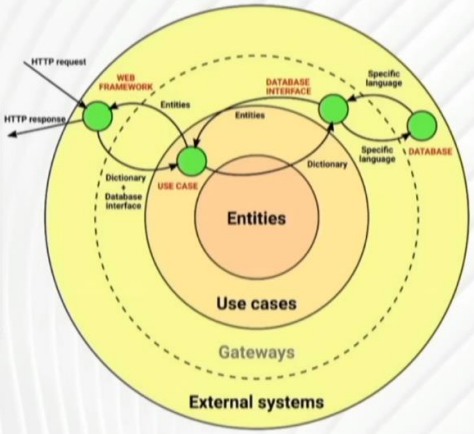
The use case returns the result of the business logic:
entities and **simple structures**

```
@blueprint.route('/items', methods=['GET'])
def items():
    repo = PostgresRepo(CONNECTION_STRING)
    result = item_list_use_case(repo, request.args)
```



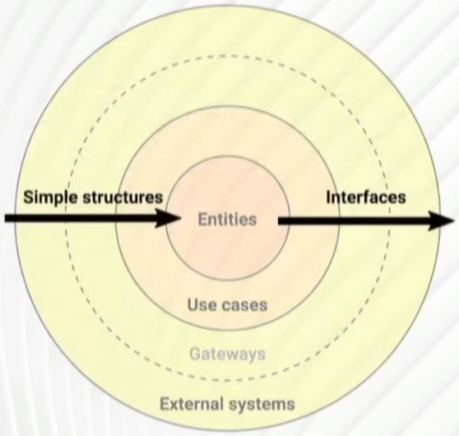
The web framework converts entities and simple structures into **HTTP responses**

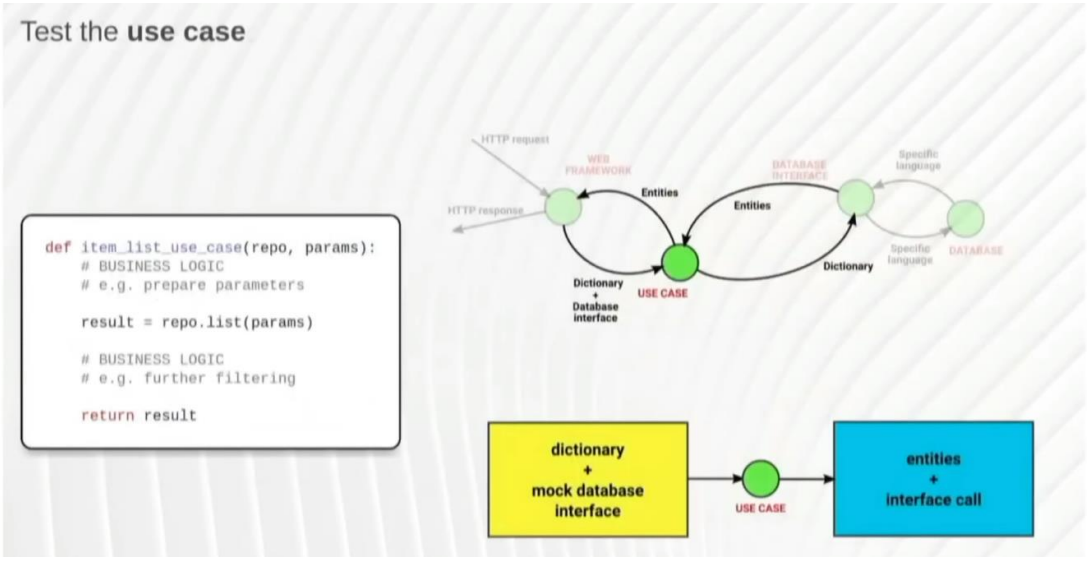
```
@blueprint.route('/items', methods=['GET'])
def items():
    repo = PostgresRepo(CONNECTION_STRING)
    result = item_list_use_case(repo, request.args)

    return Response(
        json.dumps(result),
        mimetype='application/json',
        status=200
    )
```



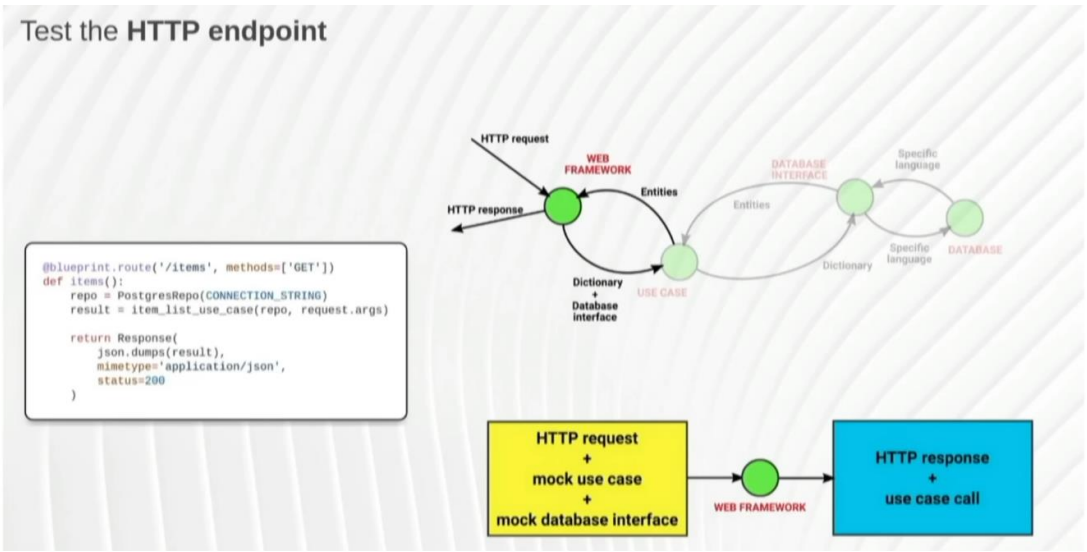
Dependency injection: use interfaces to reduce coupling

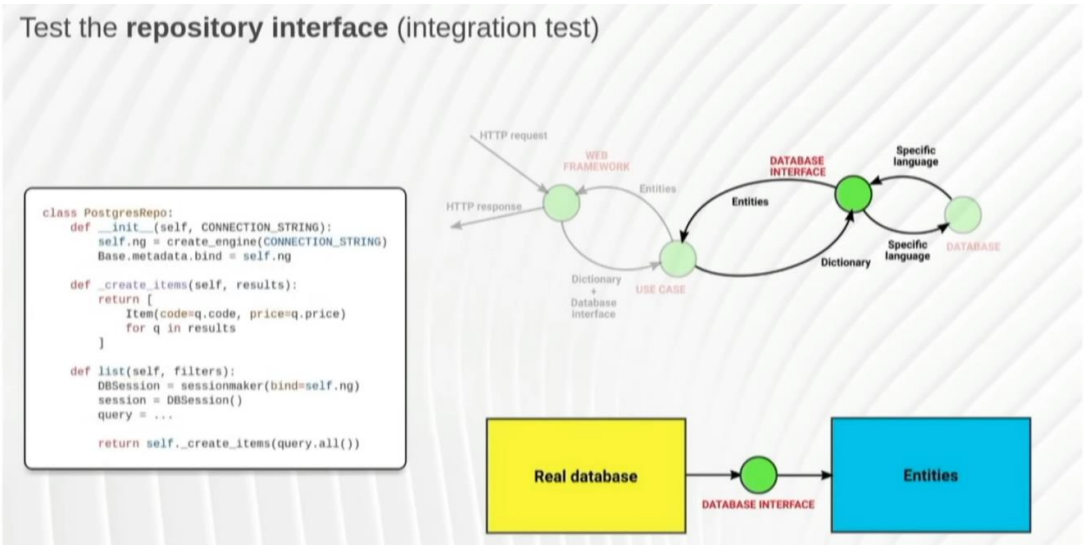
```
def item_list_use_case():
    repo = PostgresRepo(CONNECTION_STRING)
    ...
    repo.list()

---

repo = PostgresRepo(CONNECTION_STRING)
def item_list_use_case(repo):
    repo.list()
```





We don’t need the DB to write our business logic tests because we can use mocks

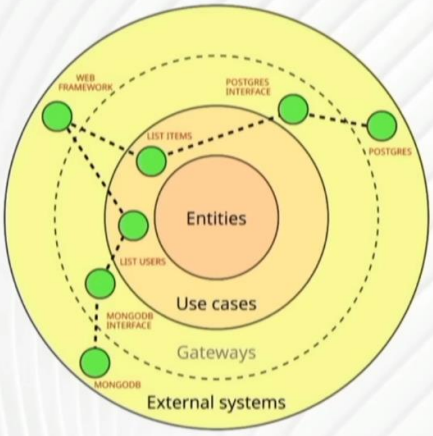




This requires the DB because we are testing that the DB façade works

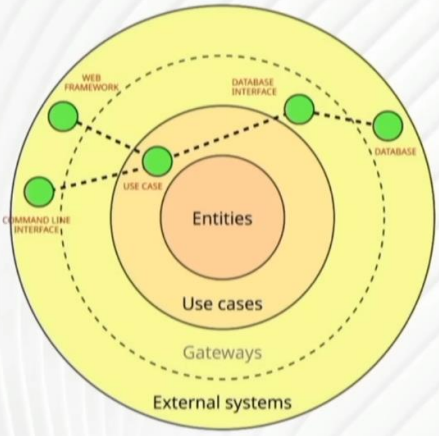
The web framework can easily run use cases that connect to **different repositories**

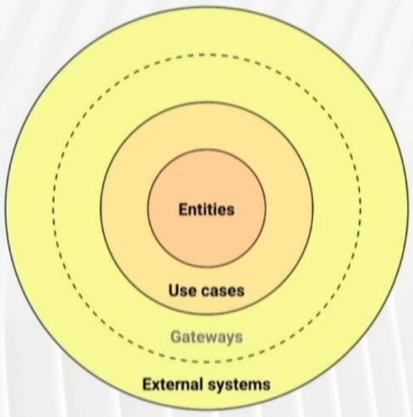
```
@blueprint.route('/items', methods=['GET'])
def items():
    repo = PostgresRepo(CONNECTION_STRING)
    result = item_list_use_case(repo, request.args)
    ...

@blueprint.route('/users', methods=['GET'])
def users():
    repo = MongoRepo(CONNECTION_STRING)
    result = users_list_use_case(repo, request.args)
    ...
```



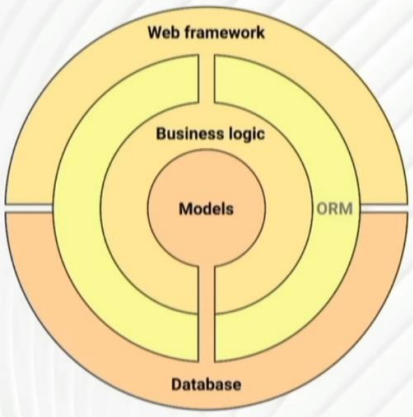
The same use case can be called by **different front-ends**

```
@blueprint.route('/items', methods=['GET'])
def items():
    repo = PostgresRepo(CONNECTION_STRING)
    result = item_list_use_case(repo, request.args)
    ...

@click.command()
@click.option('--count', default=1)
def print_items(count):
    repo = PostgresRepo(CONNECTION_STRING)
    args = {'count': count}
    result = item_list_use_case(repo, args)
    ...
```





The Clean Architecture



The Django Architecture

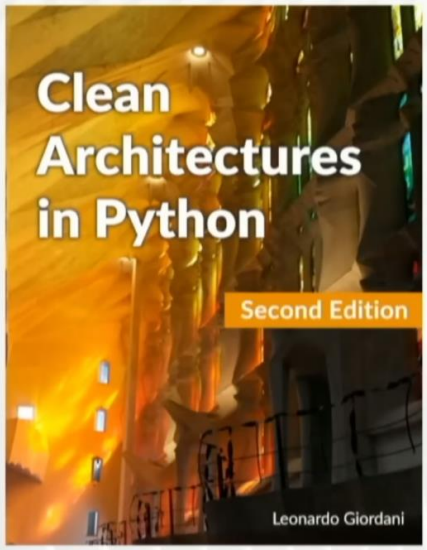
Awesome! Let's do it now!

Netscape: lesson learned

Migrations happen one step at a time

Is this the **definitive** architecture?

IT DEPENDS



Leonardo Giordani
Clean Architectures in Python

`bit.ly/getpycabook`