Pytest Design Patterns

Miloslav Pojman

EuroPython 2024, Prague

Miloslav Pojman

twitter.com/MiloslavPojman
fosstodon.org/@mila

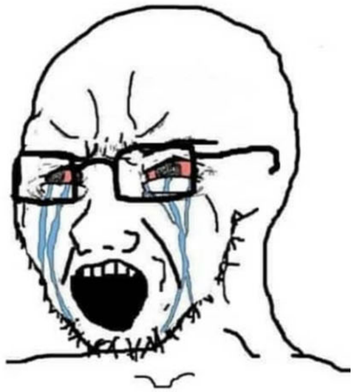
PURESTORAGE®

KISS Approach to Testing

Are the tests worth it?

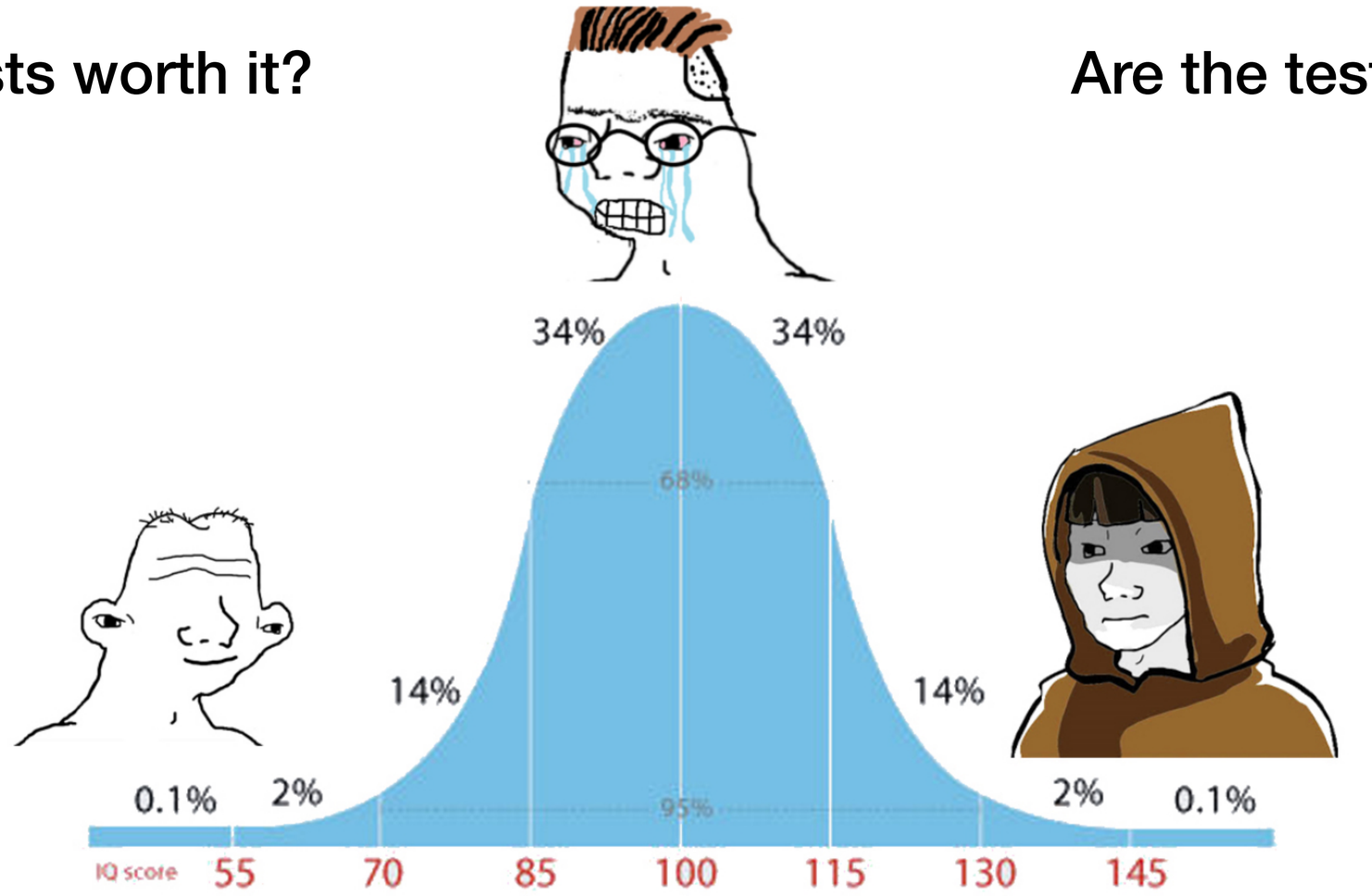
Are the tests worth it?

At least 80% coverage!



At least 80% coverage!

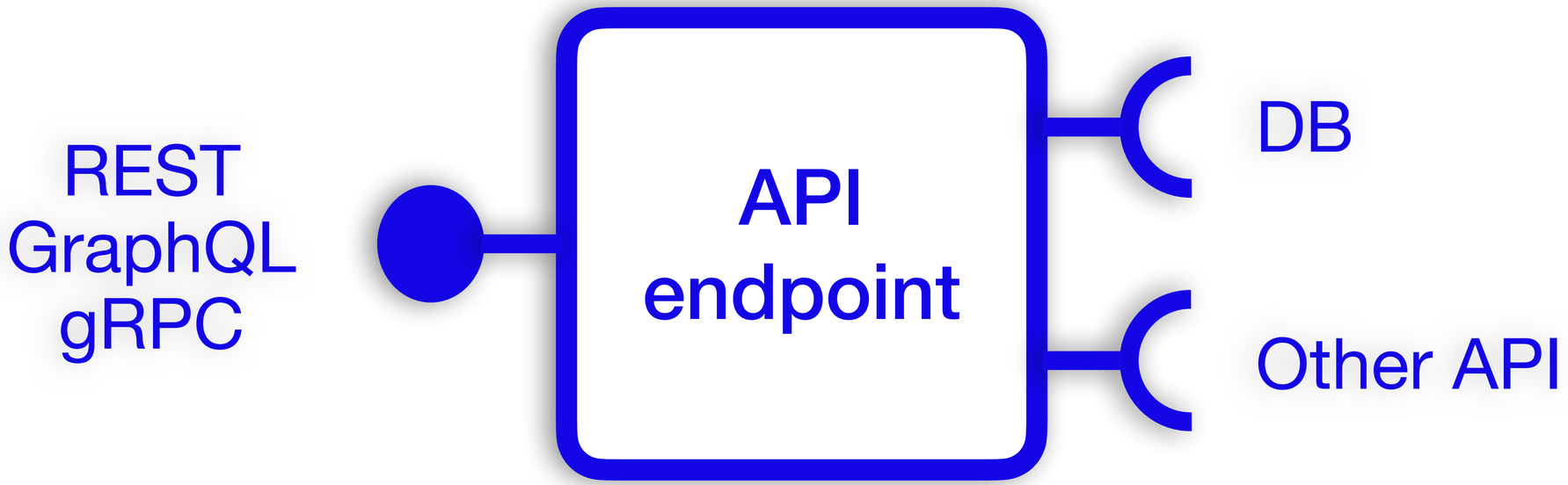
Are the tests worth it?



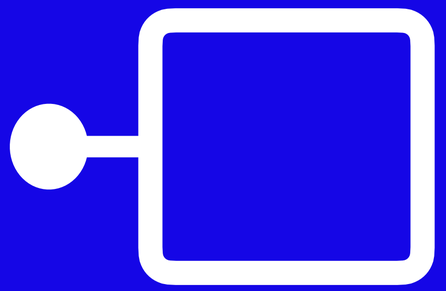
Are the tests worth it?

Understandable

Maintainable


REST
GraphQL
gRPC
API
endpoint
DB
Other API

Calling the Application



Test Client

Testing WSGI or ASGI Interface

```
def test_index(client) -> None:
    assert client.get("/").status_code == 200
```

```
def test_index(client) -> None:
    assert client.get("/").status_code == 200
```

```
@pytest.fixture(name="client")
def client_fixture(app):
    with TestClient(app=app) as client:
        yield client
```

Application Fixture

Simplifying Access and Customization

```python
@pytest.fixture(name="app")
def app_fixture():
    yield app
```

```python
@pytest.fixture(name="client")
def client_fixture(app):
    with TestClient(app=app) as client:
        yield client
```

```python
@pytest.fixture(name="app")
def app_fixture():
    app.dependency_overrides = {
        # ...
    }
    yield app
    app.dependency_overrides = {}
```

Authentication

Calling Endpoints with Valid Credentials

```
def test_admin_endpoint(client, admin_user):
    client.login(admin_user)
    # ...
```

```python
def test_admin_endpoint(client, admin_user):
    client.login(admin_user)
    # ...
```

```python
class MyTestClient(TestClient):
    def login(self, user: User):
        self.headers["Authentication"] = (
            f"Bearer {make_token(user)}"
        )
        # self.auth = ...
        # self.cookies["..."] = ...
```

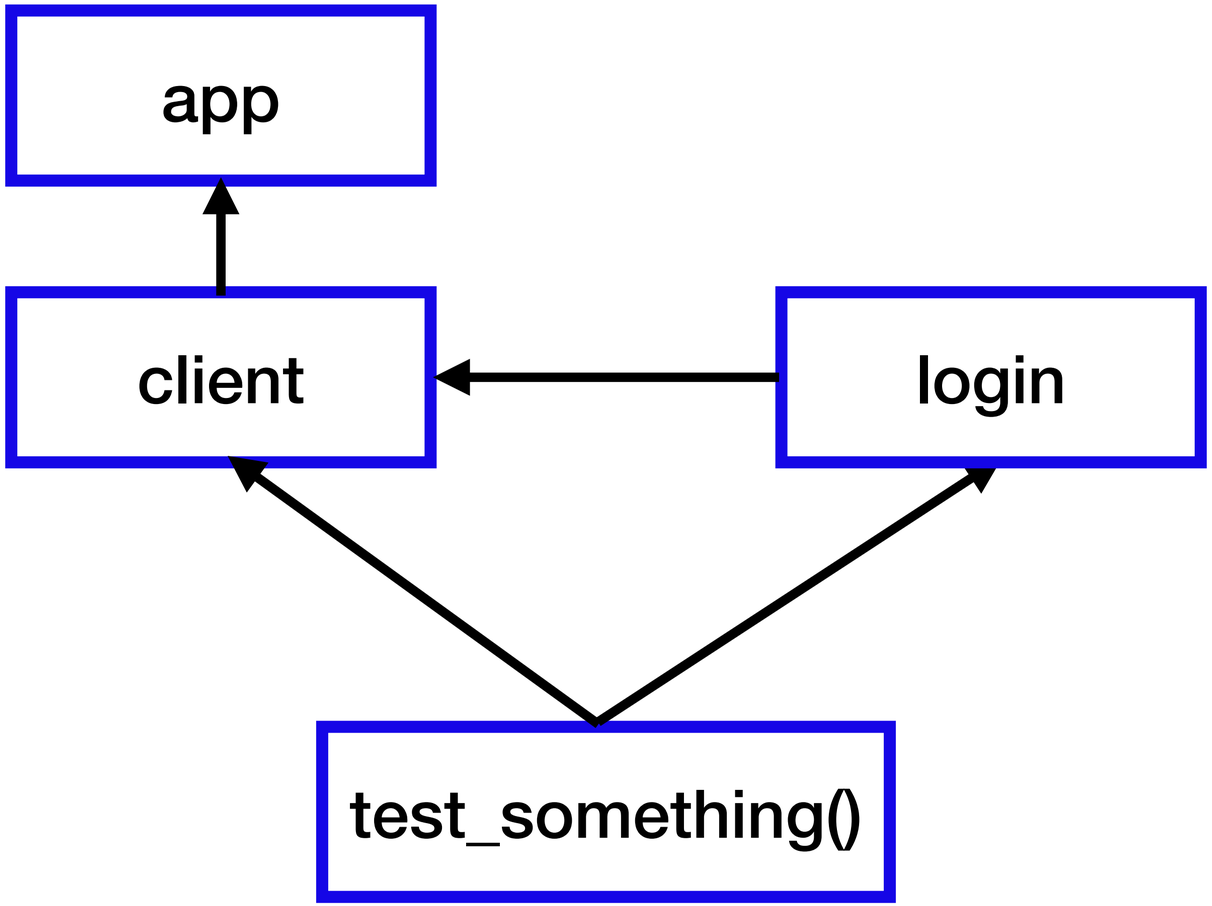
```
class TestSomethingUnrelatedToAuth:
    pytestmark = pytest.mark.usefixtures(
        "login"
    )
```

```
class TestSomethingUnrelatedToAuth:
    pytestmark = pytest.mark.usefixtures(
        "login"
    )
```

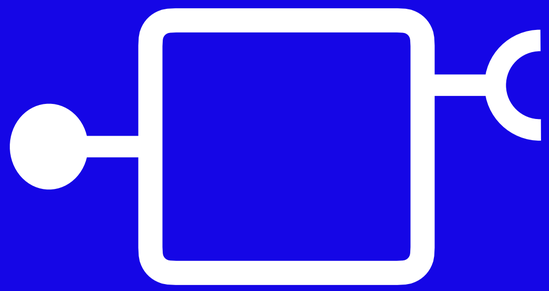
```
@pytest.fixture(name="login")
def login_fixture(client, create_user):
    user = create_user()
    client.login(user)
    return user
```

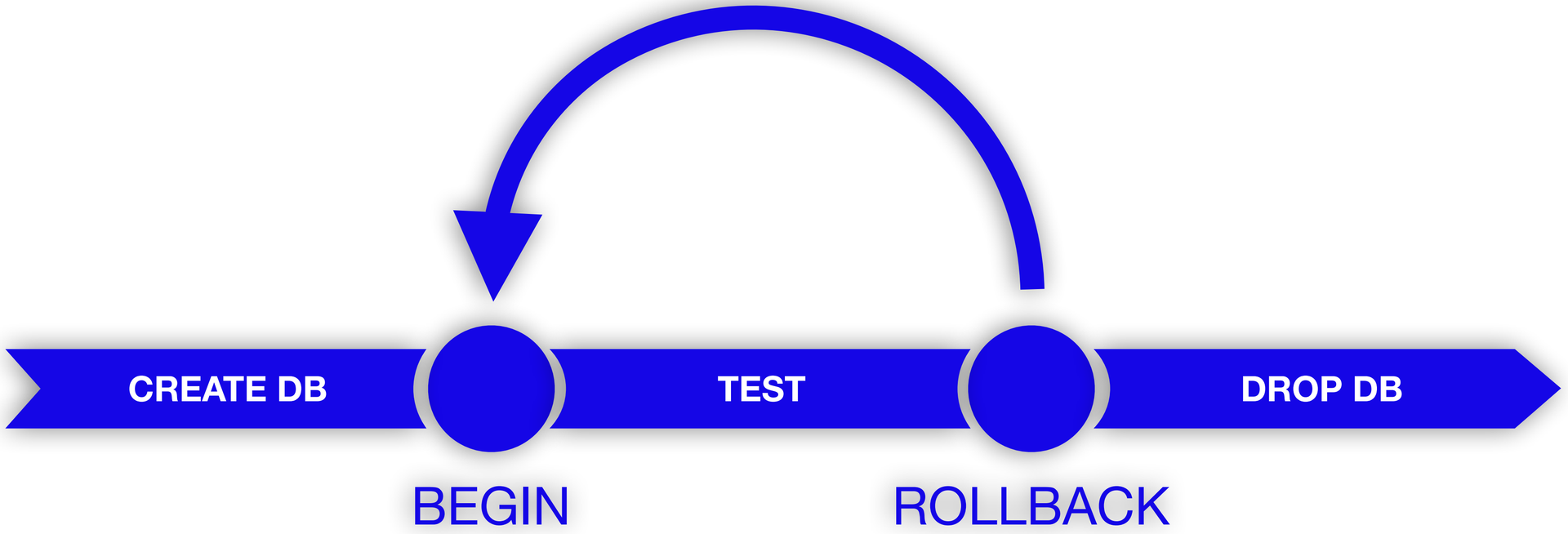
```python
def test_admin_role(client, login):
    login.add_role("staff")
    assert client.get("/admin").status_code == 200
```

```python
@pytest.fixture(name="login")
def login_fixture(client, create_user):
    user = create_user()
    client.login(user)
    return user
```


app
client
login
test_something()

Isolating a Database




CREATE DB
TEST
DROP DB
BEGIN
ROLLBACK

Transaction-Bound Tests

SQLAlchemy Example

```
@pytest.fixture(name="db_connection")
def db_connection_fixture(db_engine):
    with db_engine.connect() as db_connection:
        db_connection.begin()
        try:
            yield db_connection
        finally:
            db_connection.rollback()
```

```python
@pytest.fixture(name="db_connection")
def db_connection_fixture(db_engine):
    with db_engine.connect() as db_connection:
        db_connection.begin()
        try:
            yield db_connection
        finally:
            db_connection.rollback()
```

```python
@pytest.fixture(name="db_session")
def db_session_fixture(db_connection):
    with Session(
        db_connection,
        join_transaction_mode="create_savepoint",
    ) as db_session:
        yield db_session
```

https://docs.sqlalchemy.org/en/20/orm/session_transaction.html#joining-a-session-into-an-external-transaction-such-as-for-test-suites

```python
@pytest.fixture(name="db_session")
def db_session_fixture(db_connection):
    with Session(
        db_connection,
        join_transaction_mode="create_savepoint",
    ) as db_session:
        yield db_session
```

```python
@pytest.fixture(name="app")
def app_fixture(db_session):
    app.dependency_overrides = {
        db_session_dep: lambda: db_session,
    }
    yield app
    app.dependency_overrides = {}
```

Levels of Abstraction

No One-Size-Fits-All Solution

Test Data the Wrong Way

Global Database Fixtures

```
def test_user_filter(client):
    response = client.get("/users?org=7")
    data = response.json()
    assert [item["username"] for item in data] == [
        "user1", "user3", "user_super_admin"
    ]
```

Factory Functions

Be Explicit in the Tests

```
def test_user_filter(client):
    org = create_org()
    create_user(username="alice", org=org)
    create_user(username="bob")
    response = client.get(f"/users?org={org.id}")
    data = response.json()
    assert [item["username"] for item in data] == [
        "alice",
    ]
```

```python
def test_user_filter(client, db_session):
    org = create_org(db_session)
    create_user(db_session, username="alice", org=org)
    create_user(db_session, username="bob")
    response = client.get(f"/users?org={org.id}")
    data = response.json()
    assert [item["username"] for item in data] == [
        "alice",
    ]
```

Factory Functions as Pytest Fixtures

Avoid Boilerplate Code

```
def test_user_filter(client, create_user, create_org):
    org = create_org()
    create_user(username="alice", org=org)
    create_user(username="bob")
    response = client.get(f"/users?org={org.id}")
    data = response.json()
    assert [item["username"] for item in data] == [
        "alice",
    ]
```

```
def test_user_filter(client, create_user, create_org):
    org = create_org()
    create_user(username="alice", org=org)
    create_user(username="bob")
    response = client.get(f"/users?org={org.id}")
    data = response.json()
    assert [item["username"] for item in data] == [
        "alice",

    ]
```

```
@pytest.fixture(name="create_user")
def create_user_fixture(db_session):
    def create_user(username="test", password="", org=None):
        user = User(
            username=username, password=password, org=org
        )
        db_session.add(user)

    return create_user
```

```
@pytest.fixture(name="create_user")
def create_user_fixture(db_session):
    def create_user(username="test", password="", org=None):
        user = User(
            username=username, password=password, org=org
        )
        db_session.add(user)

    return create_user
```

```python
@pytest.fixture(name="create_user")
def create_user_fixture(db_session, create_org):
    def create_user(username="test", password="", org=None):
        if org is None:
            org = create_org()
        user = User(
            username=username, password=password, org=org
        )
        db_session.add(user)

    return create_user
```

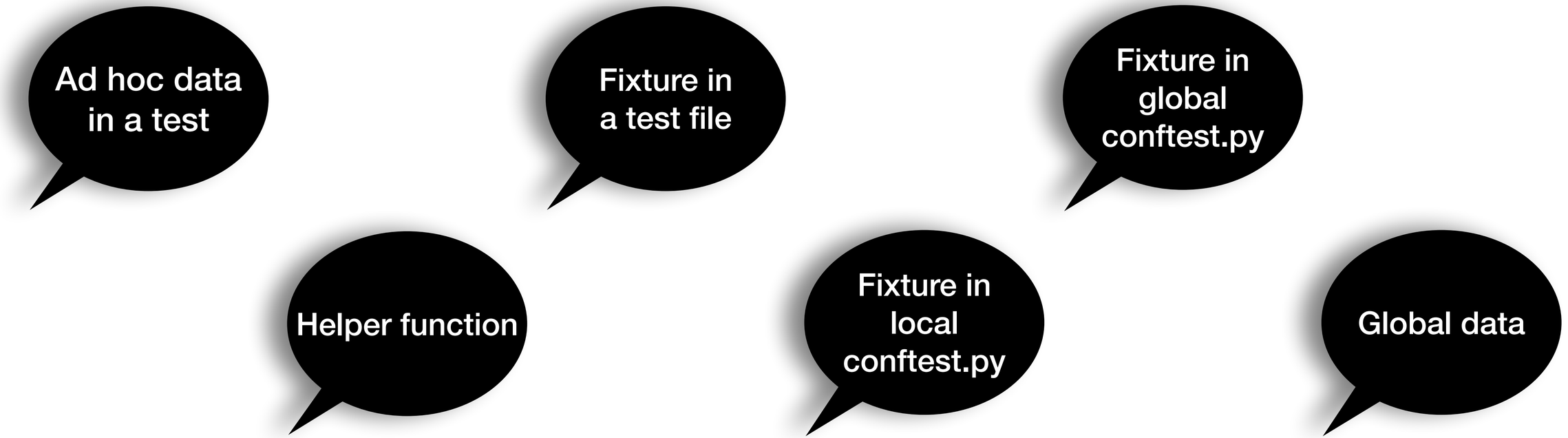
Getting Bigger

```
def test_user_comments(
    client,
    create_user,
    create_org,
    create_post,
    create_comment,
) -> None:
    org = create_org()
    # ...
```

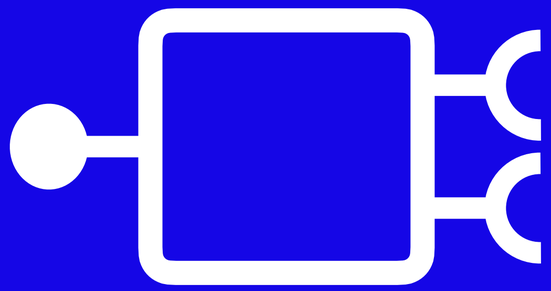
```
def test_user_comments(
    client,
    model_factory,
) -> None:
    org = model_factory.create_org()
    # ...
```

Fixture Scope

Prefer Fixtures Close to Tests



Faking the Network



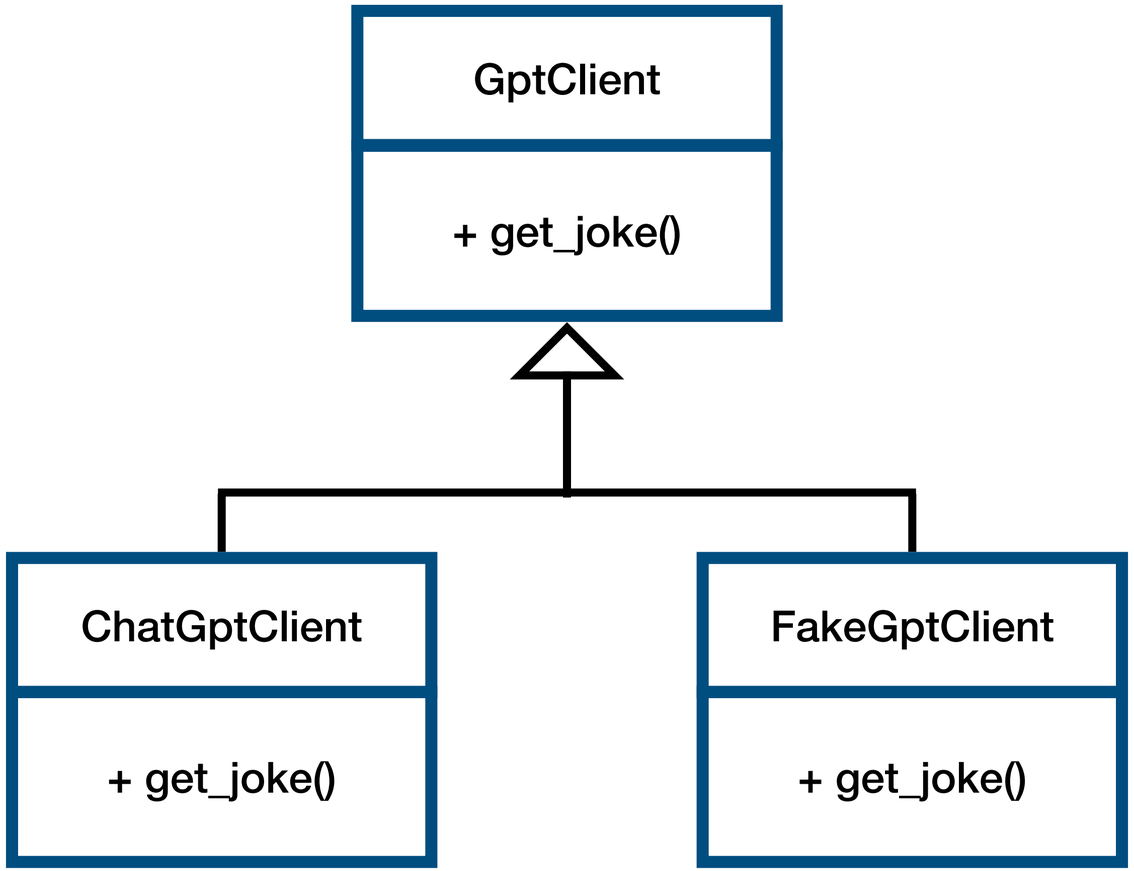
Monkey Patching and MagicMock

Mocking The Wrong Way

```
def test_make_joke(mocker):
    mock = mocker.patch("openai.chat.completions.create")
    mock.return_value.choices[0].message.content = "..."
    assert make_joke() == "..."
```

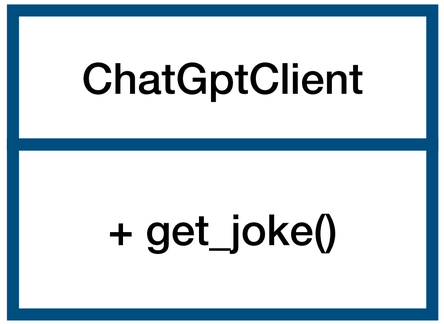
Introducing an Interface

Proper Mocking



Wrap API Clients

Limit the Scope of the Problem



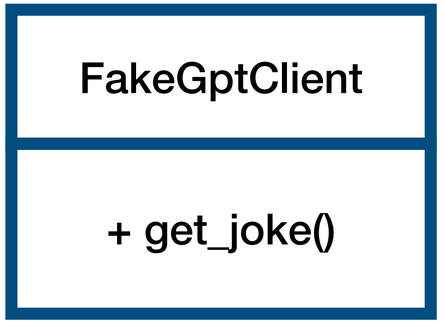

```
class ChatGptClient(GptClient):
    model = "gpt-3.5-turbo"
    prompt = "Give me a joke about the following topic:"

    def __init__(self, *, api_key):
        self._client = openai.OpenAI(api_key=api_key)

    def get_joke(self, topic):
        completion = self._client.chat.completions.create(
            messages=[
                {
                    "role": "system",
                    "content": self.prompt,
                },
                {
                    "role": "user",
                    "content": topic,
                },
            ],
            model=self.model,
        )
        result = completion.choices[0].message.content
        if result is not None
```

Fake Clients

Why to Mock when You Can Fake




```
class FakeGptClient(GptClient):
    def get_joke(self, topic):
        return f"Joke about {topic}"
```

Proper Patching

Dependency Injection (If Possible)

```
class FakeGptClient(GptClient):
    def get_joke(self, topic):
        return f"Joke about {topic}"
```

```
@pytest.fixture(name="fake_gpt")
def fake_gpt_fixture():
    return FakeGptClient()
```

```python
@pytest.fixture(name="fake_gpt")
def fake_gpt_fixture():
    return FakeGptClient()
```

```python
@pytest.fixture(name="app")
def app_fixture(fake_gpt):
    app.dependency_overrides = {
        gpt_dep: lambda: fake_gpt,
    }
    yield app
    app.dependency_overrides = {}
```

```python
@pytest.fixture(name="patch", autouse=True)
def patch_fixture(fake_gpt, mocker):
    mocker.patch("myapp.gpt", fake_gpt)
```

Integration Tests

Make them Optional

```
@pytest.fixture(name="chat_gpt")
def chat_gpt_fixture():
    api_key = os.getenv("TEST_OPENAI_API_KEY")
    if not api_key:
        pytest.skip("TEST_OPENAI_API_KEY not set")
    return ChatGptClient(api_key=api_key)
```

Parametrized Fixtures

Testing Multiple Implementations

```python
@pytest.fixture(name="chat_gpt")
def chat_gpt_fixture():
    # ...


@pytest.fixture(name="fake_gpt")
def fake_gpt_fixture():
    # ...
```

```python
@pytest.fixture(name="chat_gpt")
def chat_gpt_fixture():
    # ...


@pytest.fixture(name="fake_gpt")
def fake_gpt_fixture():
    # ...
```

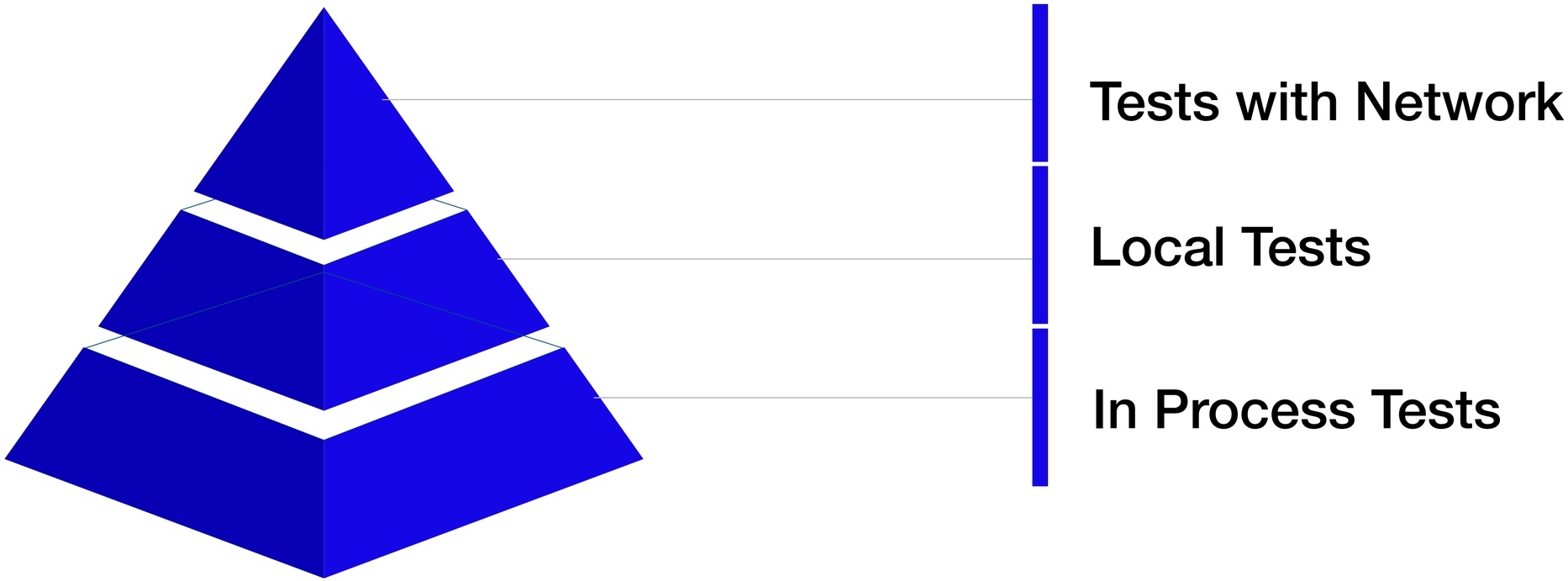
```python
@pytest.fixture(
    name="gpt", params=["fake_gpt", "chat_gpt"]
)
def gpt_fixture(request):
    return request.getfixturevalue(request.param)
```

```python
@pytest.fixture(
    name="gpt", params=["fake_gpt", "chat_gpt"]
)
def gpt_fixture(request):
    return request.getfixturevalue(request.param)
```

```python
def test_make_joke(gpt):
    joke = gpt.make_joke()
    assert is_funny(joke)
```

Practical Test Pyramid

Prerequisites, Speed, and Complexity



Pytest Design Patterns

Miloslav Pojman

pojman.cz/2024/pytest/

twitter.com/MiloslavPojman
fosstodon.org/@mila